Development of a new serious game utilising reinforcement learning to enrich high school mathematics

JAI le Roux

orcid.org/ 0009-0004-2566-8928

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North-West University

Supervisor: Prof PA van Vuuren

Graduation: June 2025

Abstract

A serious game is developed in this study to aid South African grade 9 learners with mastering algebraic mathematics. The game is a two-player zero-sum adversarial card game that uses cards to represent different mathematical symbols/objects. The second player is an AI opponent that relies on a machine learning technique known as reinforcement learning to learn how to play the game.

The AI opponent is able to "learn" basic mathematical concepts such as addition, subtraction, multiplication, division, and functions by playing the game against itself. A comparison between the SAC and PPO reinforcement learning algorithms showed that PPO outperforms SAC in the game. The inferior performance of SAC is attributed to the random elements of the card game that had adverse effects on the entropy-seeking characteristics of SAC.

The serious game was also tested with real learners at a school in Potchefstroom and Ikageng. The majority of learners did not proceed past the first level of the game and only played the game for around a week. The dire performance of the serious game is likely due to the mediocre video tutorials of the game. It is expected that learners' response to the game can be improved by enhancing the game tutorials, and incorporating the game into classes or extracurricular activities. Adding multiplayer functionality, where players duel each other, can also potentially increase player motivation. This source of motivation stems from the positive peer pressure among students.

Keywords: Serious game; High school mathematics; Reinforcement learning

Acknowledgements

Designing a video game is a daunting task that is full of insurmountable difficulties to overcome alone. It is a team effort, and I am grateful for everyone that helped, whether it was with positive encouragement or valid criticism. It is my sincerest apologies that I cannot mention every person on a single page. I highly appreciate the unsung contributions as well.

I am eternally grateful for the Lord, Jesus Christ, that placed me at the right place, at the right time to do *this* masters. The duration of the study was truly a season of isolation and many events occurred that cannot be written off as just a mere coincidence. The goodness of God cannot be denied.

I appreciate the support and guidance given by Prof Pieter. Prof Pieter had an open-door policy and was always available for a talk. Most importantly, Prof Pieter was never judgemental towards ideas and was open to considering alternative approaches. Prof Pieter gave me the freedom to experiment and allowed me to make mistakes and learn from them. There are few study leaders that provide one with these opportunities.

I treasure the company provided by *mister* Andreas Alberts and his generosity in lending his server for the experiment. Prof Alwyn Hoffman gave me a profound perspective change with his tour to the Eastern-Cape. Something I am forever thankful for. Prof Willie Venter provided valid advice and criticisms during the study, which definitely aided with the timelines of the study.

I am thankful for the schools who participated in the experiment and the service of the teachers and participants. The Xcel group at the NWU greatly helped to identify typical mistakes that learners make. I appreciate the MUST research group for openly sharing their knowledge by hosting boot-camps and weekly gatherings. They tremendously helped me with getting started on deep networks, although they likely do not even know it. Mr Hannes du Toit provided valuable networking opportunities and helped with identifying and approaching the Ikageng school. I appreciate Prof Roelof Coetzer's advice on the statistics of the experiment. Finally, I am grateful for the other M-students of the NWU that were open for discussions, sharing ideas and understanding frustrations.

Contents

1 Introduction **1**

1.1 Motivation for the study 2

1.2 Research questions 4

1.3 Overview of study's methodology 5

1.4 Chapter outline 6

2 Background **7**

2.1 Teaching mathematics 7

2.1.1 Epistemology 8

2.1.2 Pedagogy 10

2.1.3 Assessments 16

2.1.4 Summary of mathematical teaching 18

2.2 Serious game design 19

2.2.1 Definition of serious games 19

2.2.2 Game design fundamentals 21

2.2.3 Overview of existing mathematical games 30

2.2.4 Game classification 33

2.2.5 Summary of game design background 34

2.3 Reinforcement learning introduction 35

2.3.1 The RL framework 36
2.3.2 Model-free RL approaches 49
2.3.3 Multi-agent RL 68
2.3.4 Case studies of RL usage in games 71
2.4 Key conclusions from background 74

3 Game design **76**

3.1 Game design goals and requirements 76
3.2 Game genre 79
3.3 Core game loop 80
3.3.1 Principle gameplay 80
3.3.2 Gameplay with variables 87
3.4 Cards in the game 89
3.4.1 Operator cards 90
3.4.2 Mathematical object cards 91
3.4.3 Function cards 91
3.4.4 Question cards 94
3.5 Auxiliary game mechanics 97
3.6 Player retention mechanisms 102
3.6.1 Game theme 102
3.6.2 Game progression loop 102
3.6.3 Rewards 107
3.6.4 Notifications 110
3.7 Reflection and summary of game's design 111

4 RL agent design **113**

4.1 ML-Agents RL framework . . . 114
4.2 State, action and reward design . . . 116
4.2.1 State representation of mathematical functions . . . 116
4.2.2 Total game state representation . . . 122
4.2.3 Total game action representation . . . 124
4.2.4 Reward shaping . . . 125
4.2.5 State exposure . . . 126
4.3 RL algorithms used . . . 126
4.3.1 Proximal policy optimization . . . 127
4.3.2 Soft actor-critic . . . 129
4.4 Notable RL implementation difficulties . . . 131
4.5 RL design summary . . . 132

5 RL agent performance 133

5.1 Hyper parameter tuning and final agents . . . 133
5.2 Algorithm performance . . . 139
5.3 Tournament validation of agents . . . 141
5.4 Explainability of policies . . . 143
5.5 Summary of RL performance . . . 147

6 Response of school learners to the game 149

6.1 Experimental methodology . . . 150
6.2 Sampling procedure . . . 153
6.3 Experiment instruments . . . 155
6.3.1 TIMMS test . . . 155
6.3.2 Survey . . . 159

6.3.3 Passive game analytics . . . 159
6.4 Experiment results . . . 159
6.4.1 Test scores . . . 160
6.4.2 Games play time . . . 165
6.4.3 Gameplay difficulty . . . 170
6.4.4 Participant's feedback for future improvements . . . 173
6.5 Experiment conclusion . . . 177

7 Conclusion **179**

List of References **182**

A Algorithms used to answer and generate question cards **190**
A.1 Validating the player's answer . . . 190
A.2 Generating questions of question cards . . . 191
A.2.1 Helper functions . . . 191
A.2.2 Distractor generators . . . 192
A.2.3 Distractor usage . . . 195
A.2.4 Question generating functions . . . 197

B Cards in deck distributions **204**

C Game requirements fulfilled by game's design **206**

D ML-Agents hyperparameters **214**

E Game played by the control group **219**

F Adapted TIMSS test **222**

G Post test survey **229**

H Informed consent documentation **230**

I Ethical clearance **235**

J Software package versions **237**

K System requirements of the game **239**

L Specifications of computer used to train RL agents **240**

List of Figures

1.1 Number of learners writing the 11 most popular exams [1] 2

1.2 Percentage of learners that pass the 11 most popular subjects 3

2.1 Dual flow [5] 22

2.2 Illustration of game depth 29

2.3 Math is magic gameplay [47] 30

2.4 Questions in Math is magic (left) and illustration of spell book (right) [47] . 30

2.5 Clash of wizardry gameplay [48] 31

2.6 Mathy fight gameplay [41] 33

2.7 Illustration of the Markov decision process [6] 37

2.8 Backup diagram for the state-value function [6] 41

2.9 Illustration of generalized policy iteration [6] 42

2.10 Backup diagram for the action-value function [6] 42

2.11 Curse of dimensionality illustration 44

2.12 Backup diagram for Monte Carlo approach [6] 45

2.13 Overview of RL methods described in this study 50

2.14 Example action-value representation for tabular methods 50

2.15 Backup diagrams of SARSA and Q-learning [6] 53

2.16 Example action-value representation with approximation methods 56

2.17 Typical characteristics of a loss function 57

2.18 Example of an MLP neural network with two hidden layers 61

2.19 High-level illustration of the experience replay buffer 62

2.20 Example of policy gradients using discrete action probabilities for a 1D state 64

2.21 Example single agent performance measure - image obtained from [7] 68

2.22 Example ELO performance measure of MARL task - obtained from [55] . . 69

3.1 A starting state for the game 81

3.2 Playing a card on the left board 82

3.3 Effect of card played 83

3.4 Opponent's turn 83

3.5 Playing a number card 84

3.6 Options available when playing a question card 85

3.7 Question option selected 85

3.8 Correctly answered question card 86

3.9 Game over countdown 86

3.10 Game over summary 87

3.11 Example of a game with variables 87

3.12 Example of the battle round 88

3.13 Substituted battle round 89

3.14 Characteristics of a card 89

3.15 Function card in hand 92

3.16 Function card is played on the board 92

3.17 Result of function card played 93

3.18 Feedback for incorrect answers 94

3.19 Auxiliary mechanics in the game 97

3.20 Empty forge 98

3.21 Occupied forge 98

3.22 Crafted forge card 99

3.23 Graph mechanic 100

3.24 Simplify mechanic wrong 101

3.25 Simplify mechanic correct 101

3.26 Ladder mechanic in *Mortal Kombat II (1993)* 103

3.27 Game first level 104

3.28 Level progression 105

3.29 Another level of the game 105

3.30 Rewards summarized at the end of a won game 108

3.31 In-game animal rescued 108

3.32 Rescued animals in the main menu 109

3.33 Animal rescue catalogue 109

3.34 Game reminder notification 110

4.1 Framework of ML-Agents in this study 114

4.2 RequestDecision() overview 115

4.3 Example of an ordinary function 116

4.4 Example of hard knee and soft knee DRC [71] 118

4.5 Clamped DRC 118

4.6 Uniform sampling illustration 120

4.7 Variable non-uniform sampling illustration 120

4.8 Constant non-uniform sampling illustration 121

4.9 Agent state composition 123

4.10 Agent action composition 124

4.11 PPO's clipped objective [6] 128

5.1 ELO obtained by RL agents during hyperparameter turning 136

5.2 PPO hyperparameter importance 138

5.3 SAC hyperparameter importance 138

5.4 Parallel coordinates charts of hyperparameter tuning 139

5.5 Comparison of RL agents' ELO over training steps with tuned hyperparameters 140

5.6 Comparison of RL agents' ELO over longer training steps 141

5.7 RL tournament results 142

5.8 t-SNE plot of states visited by agents 144

5.9 t-SNE plot of agent actions with own visited states 145

5.10 t-SNE plot of agent actions with mixed visited states 146

5.11 Agent action distribution over training steps 147

6.1 Illustration of experiment methodology 152

6.2 Examples of question formats found in the digital test 158

6.3 Box plots of test scores at Ikageng school 161

6.4 Time taken to write tests at the Ikageng school 165

6.5 Player count over dates 166

6.6 Relationship between pre-test score and number of games played 167

6.7 Cumulative games played over dates by experimental group 168

6.8 Cumulative games played over dates by control group 168

6.9 Time of day when games were played 169

6.10 Response of experimental group to mobile notifications 170

6.11 Response of control group to mobile notifications 170

6.12 Level progress in experimental game 171

6.13 Level progress in control game 171

6.14 Game outcomes in experimental game 172

6.15 Game outcomes in control game 172

6.16 Experimental group's Likert responses in the survey 173

6.17 Control group's Likert responses in the survey 174

6.18 Relationship between personalization mechanic and games played of control games 176

6.19 Games played with internet access 176

A.1 Linear and quadratic equations 191

E.1 Number Nitro gameplay 220

E.2 Levels in control game 220

E.3 Number Nitro customization 221

E.4 Number Nitro main menu 221

G.1 Survey given to participants after the post test 229

List of Tables

3.1 List of operator cards 90

3.2 List of mathematical object cards 91

3.3 List of function cards 93

3.4 List of question cards 94

3.5 Level unlocks and opponent water supply amounts 106

4.1 Clamped DRC ranges used for agent in this study 119

5.1 Hyperparameter tuning statistics for 100 trials with 1M training steps each 135

5.2 Tuned hyperparameter values 137

6.1 Testing normality of Ikageng tests with the Shapiro–Wilk test (5% significance level) 162

6.2 Statistical differences between pre-test and post-test scores of Ikageng experimental group 162

6.3 Statistical differences between pre-test and post-test scores of Ikageng control group 163

6.4 Statistical differences between Ikageng experimental and control groups . . 164

6.5 Test scores of participants from Potchefstroom school 165

6.6 Experimental game tutorial view statistics 172

6.7 Experimental game animal rescue statistics 176

B.1 Blue deck card distribution at each level 204

B.2 Red deck card distribution at each level 204

B.3 Green deck card distribution at each level 205

C.1 Verification of game design requirements 206

D.1 Hyperparameters shared by PPO and SAC in Unity ML-Agents [58] 214

D.2 PPO hyperparameters in Unity ML-Agents framework [58] 216

D.3 SAC hyperparameters in Unity ML-Agents framework [58] 217

D.4 Self-play hyperparameters in Unity ML-Agents framework [58] 218

F.1 Adapted 2015 TIMSS test used in the study [84] 222

J.1 Python packages and versions used 238

List of Abbreviations

Below is a list of abbreviations that are used throughout the text.

AI	Artificial intelligence
ANN	Artificial neural network
DBE	Department of Basic Education
DRC	Dynamic range compression
GDD	Game design document
GX	Game experience
ICT	Information and Communication Technology
MARL	Multi-agent reinforcement learning
MDP	Markov decision process
ML	Machine learning
MSE	Mean squared error
NPC	Non-player character
NWU	North-West University
PPO	Proximal policy optimization
RL	Reinforcement learning
RPG	Role-playing game
SAC	Soft actor-critic
TD	Temporal-difference
TIMSS	Trends in International Mathematics and Science Study
UI	User interface

Chapter 1

Introduction

This study developed a serious game (called *Digit Draw*) for grade 9 learners who are taking mathematics in South Africa. Grade 9 is an influential year for a school learner since it is when learners make subject choices that will pave the path of their future career. One important choice to make is choosing between mathematics and mathematical literacy.

Digit Draw hopes to nurture a mathematical curiosity in learners that currently lack the intrinsic motivation to study mathematics. Ultimately, it is hoped that *Digit Draw* can persuade players to strongly consider taking regular mathematics in grade 10 instead of mathematical literacy. Players that are anxious about mathematics might enjoy the game's stress-free environment since there are no negative repercussions should a player fail. Players that aim to gain a deeper understanding in mathematics could possibly find the final levels of the game challenging and interesting. *Digit Draw* strives to be easy to learn, but harder to master.

The game can be played on any Android smartphone that adheres to the minimum specifications of Appendix K. It is assumed that players are already competent in using a smartphone that meets the requirements of Appendix K. The game does NOT require a mobile data connection to play.

A unique attribute of *Digit Draw* is its approach to teaching mathematics by playing against an adversarial AI opponent. The opponent is trained with a machine learning technique known as reinforcement learning (RL). The RL opponent learns how to play the game by aiming to maximize an expected reward. The opponent is rewarded for winning a game and penalized when losing a game. Machine learning RL is similar to the reinforcement learning pedagogy employed in teaching animals new tricks by rewarding them with treats. No strategies were hard-coded into the RL opponent.

The RL opponent was a major asset in the development and design of *Digit Draw* since the game has deep interconnected mechanics with various valid approaches to play. A single change in the game's design can have a cascading effect on other subsystems of the game. However, the RL opponent is easily re-trained after a game change in order to show any new exploits or bugs that the game change introduced. These breaking changes would be much harder to detect if the opponent was hard coded and unable to learn by itself.

There are four subsections in this introductory chapter. Section 1.1 establishes the motivation for this study. Section 1.2 proceeds to state the study's research questions together with their merit of being answered. An overview of the experimental methodology that is used to answer the research questions is provided in Section 1.3. Finally, a brief outline of each chapter in this dissertation is presented in Section 1.4.

1.1 Motivation for the study

Motivation for the study stemmed from the performance of grade 12 learners in the National Senior Certificate exam. The number of learners that wrote each of the 11 most popular matric subjects is given in Figure 1.1. The figure shows that there are around 1.6 times more learners taking mathematical literacy compared to regular mathematics. The lessened regular mathematics participation is problematic since certain career paths are completely closed to a learner when they choose to take mathematical literacy instead of regular mathematics.

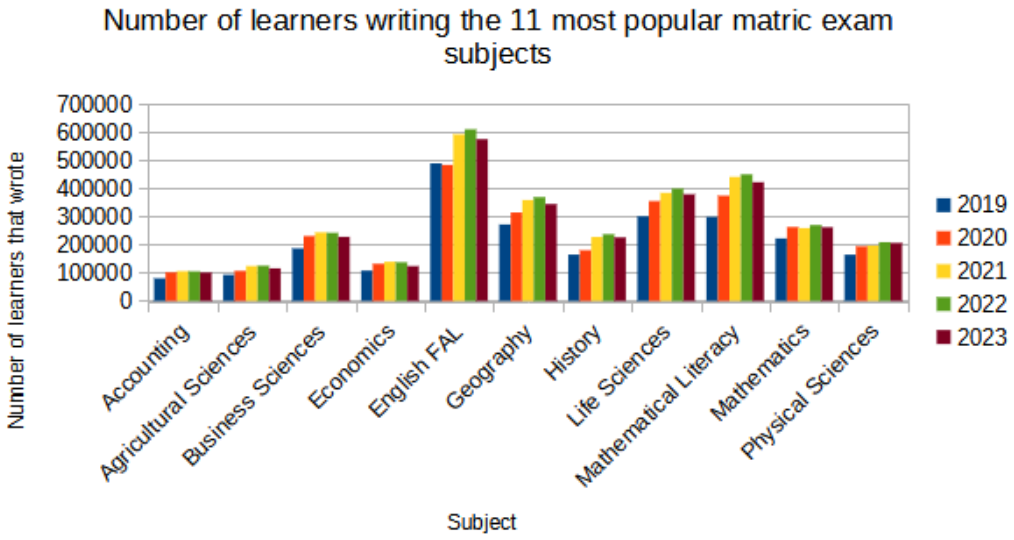


Figure 1.1: Number of learners writing the 11 most popular exams [1]

Moreover, Figure 1.2 proclaims that the regular mathematics exam is failed by approximately 40% of the students that write it. The failure statistic is even more drastic when one considers that a test candidate only has to obtain at least 30% in order to pass the exam. While these statistics are dire to observe, they at least provide hope that an intervention can contribute to a positive effect. South Africa's mathematics problem can therefore be considered a low-hanging fruit, where a little amount of effort has the potential to deliver a substantial impact.

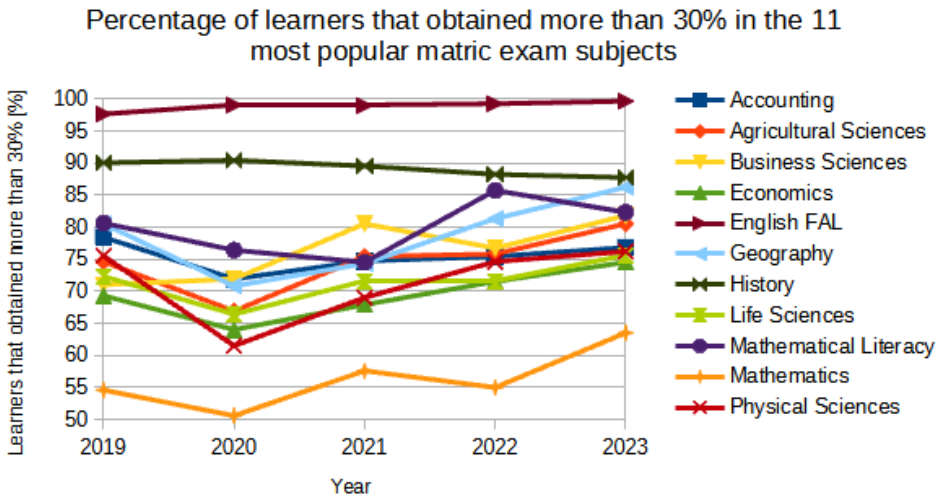


Figure 1.2: Percentage of learners that pass the 11 most popular subjects

Grade 9 learners were chosen as the target audience for the study since they are on the brink of making their subject choices. Section 2.2.3 of the dissertation presents a few case studies of existing mathematical games, where it is shown that most of the case study games cater to younger audiences. The abundance of pre-primary/primary school mathematical games makes sense since a strong mathematical foundation at a young age can be used to build new knowledge on top of. Therefore, a void exists in mathematical games that are aimed at older children. Another benefit of targeting grade 9 learners is the wide range of grade 9 specific standardized tests. These tests can be used to evaluate the effectiveness of the serious game. A discussion on standardized tests is presented in Section 6.3.1 of the dissertation.

The South African Department of Basic Education (DBE) further seems particularly interested in integrating Information and Communication Technology (ICT) within the classroom since it is part of their 2019-2023 research agenda [2]. The impact of using mathematical software in the computer labs of high schools in the EMDC East zone of Cape Town was investigated in 2010 by G. Spencer-Smith [3]. G. Spencer-Smith found in his research that the computer labs and mathematical software were likely unused, and that teachers experienced technical difficulties when attempting to conduct a class in the

computer labs. However, it is now 10+ years later and access to mobile smartphones has greatly improved. The increase in smartphone adoption means that teachers and students are likely familiar with using smartphones and that smartphones can replace the need for computer labs. A perspective change is just required to think of a smartphone as an educational device instead of a communication/entertainment device.

A reason for the lack of quality educational games produced by AAA gaming companies is primarily due to the amount of risks involved in making a game [4]. Making a quality game can consume plenty of time, which becomes a costly endeavor [4]. These companies are set on making a profit and would rather make a generic game that is known to sell well than innovate and risk a negative reaction from players [4].

Avoiding the risks of creating a serious game is completely justified since serious games have negative aspects associated with them. One negative aspect is that users might feel that they are tricked to do something unpleasant (manipulated) when playing the serious game [5]. The feeling of being manipulated alone can cause users to abandon the game entirely [5]. Furthermore, the costs of developing serious games are generally higher than traditional learning materials [5]. The increased development costs of serious games cause stakeholders to have increased expectations in order to justify the additional expenditure associated with games [5]. Thirdly, the time required to integrate serious games into existing corporate processes is cumbersome, since it can take at least six months before the integration process is finished [5].

The above-mentioned challenge makes for an exciting master project that aims to have a real world impact. The lack of resources, compared to AAA gaming companies, necessitates ingenuity (such as RL opponents) in order to develop a unique game in the short time span of two years. It is hoped that "the law of conservation of misery" is carried through and that the effort of this research can benefit the players of the game or other researchers with similar interests. The next subsection sets out the research questions that the study aims to answer.

1.2 Research questions

This study aims to answer the following four research questions:

1. Which reinforcement learning algorithm is more applicable to the serious game?
2. Does the serious game improve the algebraic mathematical abilities of grade 9 learners?
3. How long are grade 9 learners motivated to play the serious game?
4. How long does it take for grade 9 learners to master the serious game?

The first research question stems from the notion that different RL algorithms can have different decision preferences [6][7]. The difference in decision preferences causes the RL algorithms' performance to be task/problem dependent [6][7]. This study compares the soft actor-critic (SAC) and proximal policy optimization (PPO) RL algorithms to determine which one has the best performance (time to train and ELO score) in the serious game. The game of this study is an interesting environment for the algorithms to learn since it is a two-player card game that relies on randomization.

The last three research questions aim to establish the response of real grade 9 learners to the serious game. The second research question aids in determining if players can expect to have their algebraic mathematical abilities improved after playing the serious game. The third research question seeks to measure if learners are interested in the serious game and for how long they remain interested. The fourth and final research question investigates whether learners mastered the serious game and how long it takes them to complete the game and learn everything that the game has to offer. It is important that players play the serious game of this study over a long time span, not just an afternoon. The prolonged player engagement requirement is due to the teaching strategies employed by the serious game being known to produce slow initial results but anticipates greater benefits in the long term (see Section 2.1.2).

1.3 Overview of study's methodology

From Section 1.2, it can be seen that the study aims to answer one reinforcement learning question and three questions related to the response of playing the game. The reinforcement learning question is investigated in Chapter 5 by training RL agents and observing their ELO scores. The results of the ELO scores are verified by running a tournament where the agents play against each other and ensuring that the winner of the tournament corresponds with the ELO prediction. Explainability of the algorithms' performance is done by investigating t-SNE plots that present the general strategies taken by the RL algorithms.

Questions pertaining to learners' response to the serious game are answered in Chapter 6 by means of a field experiment that was conducted at a school in Potchefstroom and a school in Ikageng. Participants of the experiment had to write a pre-test, play the game for a month, and write a post-test. The scores of the pre-test and post-test are compared to determine if the game had any effect on the participants' mathematical abilities. Additionally, gameplay data was passively captured, and a survey was administered in order to gather insights into the results of the pre-test and post-test. Each school had a control group to ensure the validity of the field experiment against external threats.

1.4 Chapter outline

Finally, this section concludes the introductory chapter by presenting an overview of the remaining chapters that can be found in the dissertation. Chapter 2 is the longest chapter of this dissertation since it provides a background on the three core disciplines that the study is built upon, namely the teaching of mathematics, designing games, and using reinforcement learning as a machine learning technique. The teaching of mathematics and game design subsections are introductory and provide context as to why certain game design decisions were made. The reinforcement learning subsection assumes that the reader has some background in machine learning, but has never heard of reinforcement learning. A background in reinforcement learning is required to appreciate the design of the study's RL agent in Chapter 4 and to interpret the performance of the RL agent in Chapter 5.

Chapter 3 presents the serious game that was designed in this study. The chapter seems long when judged by its page count, but is an easy read due to the large number of gameplay images present. Chapter 4 continues to present the design of the game's RL agent together with a brief background on the SAC and PPO algorithms.

Results relating to the performance of the RL algorithms are given in Chapter 5, where it is determined which RL algorithm is superior to use in the serious game. The response of real school learners to the serious game is given in Chapter 6. Finally, a summary of the study and key conclusions drawn from the study are covered in Chapter 7.

Therefore, readers from different backgrounds may find only a subset of chapters applicable to them. Readers associating with the field of education will likely find Section 2.1, Chapter 3, and Chapter 6 applicable to them. Game designers can find key game design contributions of the study in Section 2.2, Chapter 3, and Chapter 6. Lastly, technical readers that are interested in the machine learning of the study can consult Section 2.3, Chapter 3, Chapter 4 and Chapter 5. Technical readers with a solid understanding of reinforcement learning may choose to gloss over or skip Section 2.3.

It should be noted that these chapters present the study as a linear process with a clear indication of how one conclusion led to another action. However, the design process of the study was highly non-linear and the results/findings in one chapter influenced the decisions made in another chapter, which ultimately had an effect on the original chapter that instigated the decisions. It is therefore expected and normal for a serious game's design process to be non-linear with no clear path forward. Sometimes progress can only be made when tackling the design from a different angle.

Chapter 2

Background

The background of this study is divided into three sections, namely the teaching of mathematics (Section 2.1), game design (Section 2.2), and an introduction to reinforcement learning in a machine learning context (Section 2.3). The sections on the teaching of mathematics and design of games intend to provide an introduction to the fields and aim to restrict the content only to what motivated design choices in this study.

The section covering reinforcement learning is aimed at technical readers who seek an introduction to the subject. The section assumes no prior reinforcement learning knowledge, but does assume familiarity with machine learning. The theory covered in the reinforcement learning section is needed to understand the design of the RL agent in Chapter 4.

2.1 Teaching mathematics

Teaching can be considered as the transference of knowledge from the teacher to the student. However, students might experience some difficulties in the learning process, known as learning obstacles [8].

The learning obstacles faced by students are broadly categorized as ontological, didactic, and epistemological learning obstacles [8]. Ontological obstacles are encountered when an individual is psychologically not yet ready for the learning topic, didactic learning obstacles are seen when an individual struggles with the method or approach of the learning materials, and epistemological learning obstacles are found when an individual has insufficient background knowledge of the topic being learned [8].

Ontological learning obstacles can be anticipated with the use of Piaget's theory. Piaget's theory sets forth the different developmental stages that individuals experience as they age [9]. The stages are known as the sensorimotor, pre-operational, concrete operations,

and formal operations stages [9][10]. The sensorimotor stage occurs in infancy, the pre-operational stage is experienced by pre-school children, the concrete operations stage is relevant to school-age children, and the formal operations stage appears throughout adolescence and adulthood [9][10]. Criticism faced by Piaget's theory includes the negligence of sociocultural aspects of learning, some individuals may experience the stages in a different order than suggested, the translation between stages may be continuous rather than discrete steps, and some individuals may never reach the formal operations stage [9][11].

Piaget's formal operations developmental stage is of most interest to this study since grade 9 students are likely embarking on this developmental stage and require assistance to master it. The main characteristic of the formal operation stage is individuals being able to think abstractly and hypothetically about higher-level analysis problems without the need for physical objects/representations to understand and think about concepts [9]. It will be seen in Chapter 3 that the abstract nature of the formal operations stage was a great influence in the game's design.

Furthermore, Knight et al. present a framework that can be used to address didactic and epistemological learning obstacles by considering the epistemology, pedagogy, and assessment of the subject being taught [12]. Epistemology refers to the philosophy of what knowledge is, pedagogy considers how knowledge is transferred and assessment represents the test of knowledge retained by an individual [12]. Kozulin and Presseisen support this framework by stating that learning activities should have a learning task, operations, and methods for evaluation and control [11]. It is important to note that epistemology, pedagogy, and assessments are interlinked and influence each other [12]. For example, the epistemology of a subject influences how the subject is presented (pedagogy) and the pedagogy influences the knowledge that is captured by the learners (the epistemology).

Consequently, the following subsections explore the epistemology, pedagogy, and assessment of high-school mathematics in greater detail due to the influence and impact that they have on each other and the design of the serious game.

2.1.1 Epistemology

Section 2.1 introduced epistemology as the philosophy of what knowledge is [12]. A variety of views exist on the philosophy of knowledge, with theories even hypothesizing whether knowledge is consumed, discovered, or created [12].

The two main views on mathematical knowledge are the absolutists (accepts only the maths given by an authority) and the fallibilists (views maths as a human process that can be further developed and changed) [13]. Mathematicians and mathematical teachers commonly hold contrasting views, with the mathematicians normally having a fallibilist view while the mathematical teachers normally having an absolutist view [13].

The South African Department of Basic Education (DBE) unifies these views by defining maths as a language and an activity [14]. The DBE states its definition as follows:

> Mathematics is a language that makes use of symbols and notations to describe numerical, geometric and graphical relationships. It is a human activity that involves observing, representing and investigating patterns and quantitative relationships in physical and social phenomena and between mathematical objects themselves. It helps to develop mental processes that enhance logical and critical thinking, accuracy and problem-solving that will contribute to decision-making [14].

Therefore, mathematics can be practised formally (absolutist), similar to a linguist studying a language, or mathematics can be applied and used informally (fallibilist), synonymous to vernacular language dialects.

The definition of the DBE further states that mathematics can be used to describe numerical, geometric, and graphical relationships [14]. The DBE's definition is consistent with the content found in a grade 9 South African textbook, with major topics being number systems, fractions, exponents, patterns, functions, algebraic expressions, equations, geometry, area calculations, volume calculations, straight line calculations, graphs, data analysis, and probability [15][16]. These topics provide a basis for succeeding grades (grade 10, 11 and 12) to be further expanded upon [17][18][19].

Ultimately, it is students' choice whether they accept and use mathematical knowledge, based on their own perspectives of the mathematical knowledge [13]. Students are still required to add the knowledge to their own mental structures (assimilated) and incorporate it into more general expressions [13]. The perspective that students have about the knowledge is known as their epistemic belief [12].

The epistemic beliefs of students are influenced by the certainty that the students have in the knowledge, the simplicity in which the knowledge is perceived, the source from which the knowledge comes, and the justification on why the knowledge is true [12]. Two methods that can aid in increasing the likelihood of learners incorporating knowledge into their epistemic belief are to make the knowledge applicable to real-life problems [13][20][3] and to encourage active learning [9][11][13][20][3][21][22]. Examples of active mathematical activities are counting, measuring, locating (entails navigating the "physical, socio-geographical and cosmological spaces"), designing, playing and explaining activities to the mathematics [23]. It is important to note that the activities should be mentally challenging and not just a repetition of the same activity. The next subsection discusses methods that can be used to transfer knowledge. The transference of knowledge is known as pedagogy.

2.1.2 Pedagogy

Pedagogy is presented in Section 2.1 as the study of learning and teaching [12]. This subsection discusses the pedagogy of mathematics by first distinguishing between the method and mode of teaching. Consideration will thereafter be given to the different pedagogies employed by mathematical teachers, with a section on the personalization of learning experiences following. A non-conventional teaching technique, known as desirable difficulties, is subsequently introduced and a concluding section on the importance of a mediator and scaffolding in overcoming the desirable difficulties will finalize this subsection.

Methods and mode

The teaching of mathematics consists of the method and mode in which it is taught [21]. The method describes how the mathematical content is presented to the students, while the mode describes the order in which the mathematical content is presented and developed (such as teaching addition before subtraction) [21].

The eagerness that a student has for discovery is in itself a method, known as the heuristic method [21]. The heuristic method enables students to discover mathematical concepts as if they were the founders of the concept [21]. The teacher's role is to ensure that the students have the necessary guidance to be able to discover the mathematical concepts [21]. Students should therefore still feel that they discovered the concept on their own [21]. The heuristic method is known to have slow initial results, but anticipates improved results in the long term [21]. This delayed gratification is a common theme of desirable difficulties found in Section 2.1.2.

Another entrancing method is the laboratory method, which uses experiments, estimations, approximations, observations, induction and common sense to aid students in building an intuitive understanding of the coursework [21]. The laboratory method is especially useful for providing the foundation that is needed to understand more abstract content [21][20]. Teachers are expected to intervene when students make incorrect conclusions about the phenomena of experiments since these conclusions form part of the foundational knowledge [21]. The teacher should further ensure that students do not mindlessly follow the template of the experiment and understand what they are doing [20]. Young suggests that students should use the laboratory experiments to gain intuition about the subject while drill exercises can aid the students in developing mathematical process skills since the laboratory method struggles to develop mathematical processes [21]. Lastly, the laboratory method has, likewise to the heuristic method, the undesirable characteristics of being time-consuming and instigating plenty of noise and emotions from the students [20]. These characteristics of the laboratory method may limit the settings in which it can be applied [20].

Other notable methods include teachers modelling how to solve a problem, coaching students on how to solve specific problems, providing scaffolding and fading (discussed in Section 2.1.2), questioning students on why they perform certain operations, encouraging students to articulate their reasoning, motivating exploration, inducing students to reflect on tasks done, providing cognitive tasks structuring, managing the performance feedback given to students, and using direct instructions where tasks are unclear [3].

As a final note, peer-based methods (collaborative learning) can be considered for improving subject motivation, creating autonomy in the acquisition and use of knowledge, enabling self-reflection of one's own knowledge and fostering self-confidence of students [24]. Competition between groups of peers further aids in motivating learners to master the subject's content [20]. Peer tutoring can be applied by assigning a student as the educator of a concept while the teacher is the backup expert advisor [24]. Peer tutoring has the added advantages of students explaining concepts in the terminology that they understand and enabling them to notice gaps in their knowledge while they are explaining the concept (self-reflection) [24].

The pedagogy of mathematical teachers

Section 2.1.1 mentions that high school mathematics teachers usually hold an absolutist view of mathematics, meaning that they only accept the mathematics given by an authority [13]. However, various teachers still have different perspectives on the pedagogy of mathematics. This section considers the different pedagogy methods employed by teachers, such as the instructionalist, response-strengthening, constructivist, connectivist, and empirical methods.

The instructionalist/transactional/information acquisition/transmission approaches perceive students as passive learners that must absorb the content given by the teacher [12][25][13]. The instructionalist approach requires students to only replicate algorithms as they were taught by the teacher, without adding much thought in the process [13]. Improving test scores is the main driver for the instructionalist method [12], even though it might not represent the true knowledge of the student on the subject.

A related view is the response-strengthening/reinforcement learning view [25]. The response-strengthening view uses a reward-based system where the teacher provides rewards to the students that perform well in tasks and punishments to students that perform poorly in tasks [25]. Students passively accept rewards (which strengthens the response used by the student to complete the task) and punishments (which weakens the response used by the student to complete the task) [12]. Rewards and punishments are the same technique used to train animals to do tricks, such as giving dogs treats when they do a trick. The response strengthening technique is paramount to this study since it is the same principle

used to teach the AI opponent on how to do mathematics/play the game. A discussion on the response strengthening/reinforcement learning technique therefore continues in Section 2.3.

The constructivist perspective is more of a fallibilistic view, where the students explore and conduct experiments in order to build a mental representation of the relevant information [25][12]. Students' constructed mental representation can then be integrated into their existing knowledge [25][12]. The teacher fulfills the role of a cognitive guide while the students are the active sense makers [25].

The connectivist approach entails active learning, often with real-life experiences and learner-to-learner interaction so that new knowledge can be gained and integrated (connect) with existing knowledge [12][13]. Connectivists embrace conceptual understanding opposed to procedural knowledge [13].

Finally, the empirical view uses practical approaches (such as investigations, experiments and discovery) to learn knowledge from real-life events [13]. The learners have a restricted space for discovery, since the teacher is in control of the process [13].

The variety of pedagogy styles employed by teachers might be attributed to the difficulty in applying them. For instance, previous sections revealed that self-discovery and open-ended questions contribute to the development of creative abilities and adaptive capabilities of students [20]. However, it can become difficult for teachers to assist with the unique responses that students have when they answer open-ended questions, since each answer requires a different interpretation [13]. Teachers may also experience difficulties with insufficient training on scaffolding techniques (see subsection on mediation and scaffolding below) as well as class sizes that are too large [3]. It is crucial to address these difficulties since the ultimate aim of schools should NOT be to drill proofs and equations into the students, but rather to provide the students with the necessary tools that enable independent learning [26].

Computers can assist with these difficulties by giving individualized instructions, enabling learners to learn at their own pace, providing catchup functionality for missed lessons, being available at any time, having infinite patience with struggling learners, and providing immediate feedback [3]. The advantages of using computers are influenced by the pedagogy used and not just the mere fact that it is done on a computer [12]. Subsequently, it has been found that using computers as a drill tool to teach mathematics is inefficient [3][27]. The cause for the ineffectiveness of drill tools is likely the reoccurrence of the same rules and methodology, which becomes uninteresting and ineffective after a while [20]. Similarly, it's also shown in Section 2.2.2 that video games become boring/uninteresting once the pattern has been found by the player.

Previous studies warn that students and teachers may struggle to adapt to new technology, especially if it is rarely used [3][27]. The unfamiliarity of using new technology in the classroom emphasizes that teachers should not be replaced by computers [3], and that didactic materials must be tested before production to discover any difficulties that the students may experience with it [8]. The following subsection shows techniques that can be used to provide personalized learning experiences to aid students in adapting to the didactics.

Personalized learning experiences

Students' learning experience can be personalized with the Grasha-Riechman model [28]. The Grasha-Riechman model is based on a student's traits and includes the competitive, collaborative, avoidant, participant, dependent, and independent learning styles [28].

A competitive learning style encapsulates students that desire to perform better than other students in tasks (gain rewards and appreciation) while a collaborative learning style enjoys sharing their ideas and work with teachers and fellow students (emphasis is on freedom and discussion) [28]. An avoidant learning style is characterized by students who do not participate in class activities due to a lack of interest and enthusiasm for learning [28]. In contrast, a participant learning style enjoys learning and appreciates responsibilities that are coupled with class activities [28]. Students that rely on teachers and other students for learning activities make use of a dependent learning style, while students that have self-confidence in their abilities and a preference for individual tasks are known as independent learners [29]. A stark resemblance can be found between these learning styles and types of video game players that are given by Bartle's player model (see Section 2.2.2).

Methods such as just-in-time (JIT) teaching can personalize a student's learning experience since JIT classes are orientated towards a student's needs [13][22]. JIT caters for a student's needs by providing didactic material before a class for the students to complete. The response of the students on the didactic materials is used to adjust the next class session according to the students' needs [22]. The outcome of a class can again be used to adjust the succeeding didactic materials developed for the following class. In essence, just-in-time teaching entails three objectives, namely allocating time outside of class to ensure maximum learning, increasing the efficient use of class time (since students already have a background of the class materials from the didactics), and creating and maintaining team spirit (motivation) [22]. A serious game seems to appropriately fit the role of a JIT didactic. While this subsection explored the diverse learning styles to which the pedagogy can be personalised, the next subsection introduced the concept of desirable difficulties. Desirable difficulties oppose effortless learning with the hope of increased long-term benefits.

Desirable difficulties

Desirable difficulties is a strategy used to make learning effortful in order to improve the retention, retrieval and transference of knowledge in the long term [26][30]. Desirable difficulties are strategically implemented, since not all difficult tasks yield desirable results [26]. Difficulties are undesirable when they are too difficult for a student to complete and cause the student to become demotivated and give up. Luckily, E. Bjork and R. Bjork present four methods which can be used to implement desirable difficulties, namely the variation of conditions of practice, spacing study content, interleaving study tasks, and using tests rather than presentations to learn concepts [26].

Varying the conditions of practice refers to changing the conditions under which an individual learns. A change in the learning location or even the fonts that the learning materials use are examples of varying the conditions of practice [26][31]. Varying the conditions of practice reduces the chance of students being only familiar with the layout of the learning material and helps students to learn the deeper meaning of the material [26]. The serious game of this study ensures that games are varied by using input randomization (see Section 2.2.2) to generate random questions instead of using a test bank of pre-generated questions.

Spacing refers to learning a module's contents over a longer time span to improve long-term retention [26]. Unfortunately, the prevalent strategy among students is to start learning the night before the test [26]. The practice of studying a large amount of information in a short time span is known as the cramming method [26]. Students using the cramming method often find themselves unable to recall the study content after a longer period of time, which makes it difficult to build new knowledge on top of prior knowledge [26]. Long-term memory is improved when the spacing strategy is used instead [26]. Spacing is implemented in the serious game by using levels to incrementally introduce the game's features, which forces the spacing of content (see Section 3.6.2).

Interleaving study content occurs when the study content is learned in a shuffled order instead of being grouped into specific categories (blocking) before they are learned [26]. For example, it would be better for students to learn differentiation when the differentiation rules are mixed instead of being separated apart. Students will initially take longer to learn the differentiation rules with the interleaving strategy, but they will likely have an improved long-term memory of the rules when compared to the blocking strategy [26]. The study's serious game uses decks and cards to shuffle game content (see Appendix B). The deck's card distribution and card drawn determine the type of mathematical concept that is practiced.

The improved long-term retention attributed to the interleaving of the study content is theorized to be the result of higher-order mental representations formed by the students

when they search for the similarities and differences in the study content [26]. These higher-order mental representations assist the students in connecting the different concepts of the study content [26]. Another theory suggests that interleaving of study content refreshes the students' memory more often, which may aid in improved memory retention [26].

Lastly, using tests instead of presentations can increase long-term memory retention [26]. The reason being is that tests cause self-reflection on the student's behalf as well as forcing the student to actively participate in learning activities by generating answers instead of passively accepting answers, solutions, or procedures from the teacher [26]. The serious game of this study makes use of tests when students have to answer question cards (see Section 3.4.4).

It must be emphasized that the desirable difficulties mentioned in this section will produce slow initial results with the aspiration of improving long-term retention, retrieval and transference of knowledge [26]. The succeeding subsection investigates how a mediator can aid in overcoming difficulties that are undesirable.

Mediation and scaffolding

It was stated in the previous section that desirable difficulties should be within the capabilities of the student. Ensuring that desirable difficulties are within the capabilities of the student can be accomplished with mediated learning, where an adult or other students place themselves between a struggling student and the learning environment in order to select, change, amplify and interpret objects for the struggling student so that the learning material is within the capabilities of the struggling student [11][3]. Three essential criteria for mediated learning include intentional interactive activities, transcending knowledge to other situations/tasks, and providing meaning on why the task is done [32][11].

Mediation learning can be analysed by means of the Vygotskian theory, which states that students first learn a concept socially (from the outside) and later as an independent problem solver (internally) [3][11]. The mediation process differs between individuals since it depends on an individual's zone of proximal development [3]. The zone of proximal development consists of two regions, with the first region representing the abilities of an individual using the help of others, and the second region representing what an individual can independently accomplish [3]. The purpose of the mediator is to transfer an individual from the region of being dependent on external assistance to the region of independence [3]. Typical traits of the zone of proximal development include establishing the level of difficulty, providing assisted performance to the student and evaluating the independent performance of the student [33][3].

Mediators mentioned by the Vygotskian theory are not necessarily humans, and may even be material or psychological tools [11]. The reinforcement learning opponent of the serious game (see Section 2.3) can be a possible mediator by acting as a role model for the player on how the game can be played and strategies that are possible. However, the Vygotskian theory does not indicate how the mediator must select, change, or amplify the learning material so that it is within the competencies of the student [3].

Therefore, Neo-Vygotskians established the concept of scaffolding to explain how a mediator can select, change, and amplify the learning material so that it is within the competencies of the student [3]. Examples include attracting the student's interest (recruiting), simplifying the task by reducing the number of choices to be made (reducing degrees of freedom), directing the student back to the original objective (direction maintenance), accentuating critical aspects of the problem (marking critical features), ensuring that the student does not become frustrated and demotivated (frustration control), and showing how the ideal/model solution process looks (demonstration) [34].

To conclude, scaffolding should not be used with the intent to make tasks easier, but rather to ensure that students can complete the tasks successfully [8]. Scaffolding methods are unique to a specific student and should be adjusted to the student's needs (student-centric) [22][20]. Also, the interaction between an individual and the environment (gaining experience) is never immediate and needs to be mediated by elements outside the individual [11]. The delayed interaction between an individual and the environment means that it can take a while before an individual gains the experience required for the assimilation of knowledge.

2.1.3 Assessments

This is the final section covering the framework of Knight *et al.* on teaching mathematics. Assessments are an important part of the learning experience, since the memory of learners can be enhanced through testing [29]. Passively rereading material (like a tape recorder) is not as effective as using tests as a learning method [26]. Assessments should subsequently aim to uncover the learning potential of students instead of just indicating their current level of understanding [11].

Section 2.1 mentioned that the epistemology, pedagogy, and assessments of a subject are interlinked and highly influence each other [12]. It is therefore no coincidence that the National Senior Certificate mathematics exam is divided into two question papers [35], since each paper requires a different thought process. The geometry paper (Paper 2) focuses on the synthetic method where the student solves a problem from the known to the unknown [21]. The synthetic method requires the student to use known geometry theorems to gain more information about the problem, with succeeding theorems built

on top of preceding theorems, until the problem is solved [21]. Conversely, the algebra paper (Paper 1) uses the analytic method to proceed from the unknown to the known by breaking an unknown problem into smaller sub-problems with known solutions [21]. Mastering the analytic and synthetic methods is equally important since the analytic method is normally used to solve mathematical problems while the synthetic method is used to present and share mathematical results/proofs [21]. Hence, it is important to apply diligence with assessments, since the appropriateness of an assessment is determined by the epistemology and pedagogy of a subject.

E. Bjork and R. Bjork reckon that measurements considering only the current performance of a student are an unreliable indicator of learning since a student's familiarity with the study material can easily be confused with actual learning [26]. Another point of concern is that assessments can fail to represent everyday problems, which too means that assessments fail to represent what it means to know [12]. Therefore, the grade that a student achieves can falsely represent the student's mastery of the study content if the topic is incorrectly assessed [12].

Consequently, assessments are often accompanied by feedback to help the students "confirm, add to, overwrite, tune, or restructure information in memory, whether that information is domain knowledge, meta-cognitive knowledge, beliefs about self and tasks, or cognitive tactics and strategies" [36]. Roussouw et al. further add that mathematical activities can lead to mathematical insight when reflective abstraction takes place [13]. Reflective abstraction means that students must think and reflect on the feedback that they receive from assessments in order to gain insight from the assessments. Other factors impacting the effectiveness of feedback can be the clarity of the learning goal, the relevance of the feedback to the task, and the feedback's ability to modify the behaviour of the learner [37].

Feedback received and the student's perceived likelihood of success can also reduce maths anxiety [27]. Mathematical anxiety is stated by Tatar as being the anxiousness, tension and anxiety that individuals feel when they are solving mathematical problems, whether it is in daily life or school contexts [28]. These mathematical problems are perceived as being threatening to the individual's self-esteem [28]. The effects of maths anxiety commonly cause a student to receive grades that are lower than expected, having greater pre-exam anxiety, and lower ratings of preparedness in maths courses [38]. The decrease in test performance, due to maths anxiety, can be alleviated with assessment that does not have any time constraints [38]. Subsequently, the serious game of this study doesn't have any time constraints imposed on the player. Students doing group work may also have a reduction in maths anxiety since Tatar found a significant negative correlation between a collaborative learning style and maths anxiety [28]. It is important for students to engage in mathematical class activities since an avoidant learning style (see Section 2.1.2) is the strongest learning style predictor for mathematical anxiety [28].

Fortunately, digital computers can aid in reducing maths anxiety by providing an environment where students are able to have a high success/failure rate when answering questions, without being exposed to any negative consequences [27]. However, digital computers do not necessarily improve the mathematics achievements of students when the computer is only used as a drill tool, since students can use tactics such as "impulsive clicking, trial-and-error or guessing" to obtain immediate feedback from the drill tool [3][27]. Students using these effortless tactics may therefore fail to reflect and learn from their mistakes when they brute-force the questions of drill tools.

Lastly, Mevarech and Ben-Artzi found that personalized feedback by a computer does not make any significant difference to students' mathematical anxiety and achievements when compared to regular (non-personal) feedback [27]. The analogous mathematical anxiety levels between personalized and non-personalized feedback suggest that simple feedback (indicating whether an answer is correct or wrong) is just as effective as accompanying the answer with personalized motivational messages. The serious game of this study does not provide personalized feedback and only indicates that an answer is correct or wrong when playing a question card.

2.1.4 Summary of mathematical teaching

This section presents a framework that can be used to teach mathematics. The framework consists of three elements, namely the epistemology, pedagogy, and assessment of mathematics.

The DBE presents the epistemology of mathematics as a language using symbols and notations to describe numerical, geometric, and graphical relationships. The perception of mathematical knowledge is divided, with some holding an absolutist view and others holding a fallibilistic view. The epistemology of the grade 9 curriculum can be categorized into two sections, namely algebra and geometry. Algebra was found to benefit more from analytical reasoning, while geometry focused on synthetic reasoning. Piaget's theory further indicated that an emphasis must be placed on abstract concepts since grade 9 learners are likely developing in the formal operations phase.

The section on the pedagogy of mathematics illustrated that there are various methods employed by teachers to teach mathematics. Some of the popular methods include instructionalist, response-strengthening, constructivist, connectivist, and empirical methods. The response-strengthening method (also known as reinforcement learning) is very important to this study since it is used to train the AI opponent of the serious game (see Section 2.3).

The constructivist and connectivist methods yield open-ended responses from students, which aids students in developing creative and adaptive capabilities. These open-ended

responses are often difficult for teachers to assist with, since each answer requires a different interpretation from the teacher. Computers can aid teachers with these individualized types of instruction.

The pedagogy section presented a non-conventional teaching method, known as desirable difficulties. Desirable difficulties utilises strategies such as the variation of conditions of practice, spacing study content, interleaving study tasks, and using tests rather than presentations to make learning effortful. Desirable difficulties are known to show slow initial results with the aspiration of improving long-term retention, retrieval and transference of knowledge. Mediation and scaffolding can otherwise be used to ensure that the desirable difficulties are within the students' independent zone of proximal development.

Finally, assessments are interlinked with the epistemology and pedagogy of a subject and have an influence on them. Assessments should be carefully constructed in order to correctly evaluate students' mastery of a subject. Mathematical anxiety related to assessments can be alleviated when the assessment does not have any time constraints. Digital computers also prove useful in reducing mathematical anxiety since they provide a safe environment for students to have a high success/failure rate when answering questions. Caution should just be taken with digital assessments to ensure that the assessment still requires active participation from students and that students can't use impulsive clicking, trial-and-error or guessing to obtain immediate feedback.

2.2 Serious game design

Background related to serious games and game design can be found in this section. This section is divided into four subsections, with the first subsection defining serious games and their typical characteristics. The second subsection considers core game design fundamentals that are found in well-designed games. The third subsection provides case studies of existing mathematical serious games, while the final subsection mentions how game theory can be used to classify games. A summary of this section can be found at the end of the section. The information presented in this section will prove useful when designing the serious game (Chapter 3) and choosing a reinforcement learning strategy (Chapter 4).

2.2.1 Definition of serious games

Firstly, the difference between playing and gaming is an important distinction to make. Playing is defined as "purposeless, intrinsically motivated human activity without explicit rules", while gaming is defined the same as playing, with the addition of having explicitly defined rules [5]. Therefore, a game's essence is its rules. The importance of rules defining a game is accentuated by games such as *Spades* and *Hearts* that have the same fundamental

gameplay (both are trick-taking card games), but have different rules on how the score is calculated [39]. The main difference in score calculation is the benefit of playing tricks in *Spades* while tricks must be avoided in *Hearts* [39]. This change in score calculation leads to radically different gameplay, even though the rules underwent minor alterations.

Serious games can then be defined as "a digital game created with the intention to entertain and to achieve at least one additional goal" [5]. This is the definition that will be used in this study, although other definitions also exist [5][40]. The additional goal is often called the "characterizing goal" [5]. Examples of characterizing goals could be to improve cognitive and perceptual skills, emotional and volitional skills, sensory-motor skills, personal skills, social skills, and media skills of individuals [5]. The characterizing goal of this study is to improve the mathematical cognitive skills of high school learners.

However, serious games should not be confused with other gaming-related terms such as gamification, games with a purpose, and entertainment games. Gamification represents the addition of game elements to a non-gaming context, while games with a purpose require many people to achieve a non-gaming purpose with the use of gaming elements [5]. Entertainment games are digital games that only wish to entertain the player [5]. Entertainment games differ from serious games by having no characterizing goal [5].

The uniqueness of a serious game is attributed to the domain experts that are involved in the game's development and the characterizing goal that complements the entertainment aspect of the game [5]. Popular categories for serious games are corporate games (used for training and simulations), educational games, health games, advergames (advertising games), social awareness games, architecture and planning games, and tourism and cultural heritage games [5]. The game of this study is classified as an educational game since its characterizing goal is to assist high school learners with mathematics.

Common themes found in serious games include backstories, realism, artificial intelligence (AI), interactions (with the game world and between players), and feedback and debriefing [40]. Backstories and narratives are useful in providing a sense of progression and curiosity in the games, while realism enables the player to become immersed in the game [40]. AI refers to "seemingly intelligent or unscripted game response to player activity" [40]. Uses for AI include personalized non-player character (NPC) actions, user modeling for personalized content, personalized difficulty adjustment, and adaptive feedback [40]. Adaptive feedback is usually based on the time required for answers, player roaming the game world without purpose and players repeating the same mistakes [40]. Feedback and debriefing elements enable players to understand cause and effect relationships in a risk-free environment [40]. Feedback and debriefing elements of serious games should be used in conjunction with teachers and not replace the teachers [40].

Key serious game design guidelines include centering the game design around the player, undergoing iterative game development, making use of interdisciplinary teamwork (artists,

subject experts, marketing), and integrating play and learning [5]. Ravyse et al. add that serious games should follow principles which ensure that nothing happens without player input, earlier problems provide scaffolding for later problems, enabling players to rethink their goals and regularly providing new classes of problems to the player [40]. Other popular elements used in game design include levels, feedback, rewards, progress indicators, points, medals, narratives, leaderboards and avatars [41]. Mediating the player's learning and gaming experience with debriefing and progress indicators is especially emphasized by Ravyse *et al.* [40].

Ultimately, a measure of a serious game's success is when a player wants to play the game (engage with the learning material) instead of doing something else that is fun [40]. This voluntary engagement is achieved by games that have quality game experience (GX) designs. GX encapsulates terms such as "fun, challenge, flow, immersion, presence, tension, positive and negative emotions, curiosity, fantasy, self-efficacy, and motivation" [5]. The next subsection investigates some game design fundamentals that can be used to improve a game's GX.

2.2.2 Game design fundamentals

The former section mentioned that players are more likely to engage with a serious game when the game experience (GX) is well-designed. A discussion on GX concepts is provided in this subsection. More specifically, concepts related to game flow, what makes a game fun, different types of players that will play the game, game mechanics, luck versus skill, deep games, and player friction will be discussed. Chapter 3 applies these concepts in the game's detailed design.

The flow state

Game flow describes the player's experience when a game has the player's full concentration and the player feels in control of the game [5]. The player is able to achieve clear goals with immediate and consistent feedback [5]. Other characteristics of game flow include the merging of action and awareness, loss of self-consciousness, time distortion, and a self-rewarding experience [5].

A closely related term, unique to serious games, is dual flow [5]. Dual flow aims to strike a balance between the difficulty of a game and the skill of a player [5]. The purpose of dual flow is to develop a serious game that is both effective at reaching its characterizing goal as well as being attractive to play [5]. Figure 2.1 illustrates dual flow by graphing the relationship between a player's skill level and the difficulty of a task. It can be observed from Figure 2.1 that a serious game's efficiency at reaching the characterizing goal is

inversely proportional to the flow experienced by the player. Therefore, easier serious games might ensure a more captivating experience, but at the expense of the player not reaching the characterizing goal as efficiently.

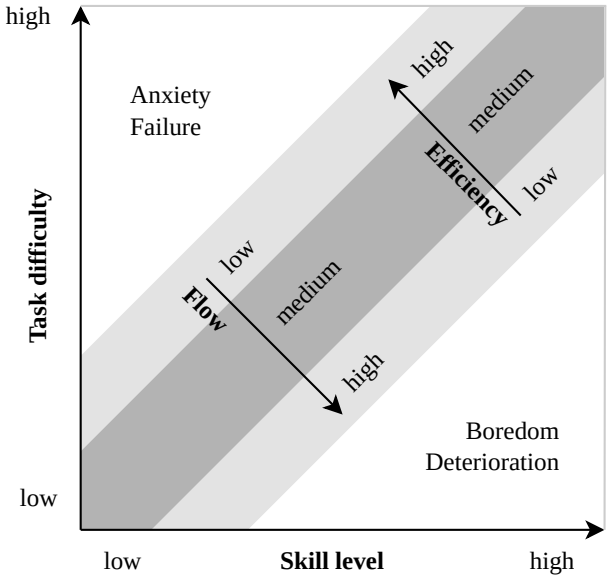


Figure 2.1: Dual flow [5]

Hence, dual flow is very similar to the desirable difficulties of Section 2.1.2, where too easy games bore the player and too hard games frustrate the player. It is important, from the perspective of desirable difficulties and dual flow, to keep players in the flow state and to not interrupt them with distracting game prompts [5][40]. Game flow can be preserved by implementing immediate feedback coupled with a minimalistic and intuitive game interface and control designs [40].

Motivation and fun

Children often find mathematics abstract, boring and unconnected to the real world [42]. Motivation to perform an activity can be classified as being intrinsically motivated, extrinsically motivated or in between [5]. Extrinsic motivation is often fostered by serious games, since the motivation is provided by an external source, while intrinsic motivation refers to the motivation that stems from within the learner self [5]. Extrinsic motivation provided by serious games can aid learners in engaging with activities where they lack intrinsic motivation [5].

Engagement factors used by serious games can be categorized as belonging to initial engagements, ongoing engagements or the engagement outcome [42]. Initial engagements constitute clear goals and thematic/visual appeal while ongoing engagement is characterized by feedback, rewards, social interactions, creativity of the game, and challenges [42]. The engagement outcome is met once the player experiences immersion [42]. It can subsequently be reasoned that the engagement outcome is reached when the player is in the flow state (described in the previous subsection).

Rewards used in ongoing engagement can be game items or features that encourage certain in-game behaviours [43]. Three fundamental behaviours that rewards elicit from players include encouraging players to return to the game, inducing consummatory behaviour when rewards are obtained, and promoting feelings of pleasure and positive emotions [43]. Common reward mechanisms used by serious games include discovering hidden secrets, visual rewards (skins), in-game resources, negative rewards (discourages behaviour), in-game items, achievements, unlocking mechanisms, and random rewards (loot boxes) [43]. Rønningen *et al.* emphasizes that users must know exactly what they did to obtain the reward so that they know what must be done for another reward [43]. These reward mechanisms are therefore reminiscent of the response strengthening pedagogy (Section 2.1.2) and the reinforcement learning method (Section 2.3).

Furthermore, players are more likely to engage in activities that they find fun. Lazzaro's four keys to fun mention the four types of fun that players experience, namely hard fun, easy fun, serious fun and people fun [4]. Hard fun is portrayed by players that receive an adrenaline rush during a difficult part of a game, while easy fun is present in players experiencing curiosity, surprise, awe, creativity and wonder [4]. Serious fun is associated with relaxation, rhythm, repetition and collection [4]. Finally, people fun consists of interacting and sharing experiences with other players/people [4]. Lazzaro's keys to fun state that a fun game requires at least three of the four types to be present in the game experience [4].

Lastly, Koster explains that the fun of games lies in the learning experiences of playing the game [44]. Players are masters at subconsciously learning patterns (also referred to as rules) of the game while they are playing it [44]. The game eventually bores the player (lost its fun) when the player has mastered the game's patterns and is able to play without adding much thought to it [44]. Games can also become boring when its patterns are too hard/easy or take too long for the player to identify [44].

Boredom in games should not be confused with familiarity since serious fun is associated with familiarity [4][5]. Players crave games that are similar to those that they are already familiar with [4][5]. Rather, boredom here refers to the mindless repetition of actions, like using muscle memory to play a game without needing to think [44].

Uniqueness of players

Factors specific to a player, such as age and gender, might influence a player's perception of a game [5]. Subsequently, Bartle's player model can be used to classify players into four distinct groups based on their psychological model [5]. It is important to note that a player can be a member of multiple groups and is not restricted to a particular Bartle player model group [5].

The four groups of Bartle's player model are known as achievers, killers, explorers, and socializers [5]. Achievers are players that want to beat every activity that the game has to offer, while killers find enjoyment in beating other players and becoming the overall winner [5]. Explorers are players that like to interact with the game world and explore every aspect of the game, such as hidden areas and Easter eggs [5]. Socializers are characterized by players who find pleasure in talking, chatting and interacting with other players [5].

Techniques and methods commonly employed to adapt to a player include procedural level and content generation, changing agents' (NPC) behavior, changing the story/narrative of the game, providing guidance and hints, motivating the player, and changing the presentation/sequence of the game [5]. Indicators frequently used for adaptivity are the game score, task completion rate, number of times the task has been completed, time spent on a task, the success rate of a task, player progress in the game world, and incongruent behaviours (the likelihood that the player achieved a goal with luck) [5].

A players' gameplay experience can be measured with physiological methods (electroencephalography, electromyography, electrodermal activity, or the heart rate of the player), psychological methods (persona models, player models, surveys and questionnaires, verbal reports, interviews, or saying thoughts aloud), and behavioural models (eye tracking, game logs, reaction time, observations of the player/gameplay, or video recordings of the player/gameplay) [5]. This data can then be used to improve the experience of the player.

Game mechanics

Game mechanics refer to the ways in which the player can interact with a game, given the game's rules and a specific situation [5]. Shell's taxonomy of game mechanics can be used to classify a game mechanic into one of six categories [4]. The six categories are:

- Space
- Objects, attributes and states
- Actions
- Rules
- Skill
- Chance

Space mechanics refers to the environment wherein the game takes place [4]. The game space is classified by its dimension and whether it is discrete or continuous [4]. The dimension of a game space refers to whether the game is played in a 2D or 3D space [4]. The five types of game spaces given by Shell's taxonomy are linear, grid, graph, points in space and divided space [4]. Meanwhile, discrete spaces represent spaces that have a limited amount of positions, while continuous spaces don't have specific demarcated cells [4]. For example, chess is a game with a discrete space since it has 64 blocks in which a piece may be placed, while a modern racing game has a continuous space since the car can be located anywhere on the racing track.

Object mechanics refer to anything that can be interacted with in the game [4]. An object is described by its attributes while the object's state conveys the values of the attributes. For instance, attributes of an object can be its colour and size while the state of the object is green and small. The state concept is extremely important for this study and is continuously encountered in the reinforcement learning background section (Section 2.3). Action mechanics are performed on the objects of the game (such as running, shooting and jumping) [4].

It is stated in Section 2.2.1 that rules are the fundamental difference between gaming and playing. Shell further states that rules are an important mechanic since "they define the space, the objects, the actions, the consequences of the actions, the constraints of the actions, and the goals" [4]. Rules can be considered as mechanics that glue game mechanics together.

Skill mechanics require a player to use physical, mental or social skills to complete game tasks [4]. Skill mechanics are useful in managing duel flow and implementing hard fun. Contrastingly, chance mechanics are unpredictable and surprise the player [4]. Chance mechanics increase the replay value of games by diversifying the gameplay experiences [4]. A discussion on luck and skill-based games is continued in the following subsection.

Kramarzewski and De Nucci add that a game's mechanics do not have to be novel and that it is encouraged to take inspiration from existing games [4]. In fact, players tend to play games that are familiar to the games that they are already good at [44]. Using existing mechanics of other games also has the advantage of being tried and tested, meaning that the response of players to the mechanic is already known.

Lastly, a stark resemblance between Shell's taxonomy of game mechanics and the Markov Decision Process (MDP) used in reinforcement learning is seen in Section 2.3.1. The environment of an MDP encapsulates the space, rule, object and chance mechanics while the reinforcement learning agent (AI) attempts to learn the skill mechanics. The agent performs actions on the environment's objects and receives states of the objects from the environment.

Therefore, the choice of game mechanics influences the reinforcement learning agent's performance. A well-defined set of mechanics can aid in the development of the reinforcement learning agent.

Luck and skill

Uncertainty in a game's outcome is required by all games, since a completely deterministic game would cause players to just perform an algorithm without adding much thought to it [45]. Koster adds that this mindless repetition of tasks is the reason why players start to find a game boring and stop playing [44]. Uncertainty can be added to a game through the use of random elements [45].

Random elements in a game can stem from overt random elements, von Neumann games and/or the human nature itself [45]. Overt random elements are the obvious sources of outcome uncertainty and include elements such as dice, cards, random number generators or the weather [45]. Von Neumann games refer to the random outcomes of players making choices privately and later revealing them to the other players [45]. Examples of von Neumann games include the fog of war in strategy games, movement of players in first-person shooters or a game of rock-paper-scissors [45]. Lastly, human nature itself is unpredictable and a source of randomness [45].The unpredictability of human nature is why skill-based games, such as chess and go, also contain uncertainty in their outcome [45].

However, players often object against in-game random elements since it reduces the influence of a player's decisions on a game's outcome [4][45]. The reduction of player influence on the game's outcome explains why novice players have a higher chance of beating expert players in a luck based game [45]. Subsequently, Elias *et al.* argues that skill and luck are independent elements of a game and that a game can contain a great amount of luck as well as a high amount of skill [45]. The reason being is that luck only causes a decrease in expert players' win percentage (returns to skill), while skill ensures that expert players consistently defeat novice players in the majority of cases [45]. In fact, the skill required to play a game is not necessarily an intrinsic property of the game, but rather something that the community of players possesses [45]. Therefore, Elias *et al.* states that the skill property of a game is agential, which means that a community of very weak players can find a simple game (such as tic-tac-toe) challenging, while an extraordinary community of players may find a sophisticated game (like chess) trivially easy [45].

Luckily the frustration of randomness in games can be reduced by distinguishing between input and output randomness [4]. Input randomness informs the player's decision and includes elements such as maps that are generated at the start of a game or cards that are drawn from a deck [4]. Output randomness determines the outcome of a decision after the player makes the decision [4]. An example of output randomness is when a player

plays a card and the card has a 50% chance of having a positive outcome. Generally, input randomness is beneficial, and output randomness is best left to be avoided [4]. The randomness of games can be reduced by adding a preparation phase, such as building a deck in a card game, before the main game commences [44].

Lastly, properly implemented luck-based mechanics benefit games by enabling skilled and less skilled players to play together, helping players feel better about a lost game (they can blame bad luck), increasing the variety of gameplay, creating player interest in what the outcome will be, helping behind players catch up, and reducing the amount of calculations that a player has to do [45]. Reducing the amount of calculations a player can do is achieved with a mechanic known as hidden information [45]. Hidden information and how it manages the depth of games is discussed in the next subsection.

Deep games

Deep games refer to games with interconnected complex systems that enable the player to make interesting choices [4]. Deep games are often described as being easy to learn and hard to master [4]. Moreover, deep games have high replay value due to their reliance on intricate mechanics instead of new content (known as the content furnace) [4].

Typical characteristics of deep games include interconnected content that creates millions of combinations, rules and mechanisms that test different types of skills, satisfaction of playing against human opponents, and a dominant strategy that constantly changes [4]. Kramarzewski and De Nucci remark that human opponents are more engaging and popular than AI opponents, due to the difficulty of creating natural AI opponents as well as the effort involved in developing AI opponents that work together and perform interesting choices [4]. Consequently, the necessity of creating convincing AI in deep games further calls for the use of reinforcement learning (Section 2.3).

One might come to the conclusion that deep games are synonymous with complex games since complex games have plenty of game mechanics and deep games require the interconnection of mechanics [4]. However, the art of game design is to have an elegant solution that favours efficiency and simplicity [4]. The preference is to have games with minimal complexity and maximum depth [4]. Tactics that can reduce a game's complexity, without affecting the depth, include improving the user interface, reducing the pace of the game, and providing tutorials on how a game's mechanics work [4]. It is encouraged to design tutorials that actively involve the player instead of making the player a passive observer [4].

Care should be taken to ensure that there are no optimal strategies when developing deep mechanics [4]. Optimal strategies are detrimental to a game's depth since players will only use the mechanics that are part of the optimal strategy and ignore the suboptimal ones [4][44]. Subsequently, the game loses its depth as it loses its mechanics. Unfortunately, it is in the nature of players to search for an optimal strategy that can be exploited to win games, even if it causes the game to become less fun [44]. Ultimately, players will go as far as cheating if they are given the opportunity [44]. Players' decisions should therefore have an element of risk, with positive and negative consequences, when made in different scenarios to ensure that there is no optimal strategy for every game scenario [4][44].

The opposite is also true, where players can have too many choices and ultimately struggle to make a decision (analysis paralysis) [4][44][45]. A solution to this dilemma is to make use of hidden information [45]. Hidden information refers to hiding information from the player to ensure that the player cannot overanalyse the game [45]. Generally, hidden information is implemented using private information, "puzzlelike" information or randomness [45]. Private information is generated by a random process and assumes that the information is not being shown to the player [45]. Examples of private information are a player's cards that are hidden from the opposing player in a card game or the fog of war in a real-time strategy game [4]. "Puzzlelike" information remains hidden from a player until it is discovered [45]. An example of "puzzlelike" information is a crossword puzzle which loses its secret as soon as a player completes it [4]. Finally, randomness can be used to hide information from a player by making sure that the player is unable to calculate the stochastic outcome [4].

Figure 2.2 provides an illustrative summary of the above-mentioned game depth concepts. It can be seen from Figure 2.2 that simplistic games don't have many interconnected mechanics while complex games have a large amount of mechanics. On the other hand, elegant deep games are able to have many interconnected mechanics using only a few mechanics. For instance, notice from Figure 2.2 that both the complex games illustration and the elegant deep games illustration have eight interconnections between mechanics, but the elegant deep games appear much simpler due to having fewer mechanics. An optimal strategy causes many of a game's mechanics to become irrelevant while hidden information segments the depth space so that a player has less to focus on when making a decision.

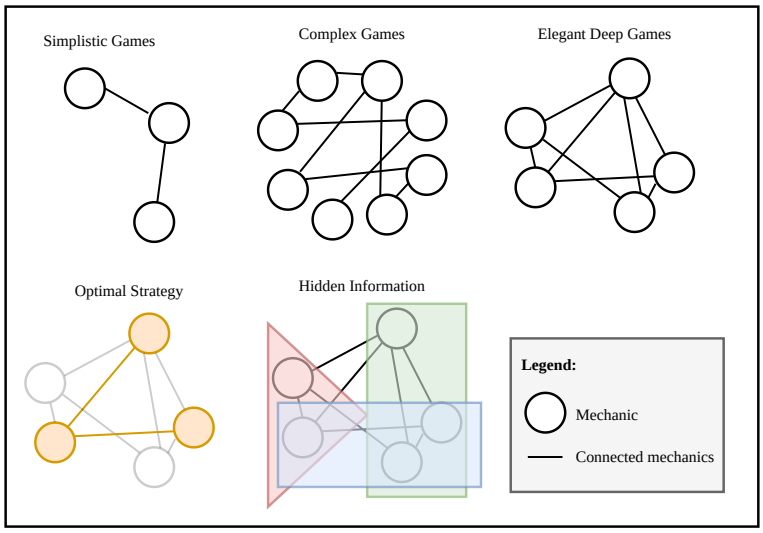


Figure 2.2: Illustration of game depth

Lastly, deep games have increased game development difficulties such as taking more time to design and having increased odds of a game-breaking bug occurring due to the larger possibility space [4]. Fortunately, Chapter 4 shows that a reinforcement learning agent can be used to search the possibility space in order to reduce the chance of a game-breaking bug occurring. The next subsection is the final subsection on game design fundamentals and provides an overview of the negative effects associated with player friction.

Player friction

Friction in games can frustrate the player to the point of quitting the game [4]. Common elements that lead to friction include mandatory puzzles that block player progression, having a user interface/game control design that ignores established conventions, or confusing players with unclear objectives [4].

Interestingly, the introduction of progression blockers and progress removal can reduce friction if it is done appropriately. *The Oregon Trail* is a serious game that does just this by introducing progression blockers that will certainly cause a player to fail and start over when encountering the blocker for the first time [46]. The idea is to remove the stigma of failure and to challenge players on how far they can go before failing in the game [46]. The art of good game design is knowing when to break the game design rules in order to create innovative and interesting games [4].

This marks the end of the final subsection on game design fundamentals. The following subsection probes a few existing mathematical serious games in order to observe the design trends that are followed by them.

2.2.3 Overview of existing mathematical games

Case studies of a few mathematical serious games are presented in this section. These case studies aim to provide a background on available mathematical serious games and serve as inspiration for how a mathematical serious game can be implemented. The mathematical serious games presented in this section are Math is Magic, Clash of Wizardry, and Mathy Fight.

Math is magic

Math is Magic is a story-based role-playing game (RPG), similar to the old Pokémon games. The game measures learning effectiveness, cognitive workload, and engagement of learners [47]. The mathematical skills developed by the game include problem analysis,

communicative property, invariant property, associative property, dissociative property, distributive property, and base ten multiplications [47]. A screenshot of the game can be seen in the figure below.

Figure 2.3: Math is magic gameplay [47]

Mathematical questions are presented as game prompts during duels [47]. Users also have the option to request in-game help via a "spell book" that illustrates how to do example questions [47].

Figure 2.4: Questions in Math is magic (left) and illustration of spell book (right) [47]

The difficulty of the game is adjusted based on the player's skill ratio and not with game levels [47]. The player's skill ratio is defined as the ratio between the number of questions correctly answered and the total number of questions answered [47]. The reason for using the skill ratio is to reduce player boredom (when the game is too easy) as well as player frustration (when the game is too difficult) [47].

Lastly, future improvements proposed by the game designers include an improved (more complex) student model, adding hidden missions and increasing the amount of learning content [47].

Clash of wizardry

Clash of Wizardry aims to aid 12-16-year-old students in solving first-degree linear mathematical equations [48]. The game has its foundations in constructivist theory since it enables learners to actively build their own equations [48]. In-game screenshots are shown in Figure 2.5 below.

Figure 2.5: Clash of wizardry gameplay [48]

Clash of Wizardry is a two-player game, with the rival player being either a human or the computer [48]. Players battle against each other by casting spells [48]. Different spells have distinct effects and are represented by a variety of mathematical equations (such as addition, multiplication, division and so forth) [48]. A spell is performed when the correct multiple-choice answer of a mathematical equation is chosen [48]. The potency of a spell is dependent on the difficulty of the mathematical equation being answered [48]. A player is finally crowned victorious when the opponent's health is fully depleted [48].

Therefore, a player wins when he solves mathematical equations quicker than the opponent. Players can develop strategies that compromise between the difficulty of a question (difficult equations provide more damage) and the speed at which they answer the question (can deal more damage to the opponent when casting spells quicker than the opponent). The game also has public leaderboards to motivate competitive players to play the game [48].

Barbieri *et al.* mentions that school teachers were initially sceptical about using the game to teach mathematics [48]. This scepticism was luckily diminished after the first experiment, since the teachers could see the results of the game and the support that it provides [48]. The support and feedback of the students and teachers were used to improve the game's mechanics, graphics and usability [48].

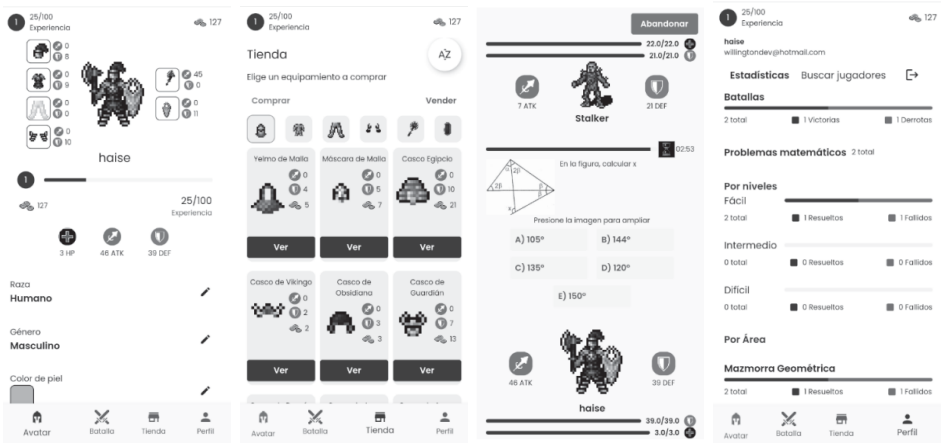
Mathy fight

Mathy Fight aims to teach high school learners geometry by providing them with a multiple choice geometry question as seen in Figure 2.6c [41]. Correctly answering the question causes the player to attack an enemy, with the effectiveness of the attack influenced by the stats of the player's avatar [41].

Consequently, the player's avatar (see Figure 2.6a) is a central element of the game since progression and gameplay are linked to the avatar [41]. Rewards earned through battles can be spent at the store (as seen in Figure 2.6b) to enhance attributes of the player's avatar [41]. However, the choice of items to buy can be perplexing since each item affects a player's attributes, such as damage or defence, differently [41]. The differences in item effects cause the player to carefully consider which item will best suit the needs of their avatar.

Figure 2.6d shows that there are statistical summaries available in the game so that players can evaluate and analyse their performances [41]. Progression is ensured by providing levels that the player should master before proceeding to the next level [41].

Lastly, the developers of Mathy Fight note that adding multiplayer can be a future improvement to the game [41]. Multiplayer will enable competition among players, thus increasing their motivation to play the game [41].



(a) Customization (b) Store (c) Battle (d) Stats

Figure 2.6: Mathy fight gameplay [41]

2.2.4 Game classification

It can prove useful to develop terminology for game classification in order to be more specific when referring to a certain type of game. A formal way of approaching this is by using game theory. Game theory is often employed in politics and economics to provide tools for analysing scenarios where players make independent decisions [49]. Games are frequently classified by the number of players, with the broad categories being one-person, two-person or n-person games [49]. One-person games are usually pitted against nature and do not interest game theorists much [49]. N-person games refer to games that are played by more than two players [49].

Players of the games can either hold perfect information (know everything that happens in the game) or imperfect information (some information is hidden from the player) [49]. An example of a perfect information game is chess (all the pieces are visible to the player) while an imperfect information game is poker (the opponent's cards are hidden) [49].

The alignment of the players' goals in the game is another basis of classifying games [49]. The games can be constant sum (resources that the players compete for remain constant throughout the game) or variable sum (the amount of resources that the players compete for can vary) [49]. Constant sum games are always competitive (players have opposing goals) while variable sum games can be cooperative or non-cooperative [49].

A special type of constant sum game is the zero-sum game. Zero-sum games require that the gain of one player implies the loss of the other player and vice versa [50]. Therefore, the net sum of the game's utilities will always equal zero [50]. The net sum of the game's utilities being zero means in a two-player zero-sum game that the winner obtains a score of +1 and the loser obtains a score of -1, so that the net sum of game scores equates to

zero. Zero-sum games are particularly important to this study and influence the type of reinforcement learning strategy employed by this study (see Section 2.3.3).

2.2.5 Summary of game design background

This section described the core considerations that should be taken into account when designing a serious game. Most notably, it is found in Section 2.2.1 that the core element of a game is its rules. Also, serious games differ from entertainment games by having a characterizing goal. The characterizing goal of this study's serious game is to aid high school learners with algebra.

Fundamental game design principles are discussed in Section 2.2.2, with topics being the flow state, motivation and fun, uniqueness of players, game mechanics, luck and skill, deep games, player, and friction. The game flow subsection states that games should aim to strike a balance between game difficulty and players' skill level to ensure that the players remain immersed in the game. Meanwhile, the motivation and fun subsection states that a game is fun when it follows at least three of Lazzaro's four keys to fun (namely hard fun, easy fun, serious fun and people fun). It is also shown that players have personalized game preferences. Therefore, the "uniqueness of players" introduced the Bartle's player model, which can be used to broadly classify players as the achiever, killer, explorer, and/or socializer player types. Moreover, Shell's taxonomy of game mechanics (space, objects, attributes, states, actions, rules, skill, chance) is discussed in the game mechanics subsection, and a stark resemblance is noticed between these game mechanics and the Markov Decision process of reinforcement learning (Section 2.3.1). Deep games advocate for elegant interconnected mechanics instead of trying to impress players with new content. Finally, the luck and skill section states that luck and skill in games are independent game attributes, and the player friction section warned against the use of progression blockers.

Three mathematical serious game case studies are considered in Section 2.2.3, with the games being *Math is Magic*, *Clash of Wizardry* and *Mathy Fight*. The main similarities between the games include explicitly practising a player's mathematics and their fantasy settings where mathematics is used to duel opponents. These games take a drill-tool approach in teaching students mathematics.

Lastly, it is shown in Section 2.2.4 that games can be classified with the use of game theory. The game theory classifications tend to be especially useful when considering multi-agent reinforcement learning approaches in the following section.

2.3 Reinforcement learning introduction

The Vygotskian theory in section Section 2.1.2 states that students require a mediator to assist them with learning difficulties so that they can eventually become independent learners. These mediators can assist students by merely demonstrating to them how an ideal solution would look like [34]. Also, Kramarzewski and De Nucci mention that it is vitally important to develop prototypes for games, before commencing production, in order to reduce the risk of overcommitting resources on ideas that are doomed to fail [4]. Game prototypes create the need for hiring playtesters to help test different game prototypes in order to determine which prototypes can be successful. Unfortunately, it also means that a large amount of resources need to be invested to hire playtesters.

Luckily, an artificial intelligence (AI) technique known as reinforcement learning (RL) is able to autonomously train a computer in making sequential decisions that are optimal in the long term [7]. The decision-making computer, called an agent, is capable of contributing as a playtester and as a mediator. The tight relationship between RL and the fields of psychology and neuroscience provides an additional benefit of making RL the type of machine learning (ML) paradigm that closest resembles the kind of learning that humans and animals do [6]. The similarities between human learning and RL have the potential side effect of the RL agent being a rough indication of whether the serious game can aid students in learning mathematics, since the RL agent has no prior mathematical knowledge and the students and RL agents undergo the same learning method (reinforcement learning) when playing the game. The lack of prior knowledge means that any mathematical concepts "learned" by the RL agent are purely due to the game.

The principle philosophy behind reinforcement learning is that desirable/correct choices (called actions) must be rewarded, and undesirable choices must be penalized. The use of rewards and penalties is precisely the techniques employed by the response-strengthening teaching pedagogy mentioned in Section 2.1.2. RL used in machine learning algorithms takes this concept further by seeking to perform actions that will yield the maximum expected long-term reward [6]. The maximum expected long-term reward is different from choosing the actions that present the largest immediate reward, since an action with a small immediate reward can potentially lead to a delayed large expected long-term reward [7]. For instance, choosing to ride a bicycle, instead of a car, to work provides negative immediate rewards (takes longer to reach work and is exhausting), but has a higher expected long-term reward (environmentally friendly, healthy exercise and saves money).

The following subsection investigates the core ideas of RL in greater detail with Section 2.3.1 providing the technical foundation of RL, Section 2.3.2 presenting background on the common types of practical RL algorithms, Section 2.3.3 discussing the application of RL in multi-agent scenarios, and Section 2.3.4 finally concluding with a few case studies

of breakthrough RL implementations in games. The design of this study's RL agent is presented in Chapter 4. It should be noted that the field of RL is quite vast and that this background section only provides a glimpse into some of the fundamental RL concepts. Much research is still being conducted in order to improve shortcomings of the RL methods mentioned in this section [7][6].

2.3.1 The RL framework

This subsection discusses the primary framework that is used to represent RL tasks. It will be seen that RL normally consists of a policy, reward signals, value functions, and optionally a model of the environment [6]. The most important aspect of an RL algorithm is to find a method that efficiently estimates the value function [6]. This subsection investigates some theoretical concepts and RL approaches, while the following subsection (Section 2.3.2) provides model-free methods that are more useful for practical applications.

Mathematical model for the decision making processes

The Markov decision process[1] (MDP) is a formal way of mathematically defining sequential decision-making [7][6]. Figure 2.7 illustrates the MDP by presenting an agent and environment interacting with one another by means of actions, states and rewards[2]. These interactions are termed as the *trajectory* of the process [6]. The trajectory keeps track of the action performed by the agent at each time step and the resulting state-reward pair that is then received from the environment $(S_0, A_0, R_1, S_1, A_1, S_2, A, 2, R_3, ...)$[3] [6]. The time step of the trajectory doesn't have to be fixed and the transitions from a state-action pair to a state-reward pair can be stochastic [6]. The conditional probabilities of receiving a specific next state (s') and reward (r) pair, given the current state (s) and action (a) pair, are referred to as the *dynamics* of the MDP. THE MDP's dynamics is mathematically represented as $p(s', r|s, a)$ [6].

Therefore, it is clear that the agent is the decision maker and the environment is everything outside the agent that the agent interacts with [6]. In the case of this study, the AI opponent (that the human player is up against) represents the agent while the serious game and even the human player himself are part of the environment.

[1] Readers from a statistical background might find it interesting to note that the MDP is very similar to Markov chains (MC). The main difference is that MC only account for the transition probabilities between states [7]. MDPs adds decision-making capabilities to MC by incorporating actions and rewards that also influences the transition between states [7].

[2] Readers from a control theory background can find analogies between RL and control theory when thinking of the agent as the controller, the environment as the plant and the action as a control signal [6].

[3] Capital variables represent random variables while lower case variables represent realized variables.

A discussion regarding the opposing player being part of the environment is given in the multi-agent RL background section (Section 2.3.3).

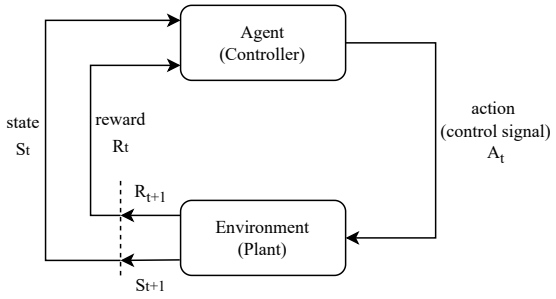


Figure 2.7: Illustration of the Markov decision process [6]

Now, the state is possibly one of the most important aspects to define in the MDP since the dimensionality of the problem is dependent on the state and a state should ideally represent all the information that is required to make a decision without relying on knowing what happened in prior decisions (satisfies the Markov property) [7][6]. Problems in which the entire state of the environment is available to the agent are known as *fully observable* MDP while problems in which some aspects of the state are hidden from the agent are known as *partially observable* MDP [7]. The state of an MDP is represented as a vector of numbers [6].

For example, when representing the state of a tennis ball in a tennis match, one might be tempted to represent the ball only by its position in Cartesian coordinates ($x, y,$ and z) in order to determine when the agent should hit the ball. However, this state representation requires one to know the previous coordinates of the ball at previous time steps in order to infer the speed of the ball. A better representation, that adheres to the Markov property, would be to represent the state of the tennis ball with its position, velocity and acceleration in Cartesian coordinates. The position, velocity, and acceleration representation allows the agent to know the state of the ball without needing to know what happened in previous time steps.

In fact, the concept of state is so important that entire computer programming constructs have been developed to handle states, such as object-oriented programming (OOP), finite-state machines and REST APIs. One can think of the *properties* of an OOP object as representing the state of the object. RL itself is differentiated from other forms of AI that seek to find optimal solutions (such as multi-arm bandits or evolutionary methods) by using the state of the problem to guide decision-making, whereas the other methods do not[6].

Actions are the choices that an agent makes, and are represented by a vector of numbers [6]. Using a vector means that actions can be discrete or continuous, and that an agent can perform multiple actions in the same time step [7]. For instance, a self-driving car agent can choose to steer its car left and accelerate in the same time step by utilizing a vector with two numeric values. Actions influence immediate rewards as well as subsequent/future states and rewards that will be visited by the agent [6].

Rewards are always a single number that is used to convey the goal to the agent [6]. It is important that rewards and penalties only indicate whether desirable or undesirable outcomes are achieved and that the rewards and penalties should not instruct how to meet these outcomes [6]. For example, when the shortest path between two destinations is desired, a good reward system will penalize the length of each path taken by the agent (shortest length will have the least amount of penalties) while a poor reward system will provide rewards for individual turns taken. This individual turn reward system is considered poor since the goal of the agent is to satisfy the individual turn requirements instead of finding the shortest path. The process of designing a good reward system is called *reward engineering* [7]. Desirable reward choices exhibit properties such as being fast to compute, are correlated with the "true" reward, are not very noisy, encode non-functional requirements (such as faster, safer, or more efficient solutions), are large enough to be distinguishable from noise, are frequent, are not delayed too far from the action that it is attributed to, have a smooth reward distribution to prevent jittery (robotic-like) behaviour, and have simple distribution conditions [7].

However, the action immediately preceding a reward is not necessarily the only action that contributed to the reward being given [6]. A series of previous actions likely helped to set up the environment for the particular state that dispensed the large reward [6]. Therefore, it can be beneficial to also take into account the influence of future rewards when rewarding a certain action at time-step t by making use of the *discounted return* (G_t). The discounted return is expressed as:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-t-1} R_T \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{2.1}$$

where γ is the *discount rate* that weights how much of the future rewards should be attributed to the current time step [6]. $\gamma = 0$ states that only the reward of the current time step should be taken into account while $\gamma = 1$ states that all future rewards from the current time step onwards should carry equal weights. The symbol T in Equation 2.1 indicates the time step when an episode ends, known as the *terminal state* [6]. An *episode* is defined as the natural subsequences that can be identified from the agent-environment interactions [6].

For instance, an example of an episode in *tic-tac-toe* is a single game of *tic-tac-toe* and the end of the episode occurs when the game ends with a player winning or both drawing. The case where $T = \infty$ is referred to as a *continuing task*, which means that there is only one indefinitely long episode [6]. Also, the terminal state's expected return is defined to be zero ($G_T = 0$) since the episode has ended in the terminal state and there is no reward further expected to be received [6].

Therefore, the main takeaway from this subsection is that the decision-making process of RL can be mathematically formalized with the MDP. It is important that the MDP representation of the problem should be simple enough to solve and reason about while still being representative of the real problem [7]. Achieving the desired model simplicity is often not an easy task and requires much effort and many trials and errors (becomes an art rather than a science) [6]. However, developing a simple MDP representation of a RL task is well worth the effort since the MDP model often has a much larger effect on the RL performance than the choice of RL algorithm being used [6]. The following subsection describes how the expected reward of different states and actions influences the agent's decision-making process.

Value functions and policies

It is stated in the previous subsection that the reward communicates the goal that the agent wishes to achieve. The agent is tasked to maximize the reward by maximizing the expected discounted return (G_t). However, one would like to know which states and actions have the maximum expected return, since greedily choosing the actions with the maximum expected return will most likely result in obtaining the maximum possible reward [6]. The choice of greedily using the expected return instead of the expected reward is due to the expected return taking into account the long-term reward benefits of a state/action and not just the immediate short-term reward benefits [6].

The expected value of a state is given by:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \quad \forall s \in S \tag{2.2}$$

where $\mathbb{E}$ denotes the expected value operator used in statistics [6]. As a brief recap, the expected value in the discrete case represents the weighted sum of all the values that a random variable (X) can assume and the probability of the random variable taking that value, $p(X = x)$ [51]. The expected value is therefore dependent on the statistical distribution of the data and estimates the approximate value that a random variable is most likely to assume [51]. The mathematical formula for the expected value is defined by :

$$\mathbb{E}[X] = \sum_x p(x) \times x \tag{2.3}$$

where the relationship between the statistical distribution of the data and the value of the data is clearly shown in the calculation of the expected value [51]. Therefore, Equation 2.2 describes how beneficial it is to be in a certain state by calculating the rewards that one can expect from being in that state. The estimated return is a balance between the frequency of encountering a reward and the magnitude of the reward. Equation 2.2 is subsequently known as the *state-value function* [6]. A similar function exists for actions and is called the *action-value function* [6]. The action value function is mathematically given as:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{2.4}$$

which showcases that the action value is dependent on the action as well as the state [6]. The action value is especially useful when deciding which action should be taken in a given state since the action with the largest action-value will most likely provide the largest reward [6]. Coincidentally, the mapping from an environment state to an agent's action is known as the *policy* of the agent [6]. The policy (π) defines the behaviour of the agent. The policy can be deterministic with $a = \pi(s)$, or stochastic with $\pi(a|s)$ (gives the probability of taking action a given state s).

Equation 2.2 and Equation 2.4 show that the value functions contain π as a subscript. The subscript emphasizes that the value functions are dependent on the current policy of the agent. The dependence of the value function on the current policy makes sense since the policy defines the subsequent actions that will be taken by the agent and makes it possible to determine the expected return of the actions/values. However, the value functions can again be used to improve the policy of the agent since states/actions with larger value functions indicate superior choices [6]. The interrelationship between the policy and value functions, together with their influence on improving each other, is further investigated in the next subsection.

Generalized policy iteration

This section is concerned with the calculation of value functions and policies. Specifically, the relationship between states, actions and rewards can be illustrated in a backup diagram, as is shown in Figure 2.8.

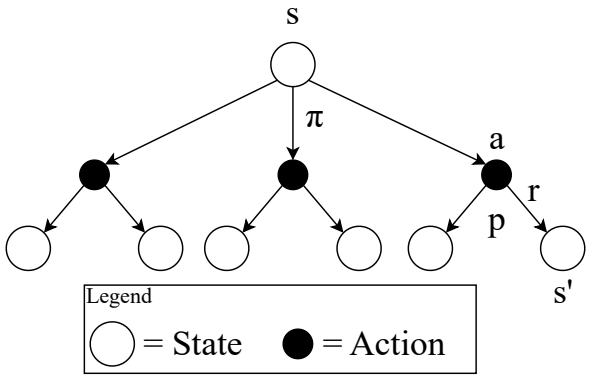


Figure 2.8: Backup diagram for the state-value function [6]

Notice from Figure 2.8 that the current state (s) has a probability (denoted by the policy π) of choosing a certain action (a). The chosen action then has a probability (denoted by the environment's dynamics p) of yielding a certain reward (r) and setting the environment into a future state (s'). Using the equations related to expected values (Equation 2.3), expected discounted return (Equation 2.1) and state-value functions (Equation 2.2) along with the backup diagram of Figure 2.8, one can develop a theoretical expression for determining the state value function as:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \quad \forall s \in S \tag{2.5}$$

,which is known as the *Bellman equation for* v_π [6]. The observant reader might find the Bellman equation for state-value functions a bit strange since calculating the state-value function requires one to already know the state-value of another state [4]. These types of dilemmas can be solved by iteratively updating estimate values of the value functions until they converge [6]. Updating Bellman's equation for state-value functions by following a policy that greedily chooses the maximum state-value yields the *Dynamic programming* algorithm. Dynamic programming is our first RL algorithm and simply updates the value function according to:

$$v_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \quad \forall s \in S \tag{2.6}$$

so that the state value function is iteratively calculated [6]. A side effect of Equation 2.6 is that an improvement in estimated state-values accuracies after each iteration also causes the policy to improve due to it having more accurate state-values to choose from [6]. The

[4]The usage of one estimate to calculate another estimate is referred to as *bootstrapping* [6]

improved policy is able to select more optimal states, which again improves the estimation accuracy of the optimal states [6]. This improvement process is known as *generalized policy iteration* (see Figure 2.9) and is the principle that almost all RL methods follow [6].

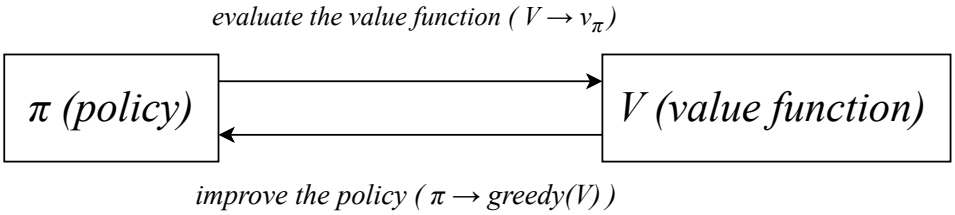


Figure 2.9: Illustration of generalized policy iteration [6]

The generalized policy iteration process is repeated until the value function is consistent with the current policy and the current policy is greedy with respect to the value function[5] [6]. Otherwise stated, the process is stopped when the change in policy and value estimates becomes sufficiently small between iterations .

Similarly, a backup diagram for actions is shown in Figure 2.10 [6].

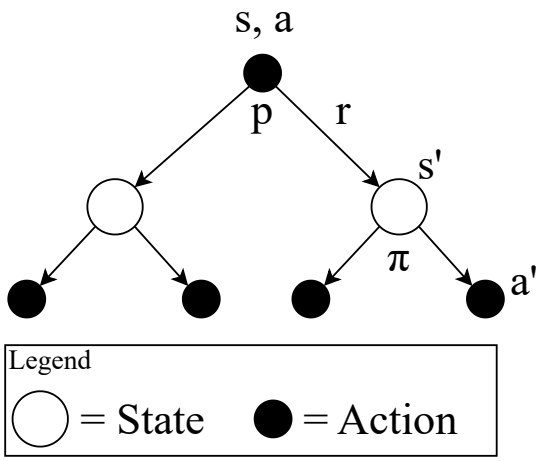


Figure 2.10: Backup diagram for the action-value function [6]

[5]The estimation of a value function is referred to as the prediction problem while the determination of the policy is referred to as the control problem [6].

Using Figure 2.10 and the methods for obtaining Equation 2.5, one can derive the *Bellman equation for* q_π [6]. The Bellman equation for q_π is exhibited by:

$$q_\pi(s,a) = \sum_{s',r} p(r,s'|s,a)(r + \gamma \sum_{a'} \pi(a',s') q_\pi(s',a')). \tag{2.7}$$

The combination of Equation 2.5 and Equation 2.7 enables one to express the state-value function in terms of the action value function and the action value function in terms of the state value function [6]. This interrelationship between the state and action value functions is shown by [6]:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) \tag{2.8}$$

$$q_\pi(s,a) = \sum_{r,s'} p(r,s'|s,a)(r + \gamma v_\pi(s')). \tag{2.9}$$

Unfortunately, there are a few assumptions that impede the applicability of dynamic programming in practice, namely having access to an accurate model of the environment ($p(r,s'|s,a)$), having sufficient computational resources (computation speed and memory), and the need for the states to obey the Markov property [6]. These assumptions must often be violated in order to produce implementations that are practically realizable (see Section 2.3.2).

The high computational requirements of dynamic programming mentioned in the previous paragraph might come at somewhat of a surprise. The high computational requirement is largely due to the summation that is done over all states (in addition to the reward and action combinations with the state) when calculating the value functions of Equation 2.5, Equation 2.7 and Equation 2.9. As mentioned in the previous sections, the state is represented by a vector of numbers and each element of the vector describes a dimension/feature of the state. The problem is that adding only one extra dimension/feature to the state vector unlocks a plethora of new state possibilities i.e. a larger summation to perform in the dynamic programming algorithm. The dramatic increase in state possibilities when barely adding more features has been coined by R. Bellman as the *curse of dimensionality* [6].

Figure 2.11 utilises an example of computer pixels to get an intuitive feel for the curse of dimensionality. A greyscale pixel can be represented with a single dimension/scalar value that depicts the intensity of the pixel. Adding only two additional dimensions enables one to position pixels in a 2D grid to form images. It is now already clear that one can represent much more information/states with an image compared to simple monochromatic colours. Taking it one step further and adding another dimension that represents frame numbers/time enables images to be ordered into videos. Not only can videos represent more information than images, but videos can also have the same image at different frames. Therefore, the same lower dimension state can appear as different states in higher

dimensions, causing an even greater variety of states. To finalize the example, another dimension can be added to indicate the movie/game from where the video came. The combination of the above-mentioned dimensions can subsequently describe every pixel of every greyscale video in existence with just this five number summary/dimension! Clearly, this shows that adding more dimensions can dramatically increase the state space and that one should carefully consider if another dimension is worth the additional computational burden.

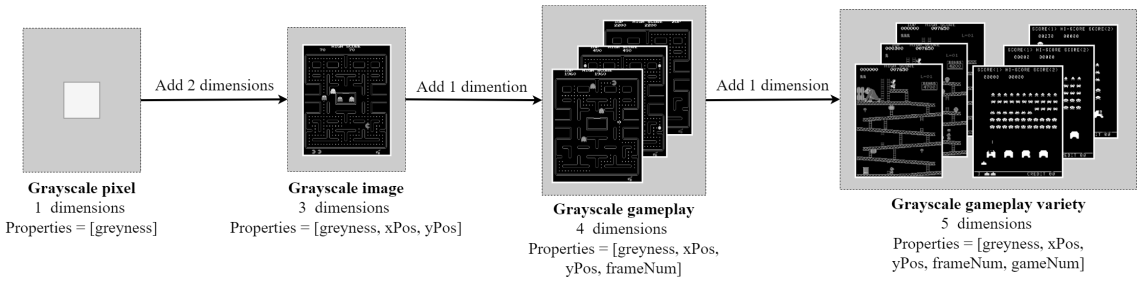


Figure 2.11: Curse of dimensionality illustration

Luckily, another RL algorithm, known as *Monte Carlo*[6], exists where the state space size doesn't affect the computation time of the algorithm [6]. The derivation of the Monte Carlo method does not consider the state-value backup diagram (Figure 2.8) or the action-value backup diagram (Figure 2.10). Instead, the original goal of calculating the expected return (G_t) is rephrased. Starting with Equation 2.1, the expected return is calculated as:

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-t-1} R_T \\ G_t &\doteq R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ... + \gamma^{T-t-2} R_T) \\ G_t &\doteq R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.10}$$

where Equation 2.10 establishes that the expected return can be calculated by knowing the reward and the expected return that follows in the trajectory. Equation 2.10 is non-causal since future returns need to be known in order to calculate past returns. An ingenious solution is to rather let a training episode fully complete and then work from the end of the episode to the start of the episode to calculate the returns [6]. The state-values ($v_\pi(s)$) and action-values ($q_\pi(s)$) are determined by taking the average of all returns (calculated with Equation 2.10) encountered for a specific state or state-action pair [6]. The backup diagram for the Monte Carlo method is shown in Figure 2.12, where it can be seen that updates are only dependent on the states encountered in an episode instead of the entire state space.

[6]The term "Monte Carlo" is often associated with estimation methods that rely on randomization [6]. However, in RL the term is used to describe methods that require the entire episode to be completed before updating the expected returns [6].

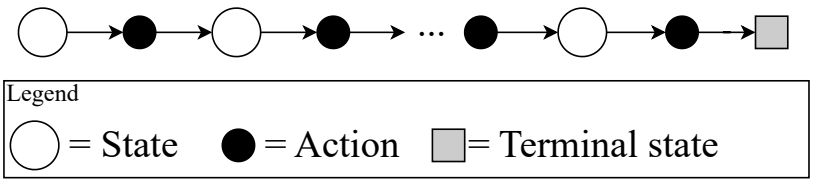


Figure 2.12: Backup diagram for Monte Carlo approach [6]

Monte Carlo has the advantages of not requiring the model/dynamics of the environment and being less affected by large state spaces slowing down training [6]. Unfortunately, disadvantages of the Monte Carlo method are that value estimates are only updated at the end of an entire training episode, meaning that Monte Carlo can only be applied to tasks that can be broken up into episodes, and Monte Carlo doesn't bootstrap (sample efficiency is reduced) [6].

Given the shortcomings of the more theoretical RL methods (dynamic programming and Monte Carlo), additional RL methods have been developed to approximate some of the theoretical RL aspects in order to gain practical implementability. These methods are showcased in Section 2.3.2. The subsequent sections in this subsection further discuss some of the finer details of RL algorithms, such as what it means to have an optimal policy, balancing exploration/exploitation, the difference between model-based and model-free approaches, and the meaning of on-policy and off-policy algorithms.

Optimal policy

The goal of RL is to find the optimal policy for an agent to follow [6]. This section provides a bit more detail on what an optimal policy entails. For one, an optimal policy has an expected return greater than or equal to the expected return of any other policy for all possible states [6]. The relationship between the expected return of an optimal policy in comparison with other policies can mathematically be presented as:

$$\pi^* \geq \pi' \iff v_{\pi^*}(s) \geq v_{\pi'}(s) \quad \forall s \in S \tag{2.11}$$

where π^* in Equation 2.11 denotes the optimal policy [6]. There is always at least one optimal policy in an RL problem [6]. Also, the equal part in the greater-than-equal sign of Equation 2.11 hints that there can be more than one optimal policy for a RL task (the optimal policy might not be unique) [6]. The non-unique property of an optimal policy can be seen in practice when different RL algorithms can have different play styles [7][6]. Section 5.4 of this dissertation further substantiates this point by showing that the agents designed in this study approach the serious game with different play styles.

Optimal policies yield optimal state-value and action-value functions that can mathematically be written as [6]:

$$v_*(s) \doteq \max_\pi v_\pi(s) \quad \forall s \in S \tag{2.12}$$

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a) \quad \forall s \in S, a \in A(s). \tag{2.13}$$

Similarly, the relationship of Equation 2.8 and Equation 2.9 with optimal policies is given by [6]:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \tag{2.14}$$

$$v_*(s) = \max_{a \in A(s)} q_{\pi^*}(s,a). \tag{2.15}$$

It should be noted that the optimal policy can be deterministic (always choosing the action with the highest expected return) or stochastic (the probability of choosing an action is influenced by its expected return) [6]. The appropriateness of a stochastic or deterministic optimal policy is problem dependent [6]. Lastly, all components of the MDP influence the optimal policy, such as state representation, available actions and reward distribution [7]. Changing even one of these components can change the policy that is deemed optimal.

Exploration and exploitation

A fundamental hurdle in RL is balancing exploitation with exploration [6]. So far, previous subsections presented the optimal action as merely choosing the action that has the highest action-value. The strategy of always choosing the maximum is known as *exploitation* since the best known action-value is continuously exploited for maximum rewards [6].

However, the dynamics of the environment can be stochastic, which means that taking the same action in the same state can lead to different future states or actions that are distributed according to the probabilities of the environment's dynamics [6]. Stochastic environment dynamics mean that the RL agent can overestimate an action's value if it insufficiently sampled rewards that are above average or the agent may underestimate the value when the insufficient samples contain rewards that are below average [6]. Therefore, care must be taken to *explore* states that at first seem sub-optimal only because they were not sufficiently sampled [6].

There should of course be a balance between exploration and exploitation since exploring too much causes the agent to make unnecessary suboptimal choices while exploring too little will cause the agent to be oblivious to the actual optimal solution [6]. A trivial method for alleviating the exploration/exploitation dilemma is to use ϵ-*greedy* action selection [6]. ϵ-greedy selection entails choosing actions to be random (exploring) $(\epsilon \times 100)\%$ of the time while exploiting actions $((1-\epsilon) \times 100)\%$ of the time [6]. The ϵ value can be

annealed to zero over time as the policies and value function improvement stabilizes and the exploitation strategy becomes more certain [7].

Other methods further exist to balance exploration and exploitation. For instance, *exploring starts* can be used in conjunction with the Monte Carlo algorithm to encourage explorations by starting an episode each time in a random starting state [7]. Another method is to use off-policy methods, which are further discussed in the "On and off-policy algorithms" section below.

Model based and model-free algorithms

A major distinction between RL algorithms is that some require a model of the environment while others are model-free. It has already been seen that dynamic programming is a model-based method while Monte Carlo is model-free since it doesn't require a model of the environment [6]. Model-based methods rely on *planning* from simulated experiences to develop a policy while model-free methods *learn* the policy from environment interactions [6].

This study concentrates more on model-free methods, since they can in theory be applied to any problems by learning strategies through environmental interactions [7]. A variety of model-free approaches can be found in Section 2.3.2.

On and off policy algorithms

An important distinction to make between RL algorithms is whether the algorithm uses *on-policy* learning or *off-policy* learning. On-policy algorithms have just one policy that generates experiences and is improved by the results of the experiences [6]. On the other hand, off-policy algorithms have two separate policies, one for generating experiences and another policy that is being improved by the generated experiences [6]. The distinction between on-policy and off-policy algorithms might seem insignificant at first glance, but off-policy allows one to make a separation between exploration strategies and learning a policy [7][6]. A plethora of additional training possibilities arise from off-policy learning, such as training from offline logs, learning optimal policies while performing exploring policies, learning from demonstration (imitation learning), and learning multiple tasks in parallel from a single environment [7]. Using multiple agents in the training process can reduce the wall-clock time spent on training [7].

On-policy algorithms are often soft (stochastic) policies to aid the policy in exploration ($\pi(a|s) > 0 \quad \forall s \in S; a \in A(s)$) [6]. The previously mentioned ϵ-greedy approach is one way of making an on-policy algorithm soft for the purpose of exploration. On-policy algorithms have the main advantages of being able to learn faster (exploit earlier) and being

more stable to train than off-policy algorithms [7]. Disadvantages of on-policy algorithms include poor sample efficiency and having an over-optimistic expected return feedback loop [7][52]. Off-policy methods are subsequently popular among state-of-the-art algorithms since they aim to improve the sample efficiency of RL algorithms by incorporating improved exploration mechanisms [7].

Off-policy algorithms consist of a *target policy* (π) and a *behaviour policy* (b) [7][6]. The target policy is the optimal policy that the RL agent attempts to learn, while the behaviour policy is used to generate the experiences that are used to train the target policy [6]. Therefore, the behaviour policy incorporates the exploration mechanism of the training process [7]. Off-policy algorithms are known to contain more variance in their updates, which causes them to converge slower than on-policy algorithms [6].

It is important that the behaviour policy possess *coverage* over the target policy by requiring that every action taken by the target policy is also taken by the behaviour policy [6]. The coverage property therefore stipulates that the target policy can be deterministic, but the behaviour policy must be stochastic in states where the behaviour policy is not identical to the target policy [6]. The coverage property of the behaviour policy can mathematically be expressed as [6]:

$$\pi(a|s) > 0 \implies b(a|s) > 0 \ .$$

On-policy methods can be considered a special case of off-policy methods, where the behaviour and target policy is the same policy [6]. However, a method known as *importance sampling* can also be used to translate the expected return of the behaviour policy into the expected return of the target policy [6]. Almost all off-policy methods use importance sampling to link the behaviour policy with the target policy [6]. Importance sampling converts the behaviour policy to the target policy by simply multiplying the returns with the *importance sampling ratio* ($\rho_{t:T-1}$) using:

$$v_\pi(s) = \mathbb{E}[\rho_{t:T-1} G_t | S_t = s] \tag{2.16}$$

,with the returns (G_t) being collected by the behaviour policy [6]. The importance sampling ratio is mathematically given as [6]:

$$\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \ . \tag{2.17}$$

Notice from Equation 2.17 that the importance sampling ratio will give a greater weight to returns that have highly probable target probabilities, while returns with unlikely target probabilities will be attenuated.

Equation 2.16 and Equation 2.17 therefore reveal that it is possible to estimate a quantity without directly sampling it [7]! It is further advised that the behaviour policy must be

more stable (often meaning smaller updates) than the target policy to ensure that learning remains stable [7].

Implementing RL concepts can also vary depending on how the algorithms are interpreted [7]. Importance sampling is a prime example thereof since importance sampling can be implemented as *ordinary importance sampling* or *weighted importance sampling* [6]. The different practical implementations of theoretical algorithms make it difficult to verify the results of research papers if the implemented RL algorithms are not provided [7]. It is therefore advised to use standard RL frameworks when testing RL algorithms to make sure that the RL implementation is consistent with those that other RL researchers use [7].

Lastly, a quick comment can be made about offline data and its relationship to on-policy and off-policy algorithms. Online data refers to real-time experiences that the agent can have influence over, while offline data is old logs that were previously saved and not re-used to train agents [7]. Offline data causes the agent's actions to have no effect on the next data sample and the agent is unable to make suboptimal choices to verify that the choices are indeed suboptimal [7]. The inability of offline agents to test suboptimal actions causes offline methods to usually exhibit inferior performance when compared to online data [7]. Offline data also demands using off-policy algorithms, since the agent's policy has no influence over the data that is fed into the training process.

2.3.2 Model-free RL approaches

This section showcases some model-free approaches that can be employed in practice. The model-free approaches discussed include temporal difference methods (SARSA and Q-learning), value approximation methods (deep Q-learning), and policy gradient methods (Reinforce algorithm). These practical algorithms generally employ approximation strategies to the theoretical models (such as the Bellman equation) to mitigate theoretical constraints that hinder practical performance.

Figure 2.13 shows a high-level overview of the practical algorithms that are given in this section and the improvements that they provide. The figure shows that temporal difference methods combine the advantages of dynamic programming and Monte Carlo to develop algorithms that can update immediately and do not require a model. Function approximations can be used on the temporal difference algorithms to apply them on continuous state space problems. Finally, policy gradients aim to approximate the policy function instead of the value function so that it can be applied on continuous state spaces and continuous action spaces. The RL algorithms of Chapter 4 are based on policy gradient methods.

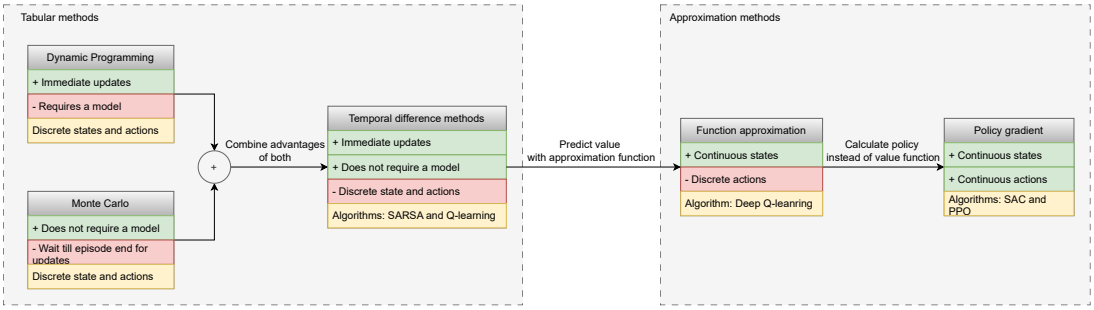


Figure 2.13: Overview of RL methods described in this study

Temporal-difference methods

Temporal-difference (TD) methods are part of tabular-based methods [6]. Tabular methods are known to have discrete action and state spaces, meaning that one can construct a table where the states are represented by the rows of the table and the actions are represented by the columns of the table [6]. This table representation makes it possible to find the action-values of a state by checking the corresponding column of a given row. Similarly, the previously encountered dynamic programming and Monte Carlo algorithms are also classified as tabular methods [6]. The main advantage of using a TD method is that it bridges the gap between dynamic programming and Monte Carlo algorithms by being able to bootstrap and learn without having a model of the environment [6].

Figure 2.14 shows an example of using a 3D bar chart to visualize tabular methods. The agent in Figure 2.14 has a one-dimensional state space and a one-dimensional action space. The value of the bars represents the action-value for each state-action pair. Notice that the state and action spaces are discrete since there are no action-values between the integer states. It can further be seen in Figure 2.14 that state 2 has higher action-values when compared to state -2, which means that state 2 is a better state to be in (higher expected return) compared to state -2. Also, Figure 2.14 shows that action 2 is consistently the action with the highest action-value for all states shown.

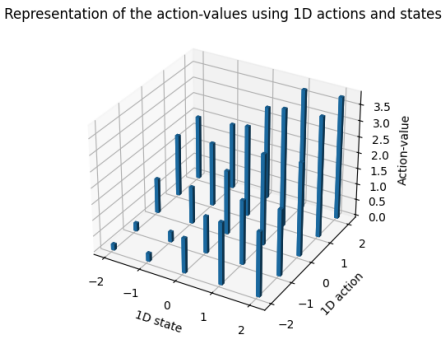


Figure 2.14: Example action-value representation for tabular methods

As with the Monte Carlo algorithm derivation, some ingenuity is applied to derive the update rule of TD-methods. Starting with Equation 2.2, one can write the state-value as $v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s]$. However, the central limit theorem allows the assumption of normally distributed returns when a large sample size is used [51]. The normally distributed return assumption signifies that the expected return of a given state can be approximated with the arithmetic mean of all the returns that were obtained in that state.

Unfortunately, directly calculating the arithmetic mean requires storing every return that has been sampled as well as involving all the sampled returns of a state in the calculation of the mean [6]. This direct calculation becomes computationally inefficient if a new mean needs to be calculated every time a single new sample is added. Instead, one can use an online update rule to calculate the new mean using only the old mean together with the new sample. The online update rule is derived as:

$$\begin{aligned}
V(s)_n &= \frac{1}{n}\sum_{i=1}^{n} G(s)_i \\
&= \frac{1}{n}\left(G(s)_n + \sum_{i=1}^{n-1} G(s)_i\right) \\
&= \frac{1}{n}\left(G(s)_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} G(s)_i\right) \\
&= \frac{1}{n}\left(G(s)_n + (n-1)V(s)_{n-1}\right) \\
&= \frac{1}{n}\left(G(s)_n + n \times V(s)_{n-1} - V(s)_{n-1}\right) \\
V(s)_n &= V(s)_{n-1} + \frac{1}{n}\left(G(s)_n - V(s)_{n-1}\right) \qquad (2.18)
\end{aligned}$$

where n is the number of samples used to calculate the average [7][6]. Equation 2.18 is also known as an exponentially weighted moving average algorithm [7]. The form of Equation 2.18 is common throughout RL, with the main aspects of the form being highlighted as [6]:

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}[\underbrace{\text{Target} - \text{OldEstimate}}_{\text{error}}] \,. \qquad (2.19)$$

Comparing Equation 2.19 with Equation 2.18 reveals that the step size is equal to $\frac{1}{n}$. However, the step size is a hyperparameter that is represented by the α symbol [6]. The step size being a hyperparameter means that the optimal value for the step size is task dependent and that different step size values should be experimented with in order to obtain the optimal value. Large step size values indicate that the error adjustment must have a large effect on the new estimate, while a small step size indicates that only minor adjustments should be made to the new estimate. Large step sizes can speed up the training time, while small or decreasing step sizes are advised in noisy environments to

mitigate the noise from creating noisy/divergent new estimates [7][6]. Non-zero step sizes can aid with non-stationary environments since recent update errors are able to adjust their new estimates to the latest environment parameters [6].

The tricks of Monte Carlo's derivation (Equation 2.10) can be used to bootstrap the expected return of Equation 2.18 as $G_t = R_{t+1} + \gamma V(S_{t+1})$ [6]. The bootstrapped expected return reveals the *one-step TD update rule*, TD(0), as [6]:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \ . \tag{2.20}$$

The error in Equation 2.20 is known as the *TD error* and is mathematically defined as [6]:

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \ . \tag{2.21}$$

The TD error provides an indication of how much the estimated/target value and the old estimate differ from each other, i.e. how large of an adjustment should be made so that the old estimate is equal to the estimated/target value [6].

The TD(0) update rule of Equation 2.20 can be used for control by predicting the action-value instead of the state-value. Directly substituting the state-value function of Equation 2.20 with the action-value function yields the *SARSA* algorithm as [6]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \ . \tag{2.22}$$

The name of the SARSA algorithm stems from the fact that the algorithm makes use of the current state (S_t), current action (A_t), next reward (R_{t+1}), next state (S_{t+1}), and next action (A_{t+1}) [6]. Subsequently, the SARSA algorithm requires a one-step delay so that the next reward, state, and action are available to the update algorithm [6]. The SARSA algorithm is considered to be an on-policy algorithm since the next action is assumed to follow the policy that is being improved upon [6].

A slight modification to the SARSA algorithm yields an off-policy algorithm known as *Q-learning*. Q-learning learns the action-value function (Q) by always updating the policy with the maximum available next state action-value instead of assuming that the behaviour policy will choose the maximum next state action-value [6]. Off-policy algorithms that always update with the maximum available next state action-value are known as using *tree-backup algorithms* [6]. The Q-learning update rule is given as [6]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \ . \tag{2.23}$$

Figure 2.15 presents the backup diagrams for SARSA and Q-learning [6]. The arc in Q-learning's backup diagram states that the maximum action-value should be used to estimate the action-value of the current state-action pair [6]. Interestingly, the behaviour/characteristics of a SARSA agent can be somewhat different from that of a Q-learning agent [6]. One

example is the gridworld environment where the SARSA agent attempts to make safe and consistent choices while the Q-learning agent prefers risky choices that yield large rewards [6].

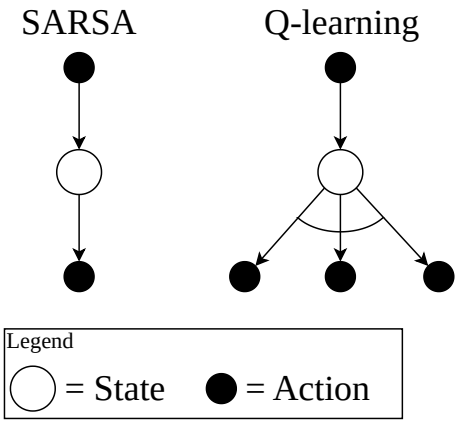


Figure 2.15: Backup diagrams of SARSA and Q-learning [6]

There are of course some improvements that can be made to these vanilla algorithms by incorporating methods such as ϵ-greedy exploration, using double Q-learning to mitigate bad updates, implementing delayed Q-learning to reduce the effect of noise, or using opposition learning to see if the opposite action is indeed suboptimal [7][6].

An elegant method of potentially improving an TD algorithm's performance is to allow the agent to look further into the future by using *n-step* methods [6]. Equation 2.10 is an example of a one-step return, since the expected return is unrolled only once [6]. However, nothing prevents us from unrolling the expected return for *n-steps*, producing [6]:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, \quad 0 \leq t < T - n \ . \tag{2.24}$$

The optimal value for the number of unrolling steps (n) in Equation 2.24 is problem dependent and is treated as a hyperparameter [6]. The resulting n-step SARSA update algorithm is subsequently given by [6]:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right], \quad 0 \leq t < T \ . \tag{2.25}$$

The n-step updates can even be extended to off-policy methods using importance sampling so that [6]:

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h, T-1)} \frac{\pi(A_k | S_k)}{b(A_k | S_k)} \tag{2.26}$$

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n} [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]; 0 \leq t < T \ . \tag{2.27}$$

Equation 2.26 show the importance sampling ratio for n-step methods while Equation 2.27 shows how n-step importance sampling is implemented by slightly modifying the SARSA algorithm.

Unfortunately, n-step methods have disadvantages such as delaying the update rule by n steps and requiring extra compute and memory compared to the base algorithms [6]. An alternative is taking a backwards view instead of the forward view employed by n-step methods [6]. The backwards view is known as *eligibility traces* and requires only a one-step delay instead of an n-step delay [6]. Eligibility traces use a *trace vector* ($\boldsymbol{z} \in \mathbb{R}^d$)[7] to mark a state that has been visited and decays the trace as time passes [6]. The trace vector is able to show how recently a state has been visited and how much influence that state had to obtain a reward (more recent states have larger influence) [6].

An intuitive example of eligibility traces is the usage of a thermographic camera that captures a uniform cold surface. A person can then drag his finger over the cold surface to change the temperature of the surface. The camera will show a trail where the person dragged his finger, with the trail closest to the person's finger being the brightest/warmest and the trail furthest from the person's finger being the darkest/coldest. The thermal trail symbolizes an eligibility trace that shows the recent states visited by the person's finger.

Algorithm 1 shows an eligibility trace variant of the SARSA RL algorithm known as *SARSA(λ)* [7]. The trace decay rate (λ) is the eligibility trace equivalent of the step count (n) in n-step methods [6]. $\lambda = 1$ is equivalent to using all steps of an episode (Monte Carlo) while $\lambda = 0$ is equivalent to only using one-step (hence the name TD(0)) [6]. Line 9 of Algorithm 1 indicates that traces are being increased when they are visited, while line 12 shows that traces are being decreased as time elapses. Subsequently, line 11 shows that the trace vector (indicating the recency of a state-action pair) weights the importance/relevance of the TD-error to a state-action pair.

Although eligibility traces have certain advantages over n-step methods, such as requiring less memory and not having as big computational delay, it still has a computational disadvantage (need to update traces of each state-action pair over multiple steps) [7]. The computational requirements can be reduced by using methods such as only updating states with non-zero trace vectors (by keeping track of them) and using single instruction multiple data (SIMD) implementations [6]. However, problems that have an abundance of data or data that can be generated cheaply, may find it faster to train agents by processing as much data as possible instead of using eligibility traces/n-step methods for more accurate estimates that would require fewer iterations [6].

This marks the end of the first model-free subsection covering TD methods. This subsection introduced the SARSA and Q-learning algorithms as well as the n-step and eligibility traces approaches. The next subsection expands on table-based methods by fitting functions on the discrete value functions of table-based methods in order to handle continuous state and discrete action spaces.

[7] Vector variables are written in a bold font

Algorithm 1 Eligibility traces in SARSA [7]

Input: step size ($0 < \alpha < 1$) AND trace decay rate ($0 \leq \lambda \leq 1$)
Output: SARSA(λ) action-value function

$Q(s, a) \leftarrow 0, \quad \forall s \in S, a \in A(s)$
loop for each episode:
    $\boldsymbol{z}(s, a) \leftarrow 0, \quad \forall s \in S, a \in A(s)$
    choose a from s using π, break ties randomly
    repeat
        Take action, a, and observe r, s'
        choose a' from s' using π, break ties randomly
        $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
        $\boldsymbol{z}(s, a) \leftarrow \boldsymbol{z}(s, a) + 1$
        for (each s ∈ S) AND (each a ∈ A(s)) **do**
            $Q(s, a) \leftarrow Q(s, a) + \alpha\delta\boldsymbol{z}(s, a)$
            $\boldsymbol{z}(s, a) \leftarrow \gamma\lambda\boldsymbol{z}(s, a)$
        end for
        $s \leftarrow s'$; $a \leftarrow a'$
    until s is not terminal
end loop
return Action-value function (Q)

Function approximation solution methods

Model-free function approximation methods are presented in this section. Function approximation methods also rely on value functions and aim to generalize experiences by utilizing supervised machine learning techniques [6]. Figure 2.16 shows that the general approach is to collect samples from the value function (indicated by the blue columns just as with tabular methods in Figure 2.14) and then fit a function approximation (the surface) to the value samples so that unsampled values can be interpolated from the function approximation.

Function approximation solutions are applicable to problems with continuous state spaces and discrete action spaces [7][6]. The reason for the discrete action space restriction is due to the computation cost of finding the maximum action-value (will be expanded upon a bit later in this subsection).

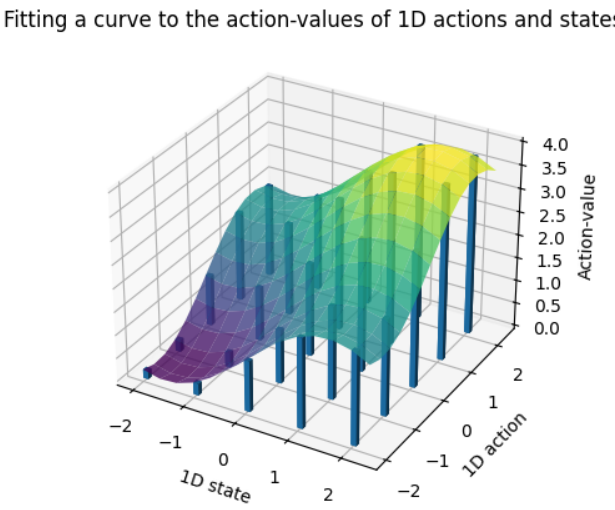


Figure 2.16: Example action-value representation with approximation methods

The primary objective of function approximation methods is to find a set of weights ($\boldsymbol{w} \doteq (w_1, w_2, ..., w_d)^\top; \boldsymbol{w} \in \mathbb{R}^d$) that can be used to approximate a value function ($\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$ or $\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$) [6]. The type of approximators used can be anything from linear functions to multi-layer artificial neural networks or decision trees [6]. A key limitation of these approximation methods is that the dimensionality of the weights is typically less than the number of states [6]. Having fewer weights than states/features means that changing a single weight to improve the approximation of one state might negatively influence the approximation of other states [6]. Luckily, this limitation has benefits as well, such as making the approximation more generalizable to other states and improving an agent's performance on partially observable problems [6].

Therefore, table based methods (of the previous subsection) can be used to obtain the state-value or action-value samples, but a method is needed to determine how well an approximation function fits to these value samples. The need for an approximation function suitability metric introduces the notion of a *loss function* ($L(w)$) that takes the weights of the approximation function as input and produces the loss value for these weights as output [53]. The goal is then to find the weights that produce the smallest loss value possible for a given set of hyperparameters [53].

One of the most common types of loss functions is the mean-squared error (MSE) loss function [7][6]. The MSE is calculated by squaring the difference between the approximated and actual value of each sample and then determining the mean of these squared differences [53]. However, some states in RL are visited more often than others, meaning that it can theoretically be beneficial to make the function approximation of frequently visited states more accurate at the expense of having less accurate predictions at infrequent states [6].

Thus, Sutton & Barto express that the best-known theoretical loss function for RL (even though it aims to find the best value function fit instead of optimal policy) is the *mean square value error* ($\overline{VE}(\boldsymbol{w})$) [6]. The mean square value error is expressed as:

$$\overline{VE}(\boldsymbol{w}) \doteq \sum_{s \in s} \mu(s)[v_\pi(s) - \hat{v}(s, \boldsymbol{w})]^2 \tag{2.28}$$

where $\mu(s)$ is the *state distribution* that indicates the fraction of time spent in each state ($\mu(s) \geq 0, \sum_s \mu(s) = 1$) [6]. Alternatively, the Huber loss function can also be used for RL applications [7]. Figure 2.17 illustrates the typical characteristic of an example loss function with a single weight parameter.

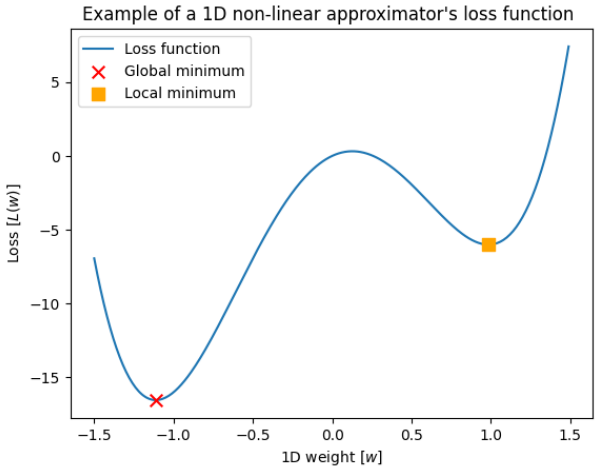


Figure 2.17: Typical characteristics of a loss function

The most popular method for finding the weights that minimizes the loss function is known as *gradient descent* [53]. The optimal set of weights is determined by iteratively testing a set of weights, measuring the loss value of using those weights and determining the adjustments that should be made to the weights in order to decrease the loss even further. The gradient of the loss function provides the magnitude and direction by which the weights should be adjusted [6].

For example, it can be seen from Figure 2.17 that the gradient of the loss function at $w = -0.5$ is large and positive, which requires a large negative adjustment to the weight so that the minimum loss is achieved at $w = -1.1$, while the gradient of the loss at $w = -1.3$ is small and negative (requires a small and positive weight adjustment to make $w = -1.1$). Subsequently, the weight update rule to minimize the MSE loss of a state-value function (assuming equal state distributions) with gradient descent is given as:

$$\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t)]^2 \tag{2.29}$$

$$= \boldsymbol{w}_t + \alpha[v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t)]\nabla\hat{v}(S_t, \boldsymbol{w}_t) \tag{2.30}$$

where ∇ is the gradient operator for scalar functions with multivariable inputs ($\nabla f(\boldsymbol{w}) \doteq \left(\frac{\partial f(\boldsymbol{w})}{\partial w_1}, \frac{\partial f(\boldsymbol{w})}{\partial w_2}, ..., \frac{\partial f(\boldsymbol{w})}{\partial w_d}\right)^\top$) [6]. The half multiplied with the step size (α) in Equation 2.29 is used to make the derivation of Equation 2.30 cleaner [53]. Equation 2.30 is obtained by performing the chain rule of differentiation on Equation 2.29 [6]. α is the same step size that was described in Equation 2.19 of the TD methods subsection [6].

In fact, it can be seen that the general form of Equation 2.30 is the same as that of Equation 2.19, where the magnitude of the error is given by $[v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{w}_t)]$ and the direction of the error is given by $\nabla\hat{v}(S_t, \boldsymbol{w}_t)$ [6]. It can further be seen in Equation 2.30 that the approximation function ($\hat{v}(S_t, \boldsymbol{w}_t)$) must be differentiable with respect to $\boldsymbol{w}$ in order to use the gradient descent weight update rule [6].

Directly applying Equation 2.30 by updating the weights every time a sample is obtained is known as *stochastic gradient descent* since initial weight changes between updates can be quite large/stochastic [53]. An alternative is to use *mini-batch stochastic gradient descent* that does not immediately adjust the weights as samples are processed, but rather collects the intended adjustment first inside a *batch* ($\mathscr{B}$) and then adjusts the weights according to the average of the batch to do fewer, but more stable, weight updates [53]. The *batch size* ($|\mathscr{B}|$) determines the number of weight adjustments that are first added to the batch before calculating the average weight that will be used for the update [53]. The weight update rule for mini-batch stochastic gradient descent can therefore be written as [53]:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{\alpha}{|\mathscr{B}|}\sum_{i\in\mathscr{B}}[v_\pi(S_i) - \hat{v}(S_i, \boldsymbol{w}_t)]\nabla\hat{v}(S_i, \boldsymbol{w}_t) \ . \tag{2.31}$$

Mini-batch stochastic gradient descent has the advantage of being vectorizable (improving computation time) and having more stable/accurate weight updates [53]. However, in Figure 2.17 it can be seen that certain loss functions might have weights that produce a global minimum (denoted as $\boldsymbol{w}*$) while other sets of weights can lead to a local minimum (only the minimum of the function for a certain local region). Therefore, stable gradient descent weight updates in Figure 2.17 when $w = 1.5$ will yield the local minimum ($w = 1.0$) instead of the global minimum ($w = -1.1$). The alternative is to use stochastic gradient descent to at least have a chance of jumping over the local minimum with its less stable updates [53]. Convergence with stochastic gradient descent is guaranteed if the step size is small enough and decreases over time [6]. Stochastic gradient descent has the computational performance trade-off of doing faster multiplications and additions (instead of moving data slowly between buffers) at the cost of not being vectorizable [53]. Ultimately, the batch size is task dependent (a hyperparameter) with certain tasks preferring smaller batch sizes while other tasks require larger batch sizes [53].

Of course, the ultimate goal is to obtain function approximations of the action-value function for the agent to use in control tasks. The stochastic gradient descent weight

update rule for function approximations with action values is given by Sutton & Barto as:

$$\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t + \alpha[U_t - \hat{q}(S_t, A_t, \boldsymbol{w}_t)]\nabla\hat{q}(S_t, A_t, \boldsymbol{w}_t) \tag{2.32}$$

where U_t is the *update target* that approximates $q_\pi(S_t, A_t)$, since the exact value of $q_\pi(S_t, A_t)$ is unknown [6]. A possible choice for the update target is to set $U_t = R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}_t)$, which is the same approximation used in one-step SARSA [6]. The resulting equation of substituting the one-step SARSA update target into Equation 2.32 yields [6]:

$$\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t + \alpha[R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}_t) - \hat{q}(S_t, A_t, \boldsymbol{w}_t)]\nabla\hat{q}(S_t, A_t, \boldsymbol{w}_t)\ . \tag{2.33}$$

As usual with value based methods, the highest action-value can greedily be chosen to obtain the largest expected return i.e. $A^*_{t+1} = \text{argmax}_a\, \hat{q}(S_{t+1}, a, \boldsymbol{w}_t)$ [6]. However, it has already been seen in this subsection that it is quite resource intensive to search for the minimum of the loss function with gradient descent [6]. Instead, it is advised to only apply value based function approximation methods on tasks with discrete action spaces, where it is practical to search through all the action-values of a state for the optimal action [6].

Getting into the nitty-gritty details of using the weights ($\boldsymbol{w}$) to approximate value functions ($\hat{v}(S_t, \boldsymbol{w}_t)$ or $\hat{q}(S_t, A_t, \boldsymbol{w}_t)$) reveals that there are two popular approximation function implementations, namely linear functions and non-linear functions implemented with artificial neural networks (ANN) [7][6]. Linear functions are appealing due to their simplicity, easy gradient calculations, having loss functions that have a single global minimum (no local minima), and have guaranteed convergence that non-linear methods don't have [7][6]. Non-linear functions are used when complex value functions need to be approximated [6][7]. Generally, simpler models are preferred to complex models in industry since they are simpler to implement, well understood and more robust [7].

Linear function approximation contains d - *basis functions* represented by *feature vectors* $\boldsymbol{x}$ [6]. The feature vectors are dependent on the state when used to approximate state-value functions ($\boldsymbol{x}(s)$) or can be dependent on a state-action pair when used for action-value approximation ($\boldsymbol{x}(s, a)$) [6]. The essence of linear methods is to take the dot product of the feature vector and the weights to obtain the approximation of value functions as shown by [6]:

$$\hat{v}(s, \boldsymbol{w}) \doteq \boldsymbol{w}^\top \boldsymbol{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s) \tag{2.34}$$

$$\hat{q}(s, a, \boldsymbol{w}) \doteq \boldsymbol{w}^\top \boldsymbol{x}(s, a) \doteq \sum_{i=1}^{d} w_i x_i(s, a)\ . \tag{2.35}$$

It should be noted that the components of the feature vectors are independent of one another and cannot interact with each other when doing the dot product for the function

approximation [6]. Therefore, several approaches have been developed to ensure that the components of the feature vectors are constructed with combinations of states (when $\boldsymbol{x}(s)$) or state-action pairs(when $\boldsymbol{x}(s,a)$) that interact with each other. For example, polynomial base feature representation can be used with a state vector $s = (s_1, s_2)^\top$ to construct a feature vector $\boldsymbol{x}(s) = (1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1^2 s_2^2)^\top$ [6]. Prior domain knowledge can even be incorporated to construct feature vector components with known mathematical formulas [6]. Other feature vector constructions include Fourier basis, course coding, tile coding or radial basis functions [7][6]. A further advantage of linear model using feature vectors is that the gradient of the approximation function is the feature vector itself ($\nabla_{\boldsymbol{w}} \hat{v}(s, \boldsymbol{w}) = \boldsymbol{x}(s)$ and $\nabla_{\boldsymbol{w}} \hat{q}(s, a, \boldsymbol{w}) = \boldsymbol{x}(s, a)$), making the calculations of Equation 2.30 and Equation 2.33 trivial and computationally efficient [6].

However, a non-linear artificial neural network (ANN) is able to automatically determine interactions among input features without requiring the handcrafted feature vector of linear methods [6]. Automatic identification of feature interactions, and the flexibility of approximating more complex functions, makes an ANN appealing despite the added computation costs, being unstable when training, difficulties in obtaining the global minimum loss, large amounts of data training required, and complexities in calculating the gradient of the approximation functions [7][6][53][54].

A variety of ANN architectures exist, with notable implementations being the multilayer perceptron (MLP), convolutional neural network (CNN), and recurrent neural networks (RNN) [6]. MLPs are the vanilla architecture, while CNNs are applicable to high-dimensional data that is locally correlated, such as images [7][6]. CNNs are therefore best suited for images, natural language, and time series applications [7]. RNNs contain internal feedback loops so that the network has memory of the previous inputs that have been given to the network [7]. Subsequently, the memory capabilities of RNNs loosen the Markov property, which assumes that decisions are independent of previous states. An alternative to RNNs could be to add input features to the ANN that are delayed versions of the other input features [7][54].

Figure 2.18 illustrates the general architecture of an MLP. It can be seen from Figure 2.18 that the nodes are organized into layers/columns and that the first layer is the input layer, the last layer is the output layer, and that every layer in between the input and output layers is known as the hidden layers (two hidden layers are shown). Each weight arrow in Figure 2.18 represents a component of the weight vector ($\boldsymbol{w}$) that multiplies with the value of the previous node so that the value of the next node is calculated with the weighted sum of previous nodes that points to the next node [6].

It can also be observed from Figure 2.18 that the value of a hidden node is first fed through a *non-linear activation function* before it is passed on for the next weighted sum calculation [6]. Non-linear activation functions are required to give the ANN its non-linear

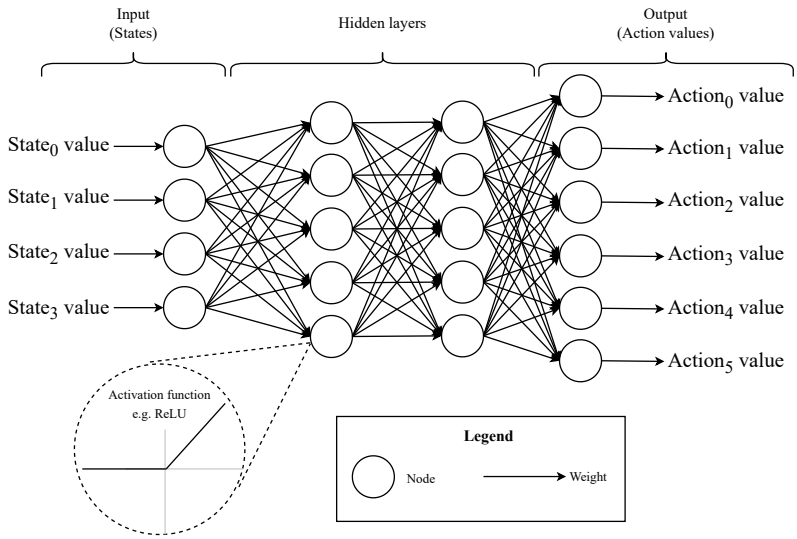


Figure 2.18: Example of an MLP neural network with two hidden layers

approximation properties [6]. Common activation functions include the sigmoid/logistic function ($f(x) = \frac{1}{1+e^{(-x)}}$) and the ReLU activation function ($f(x) = \max(0, x)$) [6].

The hidden layers of Figure 2.18 are credited for enabling an ANN to automatically learn the relationships and interactions between features without requiring handcrafted feature engineering[6]. It is theoretically possible to use only one hidden layer with many hidden nodes to approximate any continuous function on a compact region of the input space, but it is preferred in practice to rather use several hidden layers with fewer hidden nodes per hidden layer [6]. The hidden layers are able to build abstract representations of the data between hidden layers [6]. Networks with many hidden layers are known as *deep networks*. However, deep networks are sometimes suboptimal compared to shallow networks since the additional weights of the additional hidden layers can cause overfitting and make the gradient calculations of the weight adjustments harder (cause vanishing gradients) [6]. Sutton & Barto therefore suggest to rather only use between 1-2 hidden layers [6].

The complexities and flexibility of ANN cause the gradient calculations of the gradient descent weight update rule (Equation 2.30 and Equation 2.32) to become too complicated to calculate by hand [7]. Therefore, libraries and frameworks (such as TensorFlow or PyTorch) are rather employed to do these gradient calculations since one can easily make an error when trying to implement the gradient calculation algorithms yourself [7]. The primary gradient calculation approach employed by these libraries is to use automatic differentiation, more specifically back-propagation [7]. Other gradient calculation approaches include finite difference calculations and symbolic differentiation [7].

It should be remembered that there are a few other caveats with ANNs such as requiring lots of data to train, overfitting on training data, sensitivity to initial conditions, sensitiv-

ity to hyperparameters, no resiliency/robustness/convergence guarantees, and being open to adversarial attacks [7]. Overfitting occurs when the approximation function has too many degrees of freedom for a limited amount of training data [6]. ANN overfitting can be mitigated with cross validation (using validation and training datasets), discouraging complex approximation functions (regularization), sharing ANN weights and dependencies (reducing the approximating function's degrees of freedom), employing dropout (randomly kills nodes during training and observes the importance of that node) [6].

Further techniques used to improve the above-mentioned ANN limitations include using deep belief networks to train deep networks with many hidden layers, using double Q-learning with two ANNs to stabilize weight updates, injecting noise into the weights of the network to encourage exploration, employing transfer learning with initialized weights to provide superior initial conditions, and normalizing network inputs to be between -1 and 1 to improve generalization performance [7][6].

However, one fundamental technique that can improve the convergence of ANN in RL tasks is the use of an experience replay buffer (Figure 2.19) [54]. The experience replay buffer contains the last n observations, from which random samples are drawn and used for the batch update of an ANN [7][54]. Randomly sampling from the experience replay buffer (instead of using the latest observations) breaks the correlation between samples so that data is more independent and identically distributed (IID) [7][54]. IID data is preferred when using optimizers, such as stochastic gradient descent, since the approximation function is fitted to represent all the data and not only the recent observations (where the states are very similar due to them being recorded in quick succession one after the other) [7].

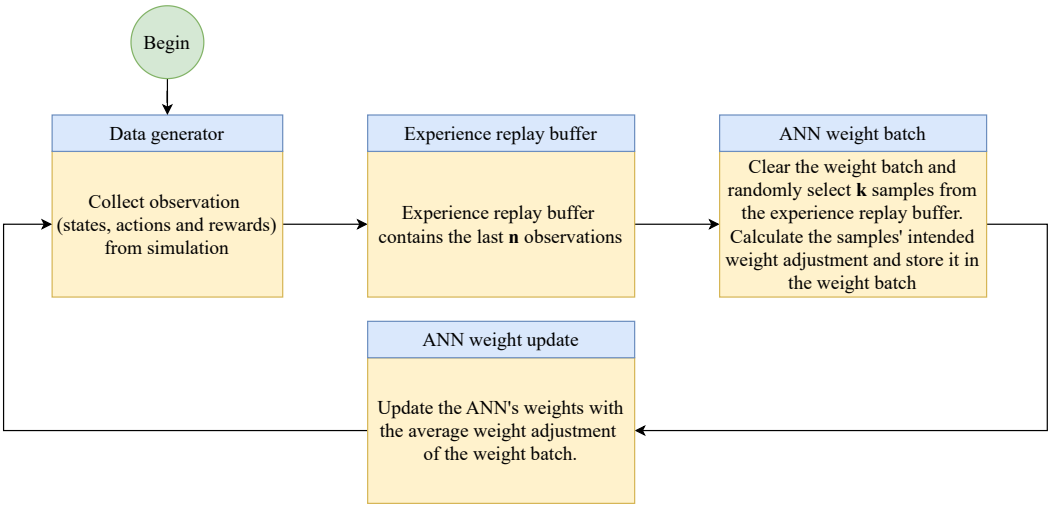


Figure 2.19: High-level illustration of the experience replay buffer

To summarize, function approximation RL algorithms are an extension to tabular approaches by catering for continuous state problems. The two main function approximation methods are linear approximation and non-linear ANNs. Linear approximators are appealing due to their simplicity and robustness while ANNs are a more hand-off solution that is capable of approximating functions that are more complex. The quality of an approximation is given by the loss function. The approximation with the smallest loss value is considered to be the optimal approximation. The following subsection investigates an alternative to value based methods, so that RL can be applied on problems with continuous state and action spaces.

Policy gradient methods

All RL algorithms considered till now heavily depend on the concept of value functions. However, the previous section showed that value-based methods are unable to handle large/continuous action spaces with existing technology due to the computational complexities of searching for the maximum action-value when going through all the possible action values. An alternative is to rather focus on constructing/improving the policy instead of the value functions [6]. Methods that aim to directly improve the policy are known as *policy gradient* methods and can become a bit abstract since the gradient of the policy is calculated in order to determine how the policy should be improved [6]. The advantages of policy gradient methods are their ability to be applied to continuous action spaces, having stronger convergence guarantees when compared to value-based methods (action probabilities of continuous policy parametrization change smoothly), and being able to directly learn stochastic policies that naturally encourage exploration [7][6]. The optimality of policy gradient methods versus value-based methods is problem dependent, with some problems having simpler policy representations while other problems have simpler value function representations [6].

In policy gradient methods, the policy itself is parametrized with the weights $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ so that $\pi(a|s, \boldsymbol{\theta}) = \Pr\{A_t = a | S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$ [6]. This parameterized policy shows that the weights can be used to approximate the policy with a linear function or an ANN, just like it was done in the function approximation subsection. Strictly speaking, value functions don't even need to be calculated with policy gradient algorithms [6]. The policy may be parametrized in any way as long as the $\pi(a|s, \boldsymbol{\theta})$ is differentiable with respect to $\boldsymbol{\theta}$ [6]. It is further advised that the policy should not become completely deterministic ($\pi(a|s, \boldsymbol{\theta}) \in (0, 1)$) to ensure that natural agent exploration remains possible [6].

Policy gradient RL methods can be applied to tasks with discrete action spaces by associating each output value of the approximating function with the probability of choosing an action (opposed to value-based methods that provide the action-value as output) [6]. The *soft-max* function is usually employed to convert the output of the function approximations

to probability values, as is done with:

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}} \tag{2.36}$$

where $h(s,a,\boldsymbol{\theta}) \in \mathbb{R}$ is known as the *numerical preference* and is implemented with approximating functions such as linear functions ($h(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{x}(s,a)$) or ANNs (input is the state, outputs are the probability of choosing each action and $\boldsymbol{\theta}$ symbolize the weights of the network) [6][7].

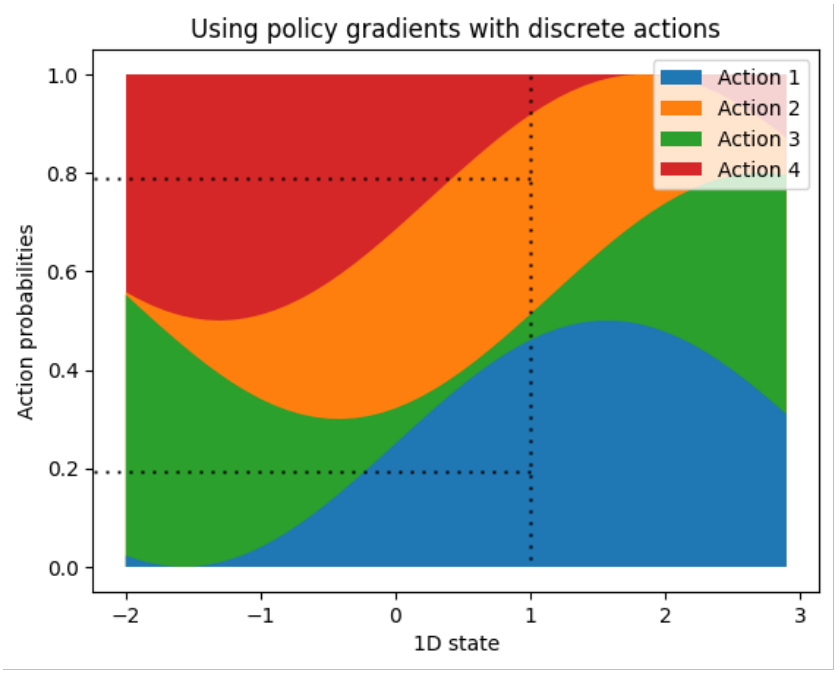


Figure 2.20: Example of policy gradients using discrete action probabilities for a 1D state

An example of using a policy gradient function with a discrete action space and a given set of weights is shown in Figure 2.20. The policy function must be supplied with a state and a set of weights, after which it provides the probabilities for choosing each action. Figure 2.20 presents the accumulated probabilities of each action so that the range of the graph is between zero and one. A random value between zero and one can then be drawn for a given state to determine which action should be chosen based on the action probability range in which the random value lies.

Taking Figure 2.20 as an example where the agent is in the 1D state equal to 1 and a random value between 0 and 1 is drawn. If the random value is equal to 0.2, Figure 2.20 shows that action 1 should be taken, but a random value equal to 0.8 indicates that action 2 should be taken. These example values are indicated with dotted lines in Figure 2.20. It can subsequently be seen that the agent has a higher chance of choosing action 1 or 2 compared to 3 and 4 when it is in the 1D state equal to 1.

The main advantage of using the soft-max function for discrete action policy gradient algorithms is that the policy will naturally approach determinism if a deterministic policy is optimal, or it will remain stochastic if the optimal policy is stochastic (such as in card games where there is imperfect information/partial observability) [6]. Stochastic policy also provides a natural exploration mechanism whereas deterministic policies might chatter (indecisive about what the optimal action is, which causes it to switch between the various options as training continues) [7].

It was shown in the previous subsection that the weights ($\boldsymbol{w}$) of a function approximation can be found by minimizing the loss function ($L(\boldsymbol{w})$) with gradient descent. Policy gradient methods follow a similar approach to obtain the policy's weights ($\boldsymbol{\theta}$) by trying to maximize the *objective function* ($J(\boldsymbol{\theta})$) with *gradient ascent* [6]. It will be seen in this subsection as well as in Section 4.3, that the objective function is the essence of a policy gradient method's RL algorithm.

The vanilla objective function for episodic tasks is defined as $J(\boldsymbol{\theta}) \doteq \mathbb{E}[G|s,a]$ [7]. Subsequently, the update rule of gradient ascent ($\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$) requires the gradient of the objective function [7]. The gradient of this objective function can be expanded using [7]:

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &\doteq \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi}[G|s,a] \\
&= \nabla_{\boldsymbol{\theta}} \sum_{a \in A} [Q(s,a)\pi(a|s,\boldsymbol{\theta})] \\
&\propto \sum_{a \in A} [Q(s,a)\nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})] \\
&\propto \sum_{a \in A} \left[Q(s,a)\nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}) \times \frac{\pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})} \right] \\
&\propto \sum_{a \in A} \left[Q(s,a)\pi(a|s,\boldsymbol{\theta}) \times \frac{\nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})} \right] && (2.37) \\
&\propto \sum_{a \in A} [Q(s,a)\pi(a|s,\boldsymbol{\theta}) \times \nabla_{\boldsymbol{\theta}} \ln(\pi(a|s,\boldsymbol{\theta}))] && (2.38) \\
&\propto \mathbb{E}[G \times \nabla_{\boldsymbol{\theta}} \ln(\pi(a|s,\boldsymbol{\theta}))] \ . && (2.39)
\end{aligned}
$$

Therefore, Equation 2.39 states that the gradient of the objective function is proportional to the size of the expected return times the direction in which the policy increases [7]. The term $\nabla_{\boldsymbol{\theta}} \ln(\pi(a|s,\boldsymbol{\theta}))$ in Equation 2.39 is known as the *eligibility vector* and was obtained by applying the $\nabla \ln x = \frac{\nabla x}{x}$ differentiation rule between Equation 2.37 and Equation 2.38 [6]. The natural logarithm used in the eligibility vector is especially helpful for reducing numerical accuracy errors when working with small gradients/probabilities [7]. The proportionality constant of Equation 2.39 can be incorporated into the step size to yield the general weight update rule for policy gradients so that [7]:

$$
\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha [G_t \times \nabla_{\boldsymbol{\theta}} \ln(\pi(A_t|S_t,\boldsymbol{\theta}_t))] \ . \tag{2.40}
$$

The gradient in Equation 2.40 can again be calculated with methods such as automatic differentiation, finite difference methods or symbolic differentiation. The return (G) of Equation 2.40 can be calculated in a variety of ways, such as episodic task using the Monte Carlo/REINFORCE algorithm to update the weights at the end of the episode with $G_t = \sum_{k=t+1}^{T} \gamma^{k-1} R_k$ (similar to the Monte Carlo approach discussed in Section 2.3.1) [7][6].

However, one shortcoming of Equation 2.40 is that the gradient updates produced by the return G can be large and lead to great variance and instabilities between updates [7][6]. A mitigation for these instabilities is to rather make smaller gradient updates by measuring the return relative to a *baseline* [7]. Any return larger than the baseline is an improvement (encouraged) while anything smaller than the baseline is a deterioration (discouraged).

The baseline must be independent of the action taken to ensure that it does not affect the expected value of the policy's gradient by introducing a bias [6]. Subsequently, the baseline is only dependent on the state and is denoted as $b(s)$ [6]. Updating Equation 2.40 to incorporate baselines yields [6]:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left[(G_t - b(S_t)) \times \nabla_{\boldsymbol{\theta}} \ln \left(\pi(A_t | S_t, \boldsymbol{\theta}_t) \right) \right] \ . \tag{2.41}$$

A suitable choice for the baseline function can be the state value function ($\hat{v}(S_t, \boldsymbol{w})$), with the weights given by $\boldsymbol{w} \in \mathbb{R}^d$ [6]. This state value baseline function reintroduces value functions in the context of policy gradient methods [6]. It should be emphasized that the weights of the approximate state value function ($\boldsymbol{w}$) can be different from the weights of the policy function ($\boldsymbol{\theta}$) [6]. The update rule for these weights is consequently given by:

$$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}} \times (G - \hat{v}(S_t, \boldsymbol{w})) \times \nabla \hat{v}(S_t, \boldsymbol{w}) \tag{2.42}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \times (G - \hat{v}(S_t, \boldsymbol{w})) \times \nabla \ln(\pi(A_t | S_t, \boldsymbol{\theta})) \tag{2.43}$$

where it can be observed that the update functions can have different step size values [6]. It is advised that the step size of the state-value function approximation ($\alpha^{\boldsymbol{w}}$) should be larger or equal to the step size of the policy ($\alpha^{\boldsymbol{\theta}}$) to ensure stable updates [7].

A special case, known as *actor-critics* exists for baseline algorithms where the return (G) is approximated with the one-step expected return ($G_{t:t+1}$) [6]. Replacing G in Equation 2.43 with $G_{t:t+1} = R_t + \gamma \hat{v}_\pi(S_{t+1}, \boldsymbol{w})$ (assuming that the optimal policy is followed) yields:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \times (R_t + \gamma \hat{v}_\pi(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})) \times \nabla_{\boldsymbol{\theta}} \ln \left(\pi(A_t | S_t, \boldsymbol{\theta}_t) \right) \tag{2.44}$$

whereas using $G_{t:t+1} = R_t + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w})$ in Equation 2.43 results in [7][6]:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \times (R_t + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})) \times \nabla_{\boldsymbol{\theta}} \ln \left(\pi(A_t | S_t, \boldsymbol{\theta}_t) \right) \ . \tag{2.45}$$

An important aspect of Equation 2.44 and Equation 2.45 is their ability to update the weights of the policy and value functions at each step instead of waiting till the end of the episode like Equation 2.43 [7]. Furthermore, in Equation 2.44 and Equation 2.45 the state-value function of the current state ($\hat{v}(S_t, \boldsymbol{w})$) is referred to as the *critic* while the value function of the next state ($\hat{v}_\pi(S_{t+1}, \boldsymbol{w})$ or $\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w})$) is known to be the *actor* [6]. The actor is the decision maker while the critic is the baseline that the actor is compared against [6][7]. Again, actor values larger than those of the critic are considered to be an improvement (encouraged) while actor values smaller than that of the critic are considered to be inferior (discouraged). In fact, the general concept of comparing the action-value against the state-value appears often enough that it is named the *advantage function* ($A_\pi(s, a) \doteq Q(s, a) - V(s)$) [7]. Actor-critic implementations using eligibility traces, instead of n-step methods, also exist [6].

Continuous action spaces can be represented with policy gradient methods by learning the statistics of the action's probability distributions instead of the action probabilities [6]. For instance, the mean ($\mu(s, \boldsymbol{\theta})$) and standard deviation ($\sigma(s, \boldsymbol{\theta})$) must be provided when assuming that the action is normally distributed [6]. The equations relating to the policy, mean, and standard deviation of continuous action spaces with policy gradient methods are provided as [6]:

$$\pi(a|s, \boldsymbol{\theta}) \doteq \frac{1}{\sigma(s, \boldsymbol{\theta})\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s, \boldsymbol{\theta}))^2}{2\sigma(s, \boldsymbol{\theta})^2}\right) \tag{2.46}$$

$$\mu(s, \boldsymbol{\theta}) \doteq \boldsymbol{\theta}_\mu^\top \boldsymbol{x}_\mu(s) \tag{2.47}$$

$$\sigma(s, \boldsymbol{\theta}) \doteq \exp\left(\boldsymbol{\theta}_\sigma^\top \boldsymbol{x}_\sigma(s)\right) . \tag{2.48}$$

Equation 2.46 shows how a continuous action policy is represented by the normal distribution, while Equation 2.47 illustrates how the mean can be approximated with a linear function [6]. Also, the standard deviation can for instance be approximated with Equation 2.48, which instead uses the exponential of a linear function since the standard deviation is always positive [6].

This marks the end of the fundamental RL concepts to date. These concepts will likely be expanded upon in future research since RL is still an active research field with a strong focus on improving the understanding/performance of RL algorithms [7]. For instance, P. Winder suspects that future theories/understanding will bridge value-based functions with policy gradient methods so that they become inherently interlinked [7]. As for the rest of the dissertation, the next subsection mentions a subgenre of RL that uses multiple agents, known as MARL. State-of-the-art policy gradient RL algorithms that are used to implement the RL agents of this study are discussed in Section 4.3.

2.3.3 Multi-agent RL

The start of Section 2.3 mentions that the main motivation for using RL in this study is to develop an AI opponent for the player to play and learn against. The premise of a player against an opponent means that the serious game is a two-player game and is modelled by two agents (the human player and the AI opponent). Having more than one agent in an RL task introduces the sub-field of RL known as multi-agent reinforcement learning (MARL) [7]. As seen with game theory (Section 2.2.4), MARL tasks can be competitive, cooperative or a mixture of both [7]. The MARL task of this study is subsequently classified as a zero-sum competitive two-player game.

The first difficulty of MARL tasks is finding a performance metric to evaluate agents with. Figure 2.21 illustrates that the average sum of rewards per episode is a performance metric often employed with single player agents [7]. However, this metric falls short for MARL tasks since the performance of one agent directly influences the environment and performance of another agent (agents are not independent) [7]. For example, an average tennis player will look like a professional when playing against a novice player (win most games and obtain a high reward), but the same average player will look like a novice when playing against an actual professional (lose most of the games and obtain negative rewards).

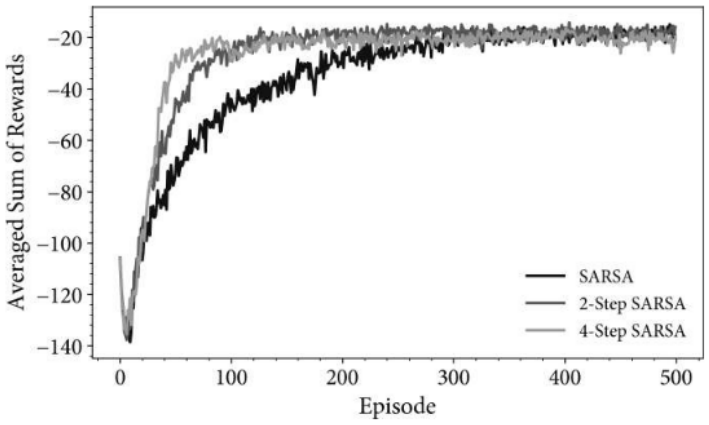


Figure 2.21: Example single agent performance measure - image obtained from [7]

An alternative is to incorporate the skill difference of players into the performance metric by using match making metrics such as ELO, Glicko or TrueSkill [5][55][56]. The ELO rating system was traditionally used to rank chess players, but can be applied to other two-player zero-sum competitive games [5]. A player's ELO rating can even predict the probability of a player winning a game by using:

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}} \tag{2.49}$$

where $E_A \in \mathbb{R} | 0 \leq E_A \leq 1$ is the expected score/probability of winning for player A, R_A is the ELO rating of player A, and R_B is the ELO rating of player B [5][50]. Therefore, Equation 2.49 shows that the chance of winning/losing increases as the difference between the ELO ratings of players becomes greater. The ELO rating of player A can subsequently be updated with:

$$R'_A = R_A + K(S_A - E_A) \tag{2.50}$$

whereas the ELO rating of player B is calculated in the same manner, except by replacing all references to A in Equation 2.50 with B [5][50].

New terms appearing in Equation 2.50 is the player's updated ELO rating (R'_A), the player's performed score (S_A), and the adjustment rating/K-factor (K) [50]. The performed score is dependent on the outcome of the game with $S_A = 1$ for a won game, $S_A = -1$ for a lost game and $S_A = 0$ for a drawn game [5][50]. Also, the expected score E_A determines the size of the update since a player should receive a smaller rating increase when winning a lower rated opponent compared to winning a higher rated opponent. The K-factor is used to amplify the score adjustment and is commonly decreased as the number of games played by the player increases (heavy adjustments for new players and minor adjustments for experienced players) [5].

The desired effect of using the ELO rating to define two-player adversarial MARL performance is shown in Figure 2.22 [55]. Notice the similarity between the MARL performance characteristics of Figure 2.22 and the single agent performance characteristics of Figure 2.21. Both figures show the agents' performance characteristics being reminiscent of a smoothed step function with drastic initial performance improvements that taper off in the end.

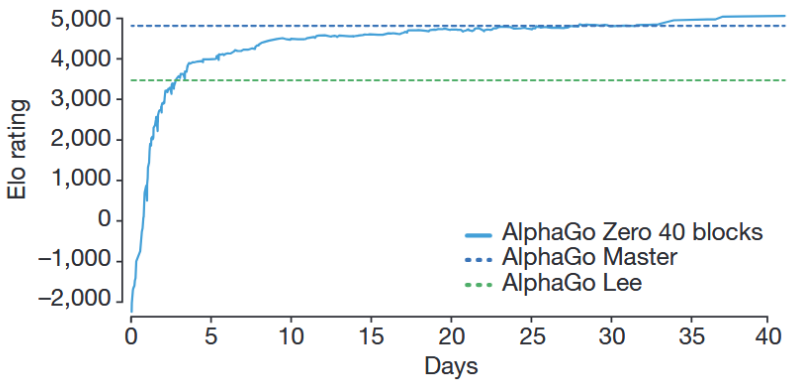


Figure 2.22: Example ELO performance measure of MARL task - obtained from [55]

The second issue of MARL is that the environment is non-stationary due to the skill level and strategy differences between opponents [7]. The non-stationary environment makes it somewhat difficult to choose an appropriate opponent to train against since a veteran opponent will demolish the agent while an inferior opponent making random moves

presents no learning opportunities for the agent [57]. Instead, a suitable opponent can be the agent itself since the skill level of the agent and its opponent will be the same and increases at a similar pace as they duel against each other [57]. The technique of using the agent to train against itself is known as *self-play* [57]. For example, self-play can be seen in practice when a tennis player trains against a wall, since the wall imitates the tennis player by returning the same strokes that it receives.

The Self-play environment can be made somewhat stationary by presenting the agent with a single static opponent to train with for a period of time, before updating the opponent to incorporate the new knowledge that was learned (raising the opponent's skill level as well) [58]. The collection of opponents used in the training creates a natural curriculum for the agent to learn from by starting with easy/random tactics and advancing to more advanced strategies as the opponents improve [57][55]. However, it is vitally important that the agent doesn't only train against the best opponent, since the agent may over-fit to that opponent and even begin to forget foundational strategies that were previously learned against easier opponents [57][58].

The solution to this phenomenon is to rather have a collection of opponents with which the agent trains a certain amount of time against the newest/best opponent and the other times against one of the earlier opponents [57][58]. In general, introducing randomness into the environment (such as swapping opponents) improves the generalizability of the agent, but at the cost of also increasing the instability (non-stationary environment) of the training process [58]. Hyperparameters for training an agent with self-play using the Unity game engine's ML-Agents RL framework are given in Table D.4 of Appendix D[58].

A concluding remark for this subsection is that MARL can become quite complex to solve, in addition to normal RL difficulties, since MARL problems have a broader variety of potential solutions, the goals of agents can differ, difficult to synchronize information between agents efficiently, instability of non-stationary environments (agents affect the environment of each other), and there are no convergence guarantees for non-trivial MARL environments [7]. The phenomenon of agents changing the environment of other agents makes MARL inherently a partially observable MDP, which means that the optimal policy in MARL tasks is most likely stochastic in order to account for effects that cannot be observed [59]. The interaction between agents can also make MARL tasks much more complex than the environments in which they take place [57]. Fortunately, Section 2.3.4 presents some case studies of successful state-of-the-art MARL integrations that can outperform humans in games. These MARL integrations show that it is possible to successfully apply MARL (with all of its difficulties and complexities) if one possesses grit and determination.

2.3.4 Case studies of RL usage in games

This section discusses milestones of applying RL and MARL/self-play in adversarial games. It should be noted that the case studies had the primary goal of beating the best human players in the world. Aiming to beat humans with RL has the side effect of forcing the agents to adopt strategies that are unknown to human players in order to gain a tactical advantage. These strategies can then be used to teach human players and raise the overall skill level of the game. Case studies presented in this section include TD-Gammon, Deep Q-network on Atari 2006 games, AlphaGo, and OpenAI Five.

TD-Gammon

One of the first milestones of adversarial agents using RL is TD-Gammon, which is trained to play Backgammon [6]. TD-Gammon was developed in the 1990s and had to overcome problems such as opposing players changing the state of the board (the state of the board isn't determined entirely by the action taken by the agent) as well as difficult strategies such as blocking the opponent with the game pieces [6]. Furthermore, the state space of the game was very large (30 game pieces that can be at 26 possible locations with 20 actions that can normally be taken), which made it impossible to store the state space in the memory elements of any physically realizable computer [6].

Fortunately, the complete state of the game is observable at all times, although the game is highly stochastic [6]. The full observability of the game made it possible to use the TD(λ) reinforcement learning algorithm with a non-linear multilayer artificial neural network that is trained with the back-propagation of TD errors. The neural network had 198 input units that represented the state of the board and provided the probability of winning the game in that state [6]. A reward is only given to the agent when a game is won [6]. The weights of the network are initialized with random values and cause the agent to initially make random choices [6].

The agent plays against itself to train and is expected to become as good as the best backgammon computer program after $\approx$ 300 000 games. The algorithm underwent various iterations with version 1.0 adding specialized backgammon features, 2.0 and 2.1 adding more hidden nodes to the artificial network as well as two-ply search procedures[8], and version 3.0 and 3.1 adding more hidden nodes with three-ply search procedures [6]. TD-Gammon 3.0 was finally able to beat a professional backgammon player (Kazaros) after

[8] N-ply refers to the number of future player moves that an agent simulates when evaluating an action [6]. For instance, two-ply will simulate the outcomes of taking an action together with one player move thereafter, while three-ply simulates the different combinations of three player moves into the future. The n-ply concept exhibits a close resemblance to the n-step methods discussed in Section 2.3.2. N-ply is used to simulate the consequences of n number of future player moves while n-step is used to calculate the expected state/action return using n subsequent/future rewards [6].

training for ≈ 1 500 000 games [6]. TD-Gammon raised the skill ceiling of human tournament play by introducing new strategies that were previously unknown to professional players [6].

Deep Q-network on Atari 2006 games

An article released by Google DeepMind in 2015 discusses the use of Deep Q-learning to play Atari 2006 games using only the pixels of the screen and in-game score [54]. The motivation of the article was to find a method that can learn generalizable policies from high-dimensional sensory inputs [54]. The Deep Q-network architecture was successfully generalizable to many Atari 2006 games since the RL agents were able to train on many of the Atari 2006 games using the same algorithm, network architecture, and hyperparameters [54].

The Deep Q-network consists of a deep convolutional network that has several layers to learn the abstract representation of the data [54]. The deep convolutional network enables the network to use local spatial correlations of the images to build a network that is robust against changes in the viewpoint or scale of the image [54]. However, using non-linear function approximators, such as an artificial neural network, tends to yield unstable or divergent results in reinforcement learning [54]. Unstable/divergent training with ANN in RL can be mitigated by using an experience replay buffer and only iteratively updating action values towards target values that are only periodically updated [54].

Algorithm performance was further measured by taking the average game score per episode, due to the games being single-player games [54]. The t-SNE algorithm was thereafter run on the expected state-values of the different states to group states with their expected value [54]. Consequently, the t-SNE algorithm showed that visually similar states have similar expected rewards and that visually dissimilar states may also have the same expected rewards [54]. The relationship between the visual states and their expected rewards illustrates that the Deep Q-network can "learn representations that support adaptive behaviour from high-dimensional sensory inputs" [54].

The RL agent played some Atari 2006 games, such as Breakout, exceptionally well due to unexpected strategies that were learned by the agent (as can be seen at `https://www.youtube.com/watch?v=TmPfTpjtdgg`) [54]. Other games, such as Montezuma's Revenge, stifled the RL agents' performance since those games required "more temporally extended planning strategies" [54].

AlphaGo

Go is a traditional Chinese board game that has a large state space and cannot be solved with a brute-force method as has been done with Chess [60]. The Google DeepMind team managed to overcome this obstacle by implementing a reinforcement agent (named AlphaGo) that was able to defeat the reigning human Go champion (Lee Sedol) 4-1 in March 2016 [55].

The AlphaGo agent was implemented with two deep neural networks [55]. The first network was used to predict move probabilities (policy/actor network), while the second network provided the value of a position (value/critic network) [55]. The two networks were combined with a Monte Carlo tree search to investigate future outcomes of choices by only looking at highly probable moves of the policy network and evaluating the eligibility of a move with the value network and Monte Carlo rollouts [55]. It should be noted that the policy network was trained with supervised learning using the games of expert Go players in order to accurately predict the moves of human experts.

An improved version, named AlphaGo Zero, is able to train solely by self-play instead of requiring the data of human expert players or domain knowledge beyond the game rules [55]. AlphaGo Zero further required only the black and white stones of the board as input features, a single neural network, and used a simpler tree search that does not require any Monte Carlo rollouts [55]. These improvements enabled AlphaGo Zero to beat AlphaGo Lee 100-0 [55]. AlphaGo Zero is superior in terms of training speed and efficiency, since AlphaGo Zero can beat AlphaGo Lee when trained for only 36 hours on 4 tensor processing units (TPU), as opposed to months of training on 48 TPUs.

Therefore, AlphaGo Zero was able to rediscover many of the traditional Go knowledge and provide new strategies that can help professional players with new tactics [55]. Regrettably, AlphaGo had detrimental effects such as causing the world's best human Go player (Lee Sedol) to retire due to AI being undefeatable and superior to human Go players [60].

OpenAI Five

Dota 2 is a multiplayer real-time strategy game that is played by two teams (five players in a team) against each other [56]. Dota 2 is challenging for reinforcement learning agents due to "long time horizons, partial observability, and high dimensionality of observation and action spaces" [56]. Nevertheless, OpenAI managed to beat the human Dota world champions using five reinforcement learning agents (thus the name OpenAI Five) in April 2019 [56]. OpenAI Five further managed to win 99.4% of the 7000 games that it played against the online Dota community [56].

The implementation of OpenAI five consisted of a single layer 4096-unit LSTM with 159 million parameters [56]. The reinforcement learning policy was trained with Proximal Policy Optimization (PPO) using Generalized Advantage Estimation (GAE) [56]. The training was based entirely on self-play (requires no human data) and trained for 10 months (sometimes using as much as 1536 GPUs) [56]. Dota was often updated with new features during this training period, which required OpenAI to perform "surgery" on the agent in order to incorporate the new features with the existing knowledge of the agent [56].

Some techniques used by OpenAI to ease the implementation of the RL agent include giving specific rewards and penalties for specific events in the game, training the agents in randomized environments, using diverse agents to train (not training with just the best agents but also previous inferior agents), and limiting games to only a subset of heroes and items [56]. Human scripted elements of the agent include the choice of which item to buy, courier control and item reserve choices [56]. Lastly, OpenAI warns to be observant for bugs in implemented code since the RL algorithms may seem to work at first glance, but suffer from subpar performance due to bugs being present in the code [61].

2.4 Key conclusions from background

The background section of this study is quite long due to it covering three disciplines, namely the teaching of mathematics, designing serious games, and introducing reinforcement learning in a machine learning context. These background concepts played a key role in the design of the serious game. In particular, the teaching of mathematics section shows that there are various perspectives on how maths can be taught and that learners can even construct their own knowledge by playing with the mathematics. The DBE also defined maths as a language that consists of symbols and notations. The DBE's definition eventually led to the study's serious game becoming a card game, since the mathematical symbols are represented by the different cards of the game.

The game design section gives a brief introduction to game design by describing key principles such as the flow state, motivation and fun in video games, uniqueness of players, game mechanics, the relationship between luck and skill, deep game, and player friction. An overview of existing mathematical games is given in Section 2.2.3, where it is shown that the majority of mathematical games are aimed at young children and not high school learners. This scarcity of high-school mathematical serious games contributed to the choice of creating a mathematical serious game for grade 9 learners.

The reinforcement learning (RL) section shows that an AI opponent can be created to be a challenging opponent in the game. The AI opponent plays the role of a mediator by demonstrating to the player some strategies of the game while the player is playing

against it. The RL section further illustrates that RL tasks are modelled according to the Markov Decision Process (MDP) and that RL algorithms are used to obtain the policies that agents should follow. The RL section also categorized RL algorithms in three broad categories, namely tabular (temporal difference) methods, value function approximation methods, and policy gradient methods. Policy gradient methods are of particular interest to this study since they are used in the RL agent design of Chapter 4. Lastly, the RL background section contains case studies showing that it is possible to develop RL agents that can surpass human performance in multi-player adversarial games.

Chapter 3

Game design

This chapter presents the serious game designed in this study. The game is named *Digit Draw* to emphasize that it is a card game about numbers/mathematics. *Digit Draw* is a two-player game, with the second player always being an AI opponent that has been trained with reinforcement learning (RL) (see Chapter 4 for the RL agent design). Much effort has been invested to ensure that *Digit Draw* follows the foundational rules of mathematics as closely as possible.

The content of this chapter is divided into six sections. The first section describes the goals and requirements of the game, and the second section discusses game genres that can fulfill these requirements. The core game concept and rules are presented in the third section, while the fourth section exhibits all the cards that are available in the game. The fifth section mentions auxiliary game mechanics that are added to the core game, and the sixth section considers mechanisms that are implemented to retain players in the game. Finally, a summary of this chapter is presented in the last section.

3.1 Game design goals and requirements

A Game Design Document (GDD) was drafted before commencing with the game's development to ensure that the vision of the game is not lost during development. The game design documentation is strongly influenced by the background presented in Chapter 2.

First off, the vision of *Digit Draw* is given in the GDD as :

> A mathematical game that can be used by high school learners to gain a feel for mathematics instead of just memorizing the rules. The game will enable the high school learners to play with the mathematics in a fun and interactive manner. The players will be able to make the mathematics their own and

express the mathematics in their own words in a way that makes most sense to them. Ultimately, the players must feel like they discovered the mathematical rules by themselves.

An AI opponent will serve as an exciting challenger for the player to outwit. The AI opponent will have the auxiliary role of being a mentor to the player by presenting interesting moves and solutions that the player can learn from. The player and AI opponent will compete for the same goal.

The vision of *Digit Draw* led to the following game design pillars. These pillars are the core concepts used in the game to support the vision of *Digit Draw* and every choice made in the game considered how it affected the game design pillars. The game design pillars are stated as:

- Reinforcement learning for the AI opponent and the player
- Explicit representation of the mathematics
- Feedback and scaffolding
- Desirable difficulties
- Construction/creative mechanics to create deep game with random/procedurally generated content
- Simple and intuitive input

Therefore, the design requirements of *Digit Draw* are presented below. The requirements are assigned into one of three categories, namely core requirements, technical requirements, and quality requirements. It should be reiterated that the game design requirements are influenced by the background of Chapter 2, the vision of *Digit Draw* in the GDD, and the game design pillars of the GDD.

1. **Core requirements**
   (a) Game rules and mechanics shall align naturally with those of mathematics
   (b) The game shall present the mathematics in a clear and explicit form to the player
   (c) There shall be 2+ players in the game
   (d) Opponents shall have the same type of goal as the player
   (e) The opponents shall be controlled by the computer (AI)
   (f) The opponents shall be implemented with reinforcement learning
   (g) The game levels shall introduce the different mathematical concepts. Harder levels conveys more advanced concepts
   (h) The game shall not rely on the reflexes or dexterity of the player to win games, but rather on the strategy/understanding that the player has of the mathematics

(i) Game mechanics shall be based on desirable difficulties such as varying conditions of study, spacing the study, interleaving study tasks and using tests to learn the concepts

(j) The game shall use mediation and scaffolding to provide aid such as attracting the players' attention, simplifying tasks by reducing the number of choices, directing players back to the original objective, marking critical aspects of the problem, controlling the frustration of the player, and demonstrating the ideal solution

2. **Technical requirements**

(a) The core gameplay shall function without internet

(b) The game shall consume mobile data sparingly for non-critical functions where there is no other option

(c) The game shall be developed for low performance devices

(d) The game's audio shall not be compulsory to play the game

3. **Quality requirements**

(a) The game shall cater for Bartle's player model (explorers, achievers, killers and socializers)

(b) At least three of the four key elements of fun (hard, people, easy, serious) mentioned by Lazzaro shall be present in the game

(c) The game shall use input randomization to keep the game interesting

(d) The game shall keep output randomization to a minimum

(e) The game shall provide a variety of choices with no apparent meta strategy. Experience can guide the player on what choices are superior for the situation

(f) Choices made by the player shall have an impact on the outcome of the game

(g) Players shall receive clear feedback/rewards when they perform the correct action

(h) The game shall create a stress-free learning environment for the player to experiment/fail in order to learn

(i) The game shall provide feedback on how to solve/improve on the mathematical problem. Possible forms of feedback includes learning from the opponent's strategy or funnelling the player to the answer/solution.

(j) The game flow shall not be interrupted by prompts

(k) The game must prevent users from impulsive clicking without thinking about a solution

The following section procures a game genre that is able to satisfy these requirements.

3.2 Game genre

The game genre informs the player about the type of game they will be playing [4]. Example genres include platformers, role-playing games, strategy games, and first-person shooters [4]. Game genres can often be separated into sub-genres, and a game can belong to more than one genre [4].

Furthermore, requirement $1j$ of the previous section demands that the game should have a mediator to demonstrate how the game can be played, while requirements $1c$ and $1d$ state that the game should be a symmetric multiplayer game. Koster emphasizes these requirements by stating that most historic games have been competitive head-to-head activities since it is the easiest way to constantly supply new challenges and content to the player [44]. Requirement $1a$ further adds that the rules of mathematics should be the basis of the serious games. Thus, the majority of the game's rules should be encapsulated by the rules of mathematics, and the game's genre must support multiplayer gameplay.

At first, a suitable game genre was probed by considering older popular genres since these genres tend to resurface and become popular again [4]. The arcade genre was therefore thoroughly considered since its gameplay is iconic, designed to attract players' attention[1], and is usually simple games to understand and implement. Therefore, popular arcade games such as *Pacman*, *Space Invaders*, *Donkey Kong*, *Astroids*, *Defender*, *Centipede*, *Galaxian*, *Frogger*, *Tempest*, *Q*bert*, *Dig Dug*, *Pong*, and *Lunar Lander* were considered. Surprisingly, most of these games were not completely original and were very similar to *Space Invaders*, either in theme or gameplay. However, it was found that an arcade-like game would be difficult to implement due to the genre's fast-paced nature and high dexterity requirements. These attributes are in direct conflict with requirement $1h$ of the previous section.

Instead, the genre selection strategy was adjusted to consider the essence of what the game is actually about. For instance, the *Age of Empires* games simulate historic events. Therefore, a suitable genre for *Age of Empires* would be strategy, since it enables the player to experience history as a ruler fighting battles and conquering land.

Similarly, Section 2.1.1 states that language and symbols are the core of mathematics. Serious games related to programming (the language for computers) were subsequently studied to gain inspiration. Some serious games that were studied include *Human Resource Machine* and *Baba is You*. These games were interesting, but somewhat complicated to implement, due to their game depth being literal programming languages.

[1] Arcade games were normally played in an arcade full of other games. Therefore, developers of these games went to great lengths to ensure that their game draws more attention than those surrounding it.

The breakthrough genre of this study was collectable card games (CCG). CCG includes games such as *Magic the Gathering*, *Yu-Gi-Oh*, *Gwent*, *Marvel Snap*, and *Heartstone*. The CCG was inspiring due to the different types of cards that could be played. For instance, *Magic the Gathering* contains creatures, artefacts, instants, sorceries, and land cards [62], while a game such as *Yu-Gi-Oh* contains monster, spell and trap cards [63]. These different card types provide the functionality required to represent the core parts of a language such as verbs, nouns, adjectives and conjunctions. In the case of mathematical language, the card types could be numbers, variables, operators and functions. *Poker* was also influential due to its popularity and the way in which the value of a player's hand depends on the highest value card combination that the player recognizes.

Therefore, the card game genre was alluring for its ability to represent a language and its wider audience appeal (is not associated with people playing digital video games). However, drawbacks of the card game genre include that the game can quickly become very abstract, and the rules can be tiresome to learn if the player has to read the rules instead of being shown how to play. The next section proceeds with an exploration of *Digit Draw*'s game rules and design.

3.3 Core game loop

This section provides the core gameplay and rules of *Digit Draw* and consists of two subsections. The first subsection shows the fundamental gameplay without any variable cards, while the second subsection shows minor rule alterations that occur when variable cards are introduced. A complete list of the game's cards is given in Section 3.4.

3.3.1 Principle gameplay

The foundational rules of the game are showcased in this section. A screenshot of a starting state for the game together with labelled elements is presented in Figure 3.1 below. The foremost goal of the game is to alter the left board and right board so that the comparator sign of the player is satisfied when the game ends. The starting values of the left board and right board are randomized to make the games more interesting and to force the game participants to work with an unbalanced equation. Furthermore, the starting values of the left board and right board are randomized to be between -30 and 30. The absolute difference between the starting numbers of the left board and right board will always be between 1 and 4.

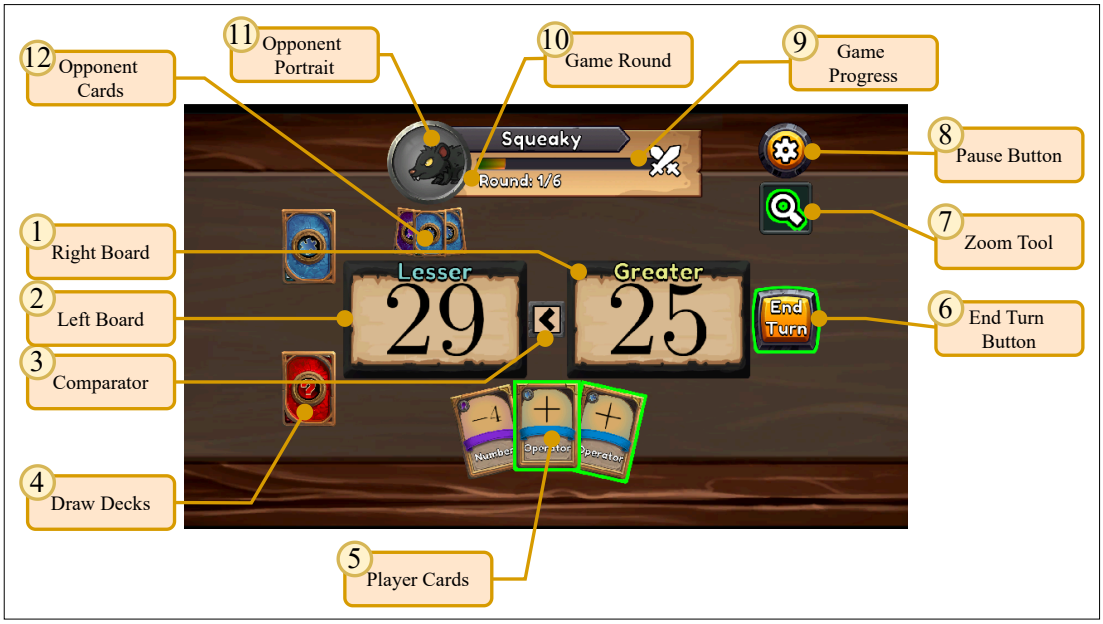


Figure 3.1: A starting state for the game

Each labelled item of Figure 3.1 carries the following significance:

1. **Right board** The right board represents a mathematical expression (such as "$x+12$" or "25" in the case of Figure 3.1). The player as well as the opponent can alter the right board with their cards.

2. **Left Board** The left board represents a mathematical expression. The player as well as the opponent can alter the left board with their cards.

3. **Comparator** The comparator's sign signifies the objective of the playing participant. The comparator together with the left board and right board forms a mathematical equation. The player and the opponent have opposite comparator signs, meaning that if the player's comparator sign is "¿", then the opponent's comparator sign will be "¡" and vice versa.

4. **Draw Decks** A game participant must draw a card from each deck at the start of their turn. The only exception is the first round of the game, where the playing participant doesn't draw cards at the start of their turn. There are three decks in the game, namely a blue, red and green deck. The decks of the game are further discussed in Section 3.4.

5. **Player Cards** The cards in the player's hands. A player can hold a maximum of 8 cards in their hand. Drawn cards are discarded when trying to add them to a hand that is already full.

6. **End Turn Button** Game participants indicate that they wish to end their turn by pressing this button.

7. **Zoom Tool** The zoom tool can be used to enlarge the display size of mathematical expressions written on the cards or the game boards. It is added for players playing on small mobile screen sizes.

8. **Pause Button** The pause button can be pressed to restart the game or to quit to the level selection menu.

9. **Game Progress** The progress bar indicates how many game rounds has passed and how far the game is from ending.

10. **Game Round** A game consists of 6 rounds. The playing participant is alternated between rounds. For instance, the opponent might play on even rounds while the player may play odd rounds and vice versa. The game participant whose inequality sign is unsatisfied at the start of round 1 is designated as the participant playing on odd rounds.

11. **Opponent Portrait** An image of the opponent whom the player is up against.

12. **Opponent Cards** The opponent's cards are hidden from the player.

Figure 3.1 shows a starting state of the game with the left board (value of 29) being larger than the right board (value of 25). However, the player's comparator sign (”¡”) requires the right board to be larger than the left board. The player therefore plays first (on odd-numbered rounds) due to this disadvantage.

Game elements with which the player may interact are outlined in green. Figure 3.1 illustrates that the player can only play operator cards since they are the only cards with green outlines. A card can be played by dragging the card onto one of the game boards. Figure 3.2 demonstrates that the player played an operator card on the left board.

Figure 3.2: Playing a card on the left board

The result of the player's move is shown in Figure 3.3. Notice that the played operator card is removed from the player's hand and that the operator encapsulated all the previous items of the game board inside a bracket. Furthermore, Figure 3.3 emphasizes that the player cannot end his turn after the operator card has been played. The player's inability to end his turn forces the player to complete his turn by playing his number card on the left board where the operator has been played. A game participant may only play one operator card together with one number card on one of the board sides during his turn. The player therefore does not have any moves left to make in this turn and chooses to end his turn as shown in Figure 3.4.

Figure 3.3: Effect of card played

Figure 3.4: Opponent's turn

It can be seen from Figure 3.4 that it is the opponent's turn since there is a red outline around the opponent. The opponent has two extra cards in his hand since he had to draw a card from the red and blue decks when his turn started. Also, the comparator sign has changed to indicate the objective of the opponent. It is now clear that the player and opponent have conflicting objectives and that a beneficial move for one is a detrimental move for the other. The opponent chose not to make any game moves and ended his turn.

Figure 3.5 takes place on the third game round. The player drew another plus operator card and a new question card. Surprisingly, there are red outlines [2] around the operator cards of the player. The red outlines indicate that the player cannot play an operator card since there is no complementary number card to play after the operator card.

Additionally, the game has been designed so that number cards cannot be drawn from decks at the start of a round. The only way to obtain a number card is by correctly answering a question card. The outcome of playing a question card is given in Figure 3.5.

Figure 3.5: Playing a number card

A player is presented with a few questions after playing the question card, as shown in Figure 3.6. The questions are randomly generated and depend on the type of question card that has been played. The different types of question cards can be found in Section 3.4.4. The player should now select one of the two questions to solve. An interesting note to make is that the questions vary in difficulty and answers. Therefore, a player's choice of question is influenced by the difficulty of the question and the desirability of the question's answer.

[2] The use of red outlines might be criticized when considering game accessibility since ≈8% of men experience colour blindness, with red-green being the most common type of colour blindness [64]. However, the red-green combination was still chosen since it makes intuitively sense for the majority of players that green is good and red is bad. Future efforts can be made to improve the game's colour accessibility.

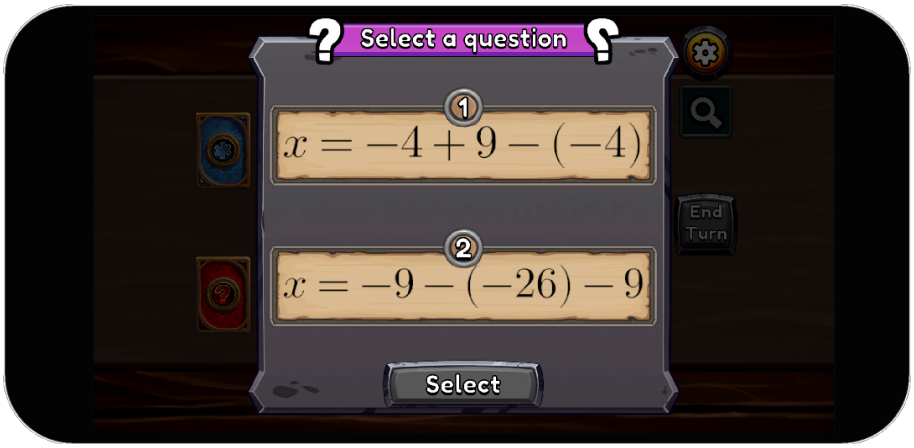


Figure 3.6: Options available when playing a question card

Figure 3.7 illustrates that the player chose to solve the second question. The player now has the opportunity to input his answer inside the text box. A correct answer will deposit a number card equal to the answer into the player's hand, while an incorrect answer will leave the player with no reward. Figure 3.8 shows that the player correctly answered that $x = 8$ and received a number card whose value is equal to the answer.

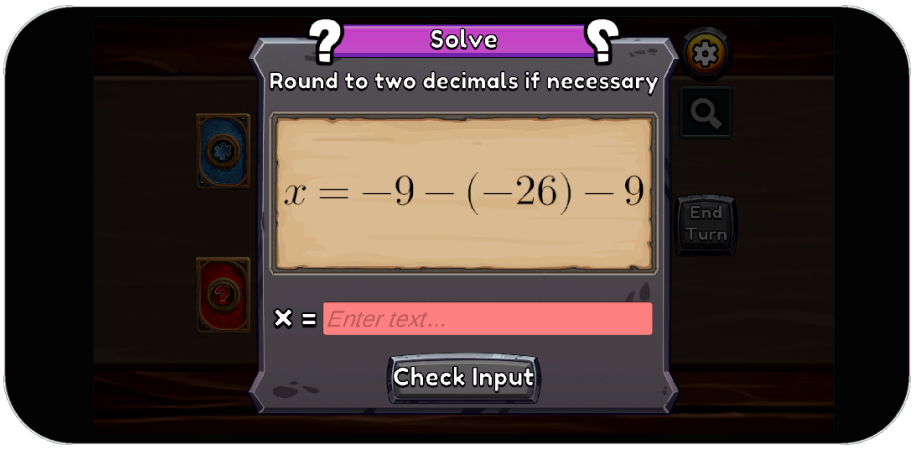


Figure 3.7: Question option selected

Figure 3.8 reveals that the player is finally able to play an operator card due to also having a number card in hand. The game loop now repeats itself where the player plays his cards on the board, the opponent responds by playing their cards, and the player needs to solve question cards in order to obtain number cards.

Figure 3.8: Correctly answered question card

Fast forwarding a few rounds presents the game state found in Figure 3.9. It can be seen in Figure 3.9 that the opponent has just finished his turn on the final game round (round 6) and a countdown commences to indicate that the game is about to end. The left board has a calculated value equal to 25 while the right board's calculated value is equal to 23. Therefore, the player loses since the player's objective was to make the right board larger than the left board.

Figure 3.9: Game over countdown

Finally, a game over screen (Figure 3.10) is presented to the player. The game over screen clearly conveys to the player that the game has been lost. The player obtained a score of zero since he didn't satisfy the objective of the game while the opponent received a score of two since the opponent's comparator sign is satisfied and the difference between the left board (value of 25) and the right board (value of 23) is equal to two ($25 - 23 = 2$). The player also received a few rewards for completing the game. Details pertaining to the rewards given are discussed in Section 3.6.3.

Figure 3.10: Game over summary

3.3.2 Gameplay with variables

Algebra is incorporated into the game with the introduction of variable cards. Variable cards are synonymous with number cards and can usually be played in all circumstances where number cards could have been played. Figure 3.11 illustrates a game where variable cards have been played on the game boards. It should be noted that the game contains only single variables, meaning that x is the only symbolic variable card in the game.

Figure 3.11: Example of a game with variables

The alert reader might start to wonder how a winner can be determined with variables on the game boards, since the variables are unknown. Introducing variable cards leads to a minor alteration of the original game rules by having a battle round before the game ends. The battle round provides game participants with the opportunity of substituting their own values for x. Players always draw one question card at the start of the battle round. An example of the battle round is given in Figure 3.12. It can be seen in Figure 3.12 that the player and the opponent can substitute different values for x. The player's substitution happens within the blue section (left-hand side) while the opponent's substitution occurs in the red section (right-hand side).

Figure 3.12: Example of the battle round

Figure 3.13 portrays a battle round where both the player and the opponent have substituted values for x. The score of a game participant is calculated in the same manner as with a normal game, meaning that a contradicting inequality sign yields a score of zero while a satisfied inequality sign provides a score that is equal to the difference of the inequality. A game participant without a substitution for x also receives a score of zero. A game is drawn when the scores of both participants are equal or zero. The player wins the game in the case of Figure 3.13, since the player obtained a much higher score than the opponent.

Figure 3.13: Substituted battle round

3.4 Cards in the game

An exhibition on the game's card collection can be found in this section. This section consists of four subsections that represent the broad categories to which the game's cards are classified. The game's card categories are defined as operator cards, mathematical object cards, function cards, and question cards.

Typical characteristics of a card are shown in Figure 3.14. The type and type colour of the card categorize the card while the card symbol conveys the meaning/value of the card. The card deck symbol informs the player from which deck the card comes. There are three decks (blue, green, and red) in the game. The distribution of cards in each deck depends on the level that the player plays. More information on the card distributions of various levels and the number of cards drawn from a deck can be found in Section 3.6.2

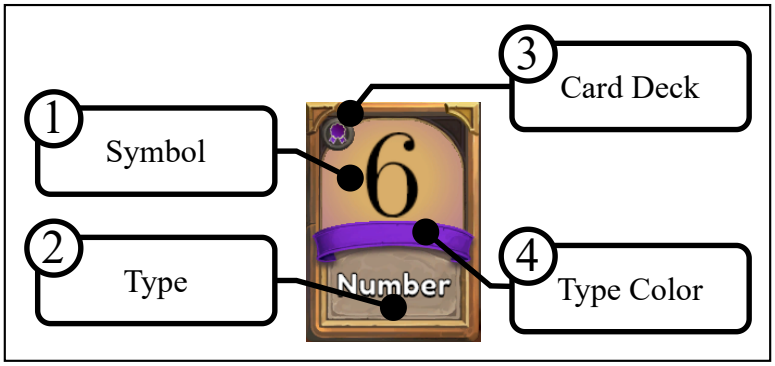


Figure 3.14: Characteristics of a card

A final note before commencing to the cards of the game. Much effort has been invested to ensure that the game's cards follow lenticular design principles. Lenticular design is a term originating from *Magic the Gathering* where the cards of the game are viewed through different lenses by players of different skill levels [65]. Novice players will often find the cards simplistic and easy to understand, while experienced players will use the hidden complexity of the cards [65].

For example, *Digit Draw* contains a division card, which would normally be used by a new player to decrease a board side. However, a division card can also be used to increase a board side by dividing with a number card that has an absolute value smaller than one. The division card further prohibits a game participant from playing zero in combination with the division card, since it causes an invalid division by zero. The reader is encouraged to ponder on other lenticular designs incorporated into the game while discovering the cards that the game has to offer.

3.4.1 Operator cards

Operator cards can only be played once in a game round and need to be succeeded by a mathematical object card. Operator cards always take the entire old game board expression as the left argument and the succeeding mathematical object card as the right argument. The game performs an internal check to ensure that a participant has a valid mathematical object card at hand before allowing the game participant to play the operator card. Moreover, operator cards are always drawn from the blue deck. The four operator cards of the game can be found in the table below.

Table 3.1: List of operator cards

Name	Symbol	Description
Plus		Performs the addition operation.
Minus		Performs the subtraction operation.
Multiply		Performs the multiplication operation.
Divide		Performs the division operation. Care is taken by the game to ensure that division by zero is not possible.

3.4.2 Mathematical object cards

There are two mathematical object card types in the game, namely number cards and variable cards. A variety of symbols exist for the mathematical object cards. For example, number cards can have symbols such as 6, 12, 3.14, or 0.2 while variable cards might have symbols like x^2 or $5 - x$. Gameplay possibilities for mathematical object cards include playing them after an operator card on the game board, playing them with a function card (Section 3.4.3), or using them inside the forge (Section 3.5). The only instance where a variable card cannot be played in place of a number card is in the substitution of the battle round (see Section 3.3.2). Table 3.2 lists the mathematical objects.

Table 3.2: List of mathematical object cards

Name	Symbol	Description
Number		Number cards are the only card type that cannot be drawn from a deck. One random number card is given to each game participant at the start of a game. The only way of obtaining additional number cards is by correctly answering question cards (see Section 3.4.4). The game typically provides number cards with values constrained between -10 and 10. Players are able to alter the number cards to have values beyond these ranges when using the forge or function cards. Constraining the number cards aids in balancing the game and improving the RL agent's performance (see Section 4.2.1).
Variable		Variable cards are drawn from the green deck. The 'x' symbol is the only variable card that can be drawn in the game. Players are capable of constructing additional variable cards by using the forge or function cards. Variable cards are introduced to the player only at later game levels (see Section 3.6.2), since they increase the game's difficulty by making the game more abstract.

3.4.3 Function cards

Function cards are unique in the sense that they are the only card type that can modify the mathematical object played on them. It is important to note that more than one function card can be played in the same round. Also, function cards are drawn from the green deck. Figure 3.15 shows a typical player hand with a function card.

Figure 3.16 illustrates the effect of playing a function card. The played function card remains on the game board until the player chooses a mathematical object card to serve as input to the function card. Figure 3.16 also shows that both the number card as well as the variable card in the player's hand are available for substitution since they have green outlines around them. A mathematical object card is substituted into the function card by dragging the mathematical object card on top of the function card.

Figure 3.15: Function card in hand

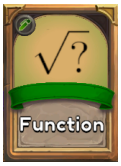
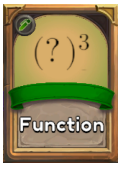
Figure 3.16: Function card is played on the board

Figure 3.17 shows the result of substituting the variable card of Figure 3.16 into the function card. The original square function card as well as the x variable card have been consumed and are replaced with an x^2 variable card.

It is interesting to know that the function cards played by the opponent are revealed to the player during gameplay. The opponent briefly shows the function card and mathematical object card, plays them together, shows the resultant card, and finally returns the resultant card face down to the opponent's hand. This demonstration from the opponent serves as a form of imitation learning to the player, since the player can observe the function play tactics of the opponent. A list of the game's function cards is given in Table 3.3.

Figure 3.17: Result of function card played

Table 3.3: List of function cards

Name	Symbol	Description
Negate		The negate function produces a resultant card with a sign opposite to that of the input card. For example, positive inputs produces negative outputs while negative inputs produces positive outputs.
Square		Square function cards produces the square of the input card. The square function card has the side effect of always producing positive outputs, irrespective of the input card's sign.
Square root		The square root function card is the only function card with a limited domain of inputs. The square root function card will only accept positive inputs and reject any negative input. Another side effect of this card is its ability to block negative numbers from being substituted in the battle round since there is a $\sqrt{x}$ present on one of the board sides. The same is true for positive numbers when there is a $\sqrt{-x}$ on one of the board sides.
Cube		The cube function card is very mighty since it has the same effect as playing two multiply cards. The output of the cube function preserves the sign of the input.

3.4.4 Question cards

Section 3.3.1 showed that playing a question card generates questions for the player to answer. Correctly answering the generated question rewards the player with a number card whose value is equal to the answer of the question. A player receives no reward from the opponent playing a question card and vice versa. The AI opponent is hard-coded to always correctly answer question cards.

Incorrect answers yield no reward to the player. Players instead receive feedback indicating what the correct answer is. An example of the feedback given for incorrect answers is shown in Figure 3.18.

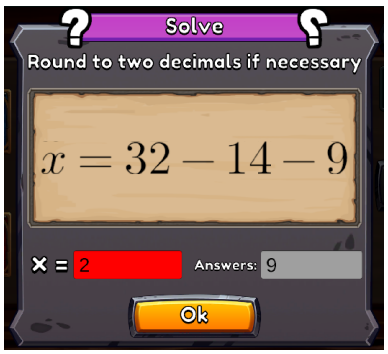
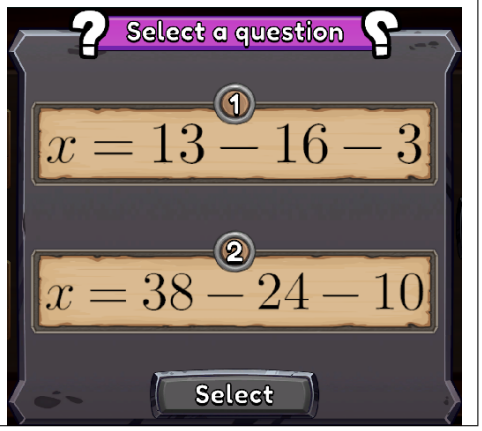
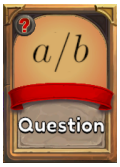
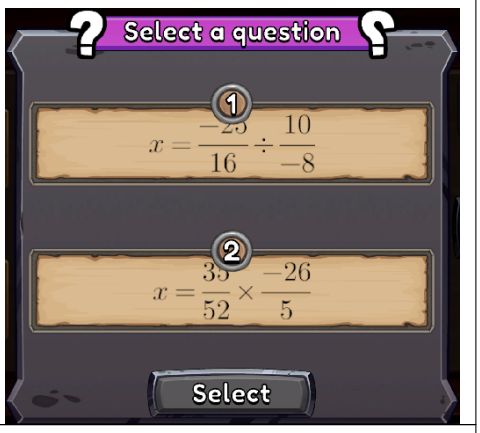
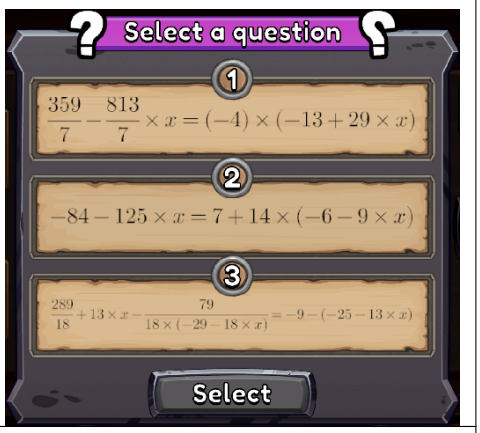
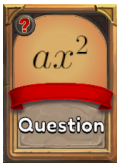
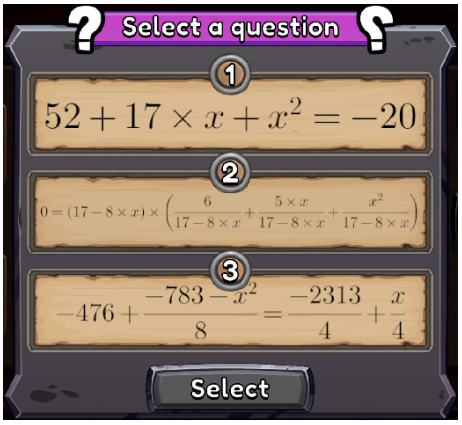


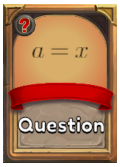
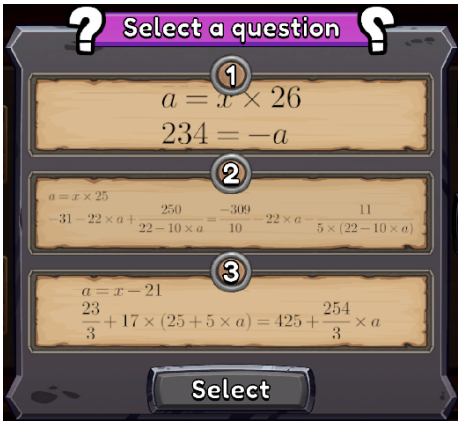
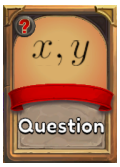
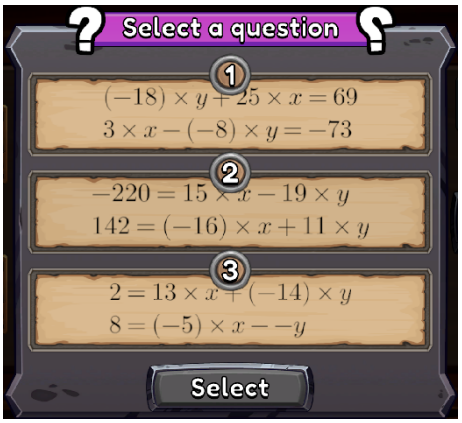
Figure 3.18: Feedback for incorrect answers

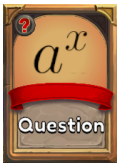
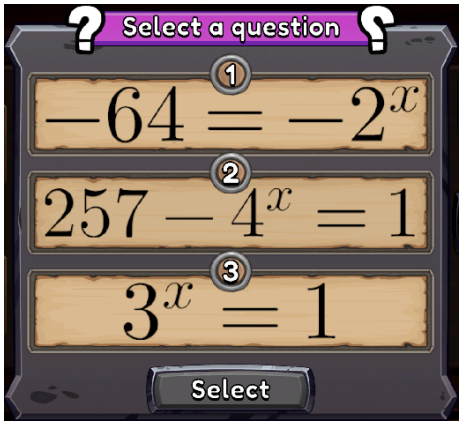
Question cards are drawn from the red deck. The different types of question cards together with their unique features are expressed in Table 3.4. The technical reader might find Appendix A intriguing, since it describes the algorithms used to generate the different types of questions as well as the algorithm used to validate the player's answers.

Table 3.4: List of question cards

Name	Symbol	Description	Example questions
Number		Number questions require adding or subtracting three numbers. Answers of number questions are integers between -10 and 10. Number questions have two generated questions to choose from.	

Fraction		Fraction questions necessitates the multiplication or division of two fractions. Answers of fraction questions are mostly rational numbers. Players are able to express their answers in decimal form (e.g. 0.24) or fractional form (e.g. 1/4). Fraction questions have two generated questions to choose from.	
Linear		The linear question card generates linear equation questions. The linear question card is the first algebra question card introduced to players. A linear equation question has a single integer answer that lies between -10 and 10. Players are able to choose from three generated question options when playing a linear question card.	
Quadratic		Quadratic question cards presents themselves as quadratic equations. Quadratic questions therefore represent a single equation has two possible answers. The player can enter only one of the answers in order to obtain one number card for the answer provided (if the answer is correct). Both answers are shown to the player when the player answered incorrectly. The answers of quadratic questions are integers that lie between -10 and 10. Therefore, the quadratic equations are always solvable without needing the quadratic formula. Players are able to choose from three generated question options when playing a quadratic question card.	

Substitution		Substitution questions consists of two variables and two equations. Players should therefore substitute one equation into the other in order to solve the variables. However, the question only seeks the value of the x variable and not the a variable. Subsequently, substitution questions are designed to have one integer answer that lies between -10 and 0. Players are able to choose from three generated question options when playing a substitution question card.	
Simultaneous		Simultaneous questions are very similar to substitution questions. However, simultaneous questions require the player to provide the solution to both variables. Therefore, players are able to obtain two number cards (one for each variable that is correctly answered) from one simultaneous question card. Simultaneous questions typically have integer answers that lie between -10 and 10. Players are able to choose from three generated question options when playing a simultaneous question card.	

Exponential		The player is required to solve an exponential equation when playing an exponential question card. The answer of an exponential questions is an integer that can range between 0 and 10. This small answer range means that players can solve the exponential questions with trail and error or by using logarithms. Players are able to choose from three generated question options when playing an exponential question card.	

3.5 Auxiliary game mechanics

Figure 3.19 presents the last few mechanics found in the core gameplay. These mechanics are called auxiliary mechanics. The auxiliary mechanics are accessed by pressing their respective buttons shown in Figure 3.19. The auxiliary mechanics are introduced to players on later levels (see Section 3.6.2), meaning that their buttons are hidden to players on earlier levels. The use of these mechanics is not strictly necessary to win the game, but they have been introduced to improve the effectiveness of the serious game in teaching mathematics. The following subsections investigate the forge, graphing, and substitution mechanics in greater detail.

Figure 3.19: Auxiliary mechanics in the game

Forge mechanic

The forge mechanic allows players to build their own cards from existing cards. Figure 3.20 conveys the forge screen that opens after a player has pressed the forge button. Players are able to add two mathematical object cards (number of variable cards) together with one operator card to build one resultant card.

Figure 3.20: Empty forge

Figure 3.21 shows the appearance of an occupied forge. Notice that the player can see the resultant forge before committing the cards to the forge. Having a preview of the resultant forge means that players can experiment with a variety of card combinations before investing the cards into the forge. The hope is that the mixing and matching of cards helps the player to easily distinguish between concepts such as $x + x = 2x$ but $x \times x = x^2$. Also, Figure 3.21 confirms that it is possible to forge number and variable cards together.

Figure 3.21: Occupied forge

Figure 3.22: Crafted forge card

The resulting forged card is shown in Figure 3.22. It can be seen from Figure 3.22 that the cards used in the forge (Figure 3.21) are removed from the player's hand and that the forge result is added as a variable card to the player's hand. Any forge result that contains a variable will be returned as a variable card to the player's hand, while strictly numeric results will be returned as a number card to the player's hand. Players are also able to see the forge actions of the opponent (as with function cards in Section 3.4.3). Lastly, Figure 3.22 shows that the forge button still has a green outline after the forge. The green outline indicates that the forge can be used more than once in a round.

Graph mechanic

The graphing mechanic aids players in visualizing what is happening on the game board. Players may use the graph as often as they like without any penalties. Figure 3.23 presents the graphing mechanic.

Figure 3.23 shows that the game's board sides are drawn by the graphing mechanic in different colours. The objective of the player is shown on the bottom left-hand corner (the left board should be smaller than the right board) while the graph's domain can be adjusted in the right-hand corner.

It is hoped that the graph forces the player to realize the significance of the domain and range of a graph. Players can only zoom in and out of the graph by changing the domain controls. Consequently, players will need to focus on their desired domain and range when trying to zoom.

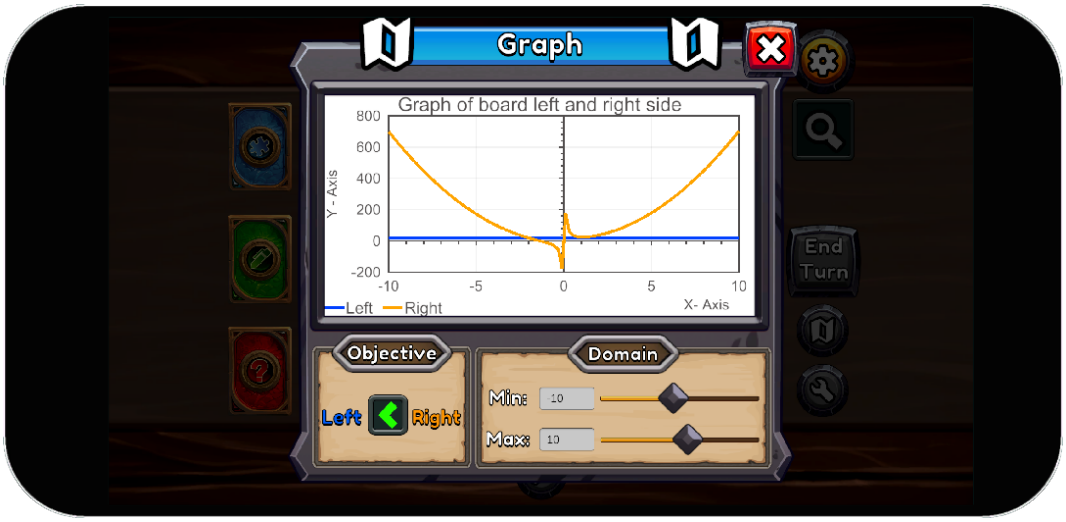


Figure 3.23: Graph mechanic

Closer inspection of Figure 3.23 reveals some strange behaviour of the orange trace near $x = 0$. The sudden change in the graph's values near $x = 0$ is due to a discontinuity that occurs at $x = 0$ for the figure's function. Unfortunately, the game's graphing feature is unable to account for the discontinuity and instead joins the closest defined samples together.

Simplify mechanic

The simplify mechanic allows players to rewrite a game board side into a simpler form. Using the simplify mechanic rewards players with game boards that are simpler to understand and increases the animal rescue chance of players. The animal rescue chance is part of a progression mechanic discussed in Section 3.6.3.

An example of the simplifying mechanic is given in Figure 3.24, where the player wrongly tried to simplify the game board. The "new equation" refers to the equation suggested by the player, while the "original equation" refers to the old value of the game board. The bottom input section shows that the player receives no pet increase chance since the new equation is not equivalent. The middle bottom section shows that the player chose to rewrite the right game board together with the text values supplied by the user to represent the rewritten equation.

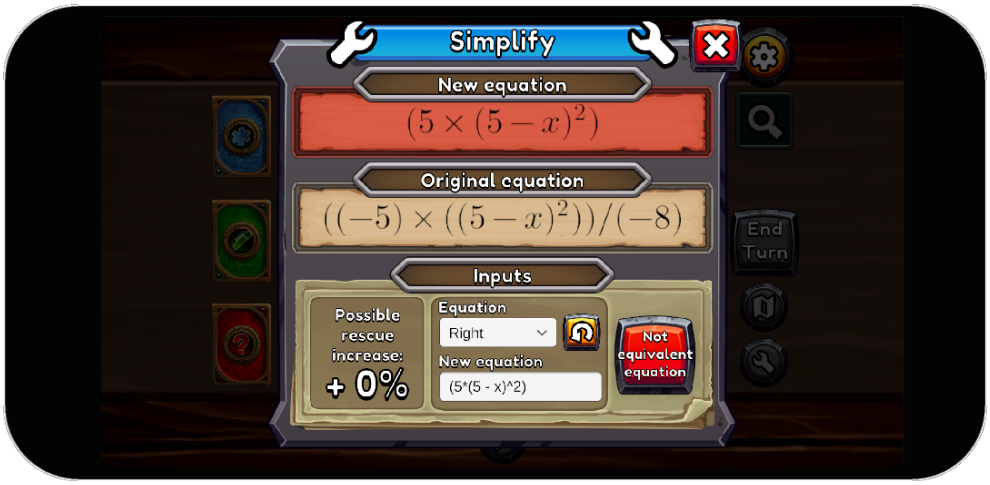


Figure 3.24: Simplify mechanic wrong

Figure 3.25 demonstrates a rewritten equation that is both equivalent and simpler than the original equation. The player is allowed to submit the rewritten equation and add an additional 3% to the pet rescue chance. The percentage increase in pet rescue chance is dependent on the degree to which the rewritten equation is simpler than the old equation. The player is prohibited from submitting a rewritten equation that is more complex than the original equation, even if they are equivalent. Lastly, players may only submit a rewritten equation once per game round. A submission restriction is done to encourage players to seek the maximum simplification/pet rescue chance for that round.

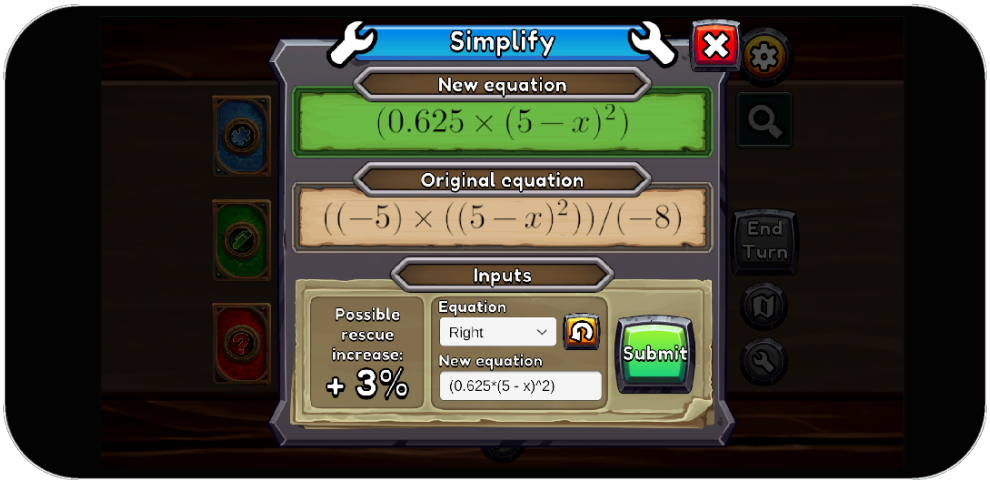


Figure 3.25: Simplify mechanic correct

3.6 Player retention mechanisms

Section 3.1 requires that desirable difficulties must be incorporated into the serious game to improve the game's learning effectiveness. However, E. Bjork and R. Bjork warn that desirable difficulties exhibit slow initial results and need to be spaced over a long time span in order to provide long-term benefits to a student [26]. It is therefore a priority to capture and retain a player's interest for the desirable difficulties to have a positive effect. Unfortunately, the dilemma is that players might feel manipulated by the serious game (tricked into doing something unpleasant), which can cause them to abandon the game entirely [5].

Therefore, effort has been invested to keep players' interest and to motivate them to continue playing. This section documents these efforts by introducing the game's theme, progression loop, in-game rewards, and using mobile notifications to remind a player to play the game.

3.6.1 Game theme

Stories in games create an emotional attachment with the player and provide the structure to organize gameplay moments [4]. Therefore, stories help to immerse the player in the game and aid in retaining the player. However, male and female players might not be interested in the same types of stories/games [4]. Male players often prefer competition and spatial puzzle games, while females enjoy nurturing and verbal puzzle games [4].

Hence, the game's theme takes place on an island where predatory animals keep smaller animals captive. The player's task is to battle the predatory animals in a series of card games in order to rescue the captured animals. Players collect water for the captured animals as they triumph over the predators. The water gathering theme attempts to interest males by initiating competition while females might enjoy nurturing the captured animals. The theme's integration into the game's progression loop is explored in the next subsection.

3.6.2 Game progression loop

The progression loop refers to the typical process that a player follows in order to progress in the game. The main source of inspiration for the game's progression loop is the ladder mechanic of the *Mortal Kombat II (1993)* video game (Figure 3.26). The ladder mechanic of *Mortal Kombat* requires a player to start at the bottom of a mountain and work their way up to the top by beating a series of opponents. The player must restart from the bottom if they fail against only one of the opponents. The difficulty of the game can be

adjusted by changing the length of the ladder since the chance of failing increases when the player has to duel more opponents before reaching the top. Figure 3.26 further shows that some opponents are anonymous and only identified once the player reaches them.

Figure 3.26: Ladder mechanic in *Mortal Kombat II (1993)*

Unfortunately, *Mortal Kombat*'s ladder mechanic could not just be directly copied and applied to the predator animals of the game. The main concern was that the progression block of resetting a player's progress upon failure might cause some players to become frustrated and quit the game (see player friction in Section 2.2.2). Another constraint is that each predator animal in the game must be implemented with a reinforcement learning agent. Section 5.1 shows that each RL agent has a size between 2.2 MB and 3.5 MB. Therefore, the game's size will become quite large if the ladder has plenty of opponents with different skill levels/ELO scores.

Instead, a variant of the ladder mechanic was implemented to address these shortcomings. The implemented progression mechanic is showcased in Figure 3.27. Figure 3.27 takes place on the first of twelve levels. Each level contains three opponents/predator animals for the player to defeat. The bottom left-hand corner accentuates the goals of each level, namely to defeat all three opponents and to collect enough water. The amount of water collected from defeating an opponent can be seen on the opponent's card. The collected water status is presented at the bottom of Figure 3.27. Also, Figure 3.27 illustrates that each level has a story (bottom centre) and a list of new unlocks (bottom right-hand corner). Some levels even have tutorial videos (bottom right-hand corner) that show players how new mechanics work. The trophies at the top of each opponent represent the player's high score for that level.

Figure 3.27: Game first level

This adjusted ladder mechanic allows players to fail without losing progression, thereby reducing player friction. Defeated opponents are marked as defeated and remain defeated even if the player loses subsequent games. Collected water is persistent and cannot be deducted.

Figure 3.27 shows that different amounts of water are collected from each opponent. The water supplied by the opponents differs due to the left-most opponents being the easiest opponent, while the right-most opponent is the hardest opponent. Beating every opponent only once does not yield enough water to proceed to the next level. Players will need to re-run the opponents until they collect enough water. Replaying the levels can be considered a passive difficulty adjustment mechanic, since skilled players are able to play only a few hard games before progressing to the next level, while struggling players receive additional practice by playing more games against easier opponents. The mechanic aids so that neither skilled nor struggling players feel like their progression is being blocked.

Figure 3.28a shows the effect of beating the opponent "Squeaky". Notice that 10% water has been added to the bottom progress bar and that the next opponent, "Major", has been revealed. The right-hand side of Figure 3.28a further illustrates that the next level is still locked since all the objectives are not yet completed. Finally, Figure 3.28b presents a completed level with all the opponents defeated at least once and 100% water collected. The arrow on the right-hand side of the screen can be used to advance to the next stage.

(a) First level progressed

(b) First level completed

Figure 3.28: Level progression

Figure 3.29: Another level of the game

A progressed game level is shown in Figure 3.29 to portray the similarities and differences between levels. An apparent difference between levels is that Figure 3.29 has a different background and set of predator animals when compared to Figure 3.28. Also, the story and level unlocks between levels differ, with Figure 3.29 not having a tutorial video. Other than that, levels remain largely the same and use the same progression loop. A summary of the game's level progression and unlocks is given in Table 3.5.

Table 3.5: Level unlocks and opponent water supply amounts

Level	Features unlocked	Left opponent	Middle opponent	Right opponent
1	• Number questions • Plus operator	+10%	+15%	+25%
2	• Minus operator	+20%	+30%	+50%
3	• Fraction questions • Multiply operator	+10%	+15%	+25%
4	• Forge mechanic • Linear questions	+20%	+30%	+50%
5	• Square function	+10%	+15%	+25%
6	• Square root function • Divide operator	+10%	+15%	+25%
7	• Graph mechanic • Negate function	+20%	+30%	+50%
8	• Variable cards	+10%	+15%	+25%
9	• Quadratic questions	+10%	+15%	+25%
10	• Simplify mechanic • Substitute questions	+10%	+15%	+25%
11	• Cubic function • Simultaneous questions	+10%	+15%	+25%
12	• Exponential questions	+10%	+15%	+25%

Table 3.5 conveys that the opponents of level 2, 4 and 6 provide more water than the opponents from other levels. The additional water supplied by these levels aims to increase the tempo of progression since these levels do not have as steep learning curve as other levels.

Variable cards and battle rounds are unfortunately only unlocked at level 8, since players should first be comfortable with the base mathematical concepts before commencing to the abstract algebra concepts. However, players already start exercising their algebra at level 4 when the linear question cards are introduced.

Auxiliary mechanics (Section 3.5) are unlocked throughout the game, with the forge mechanic unlocked at level 4, the graph mechanic unlocked at level 7, and the simplify mechanic unlocked at level 10. The simplify mechanic encourages players to play the final levels more often since it is the only way to drastically increase their animal rescue chance. The animal rescue mechanic will be discussed in the next section.

As a final note, Table 3.5 shows that the cards of the game are unlocked at different levels. The distributions of cards in a deck are fixed for a given level and players play with a deck that is provided to them. The card distributions at various levels can be found in Appendix B. The game design background (Section 2.2.2) argued in the luck and skill subsection that deck building can be added as a preparation phase to make a game more skilful. Deck building allows players to build their own card decks and possibly buy cards to add to their decks, similar to the store mechanic in *Mathy Fight* (Section 2.2.3). Even so, statistical consultation for the experiment of Chapter 6 advised against the use of deck building mechanics since it can cause players to experience drastically different games since they basically build their own levels. The difference in gameplay experiences could introduce more variance into the experiment and make it harder to draw conclusions.

3.6.3 Rewards

Section 2.1.2 mentions that the response strengthening pedagogy incorporates rewards to reinforce desirable behaviour. Similarly, rewards are also utilized to train the AI opponents to play the game (Chapter 4). Therefore, in-game rewards are given to players at the end of a game to provide response strengthening feedback. Figure 3.30 illustrates that the game over screen presents the reward summary at the end of a game.

Figure 3.30 shows that players are rewarded with experience (exp), water percentage, high scores and animal rescue percentage. Experience provides players with a cosmetic badge that indicates their experience level. The amount of experience gained is equal to the product of the current campaign level times the amount of water provided by the opponent in the case that a game is won. Lost games only provide a quarter of the experience that a victorious game would have given. Water percentages are only given to victorious games and are capped at 100% (as described in the previous subsection). Therefore, lost games do not provide any water percentages. High scores motivate players to improve their performance in each level and reward players with extra animal rescue percentages.

Figure 3.30: Rewards summarized at the end of a won game

Animal rescue percentages are used in conjunction with the fortune wheel (bottom right-hand corner of Figure 3.30)[3] to rescue animals. Players obtain rescue percentage from playing games (+5% for a win, +2% for a loss), achieving new high scores (+10%), or using the simplify auxiliary mechanic (Section 3.5). Players have the option to spin the fortune wheel only if they have won the game. Spinning the fortune wheel resets the player's rescue percentage back to 0% while giving the player the chance of rescuing an animal. Lost games or not spinning the fortune wheel preserve the player's rescue percentage. A successful fortune wheel spin is shown in Figure 3.31. The figure conveys that the player rescued a white chicken named "Persis".

Figure 3.31: In-game animal rescued

[3]There are two buttons at the bottom of the game over screen. The leftmost button returns the player to the level select menu while the rightmost button restarts the current level. This button placement was deliberately chosen to encourage players to automatically play another game. In fact, it worked so well in initial testing that players kept replaying the same level again and again, even if they could advance to the next level. The replay button is therefore hidden when a new opponent is available to challenge

Figure 3.32: Rescued animals in the main menu

Rescued animals appear in the main menu, as shown in Figure 3.32. The rescued animals wander around and occasionally speak to the player by means of a speech balloon. The speech balloons have a random chance of appearing and usually provide tips to the player, such as squaring a number will always yield positive results. A side effect of the rescued animals is that more speech bubbles appear when more animals are rescued. Therefore, the player's hint rate increases as more animals are rescued.

Figure 3.32 further shows on the bottom left-hand side that a catalogue of rescued animals can be accessed. The opened catalogue is presented in Figure 3.33, where it is seen that all animal types of the game are documented by the catalogue. Undiscovered animal types are hidden behind a silhouette similar to the ladder mechanic of *Mortal Kombat II*. However, players can see from the catalogue of Figure 3.33 that some animal types have a lower chance of being rescued and can only be found on certain levels. It is possible for players to rescue the same animal type more than once.

Figure 3.33: Animal rescue catalogue

3.6.4 Notifications

Push notifications are the final retention mechanism added to the game. Von Ahn mentions that they added push notifications to the *Duolingo* serious game since serious games have to compete with entertainment apps for a user's attention [66]. Consequently, *Duolingo* developed a variant of the multi-armed bandit algorithm[4] (called the recovering difference softmax algorithm) to choose and schedule personalized notifications to send to the users [67]. Surprisingly, the sophisticated notification algorithm of *Duolingo* boiled down to sending a notification 24 hours after the user last used the app [66]. The 24-hour delay makes sense since the user is most likely available for a game the same time that they were available in the past [66].

An example of the push notifications implemented by *Digit Draw* is shown in Figure 3.34. The push notifications implementation of *Digit Draw* is primitive and only sends non-personalized notifications after every 48 hours that the player has not played. The period has been doubled from those suggested by *Duolingo*, since a study participant might uninstall the game if the notifications become bothersome. The notifications are halted when the student has not opened the game for 6 days.

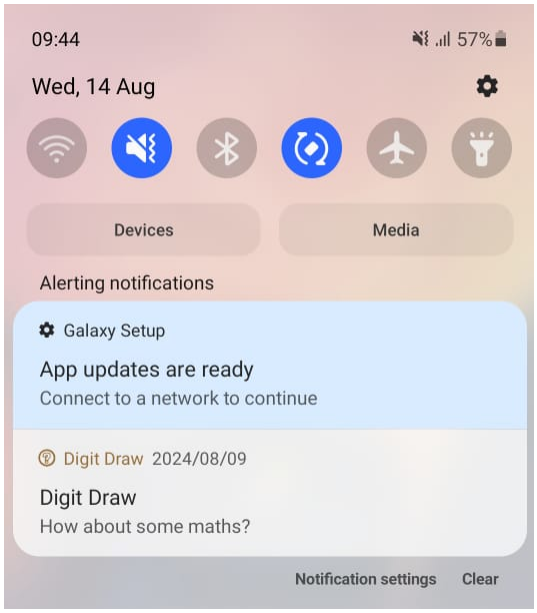


Figure 3.34: Game reminder notification

[4]Interestingly, multi-armed bandit algorithms is a special case of reinforcement learning algorithms that has only one state [6]. This example emphasizes the wide applicability of reinforcement learning in training AI

3.7 Reflection and summary of game's design

This chapter introduced the rules and design of this study's serious game, *Digit Draw*. Section 3.1 conveys the vision of *Digit Draw* as giving players a "feel for mathematics instead of just memorizing the rules" and to help them learn by constructing equations and discovering the rules themselves. This vision has materialized by developing *Digit Draw* as a card game where players attempt to satisfy their inequality sign by building mathematical expressions with their cards. *Digit Draw* is played against an AI opponent, making it a two-player zero-sum adversarial game.

Section 3.3 presented the rules and gameplay of the game while Section 3.4 showcased the variety of cards that can be found in the game. Auxiliary game mechanics are given in Section 3.5 to address deficiencies in the core gameplay while Section 3.6 presents various mechanisms employed to retain players' interest.

Unfortunately, there are a few game design considerations not discussed in this chapter for the sake of chapter brevity. Examples include the notion that longer game rounds reduce the luck of the game (since players can make more choices), but make the game more cumbersome to play. Another is that reducing the maximum amount of cards in a player's hand reduces the information horizon of the game (similar to hidden information) so that the player is less overwhelmed by the amount of choices that are available. Also, the notion of players explicitly pressing a button to signal the end of their turn helps the player to keep track of what is happening in the game. Every player-based action is intentional, meaning that a player's turn is never automatically passed. Parameters such as the maximum amount of cards in a player's hand being eight and the number of rounds in a game being six were chosen based on what felt right when playing the game. The interested reader might find that skimming over Chapter 2 after reading this chapter can aid in identifying some of these veiled design choices that players would not normally think about, but could influence the perception that they have about the game.

It is anticipated that some players can find the initial learning curve of the game difficult due to the disadvantages associated with card games (abstract and intricate rules). Therefore, guidelines from Section 2.2.2 have been followed to reduce the game's complexity. The guidelines include improving the user interface design, reducing the game's pace, and providing tutorials on how the game mechanics work.

The UI design was improved by adding green outlines to actions that a player can perform and prohibiting invalid actions. The game's pace was slowed by introducing variable cards later in the game (Section 3.6.2). Tutorial videos have also been made and embedded into the game to show players how new mechanics work, although interactive game tutorials might be superior at the expense of increased development time [4]. The core game rules were iterated upon throughout the design process to reduce the game's complexity.

Examples include the introduction of the battle round for variable cards, standardizing the player's turn order to follow those of conventional card games, and allowing players to skip their turns without being penalized.

The successfulness of *Digit Draw* in reaching Section 3.1's design requirements is given in Appendix C. The table conveys that *Digit Draw* satisfies most of the design requirements. There are still high school algebra concepts that are not implemented in the game. Additional features could be to add multiple choice question cards that ask the student to select an inverse function for a given function, or to choose a mathematical expression that is equivalent to one provided (proof of identity). Questions practicing the k-method, completion of squares, factorization, expansion of terms, long division and the quadratic formula can be beneficial. Also, exponential function cards (such as 2^x) are not added since the exponential function's output increases too rapidly (making the card too powerful), while trigonometric function cards ($\sin(x), \cos(x), \tan(x)$) are not implemented since grade 9 learners have not yet been exposed to trigonometry [15][16].

Lastly, it was interesting to see that an entertainment card game, known as *Balatro*, was released while *Digit Draw* was nearing the end of its implementation phase. *Balatro* also gained inspiration from *Poker*, where a player is dealt certain cards from a deck and it is up to the player to identify the cards combinations that allow for the highest score. *Balatro* received numerous coverage with overwhelmingly positive reviews [68][69]. The success of Balatro gave an indication that the design of *Digit Draw* can have player appeal and that there are players who are interested in these types of card games.

Chapter 4

RL agent design

This chapter presents the design of the study's RL agent. The agent is used to serve as an opponent and mediator for the two-player adversarial zero-sum card game developed in Chapter 3. It must be emphasized that the aim is to develop an agent that can serve as a mentor/example for the player to learn from. The player must gain a sense of accomplishment when beating the opponent. The aim is not to develop an unbeatable agent, as is done in the case studies of Section 2.3.4. The agent played a fundamental part in play-testing the game by detecting exploits and mechanics that did not fit well with the game.

It will be seen that the majority of the chapter covers the state and action representation instead of the implemented RL algorithms. The reason for covering the state and action representation in greater detail is due to the state, action and reward representation having potentially a much larger impact on the agent's performance compared to choosing between state-of-the-art RL algorithms or tuning hyperparameters [7]. In fact, the choice was made to rather make use of an RL library (Unity's ML-Agents) instead of natively implementing the algorithms. An RL library helps to mitigate the risk of algorithm complexities being incorrectly implemented and causing performance losses that aren't visually noticeable [7][61].

Therefore, this chapter commences with the usage of the ML-Agents RL framework to model the environment (Section 4.1), followed by Section 4.2 that gives the agent's state, action and reward designs. Section 4.3 discusses the two RL algorithms used in this study (PPO and SAC) to provide a background for interpreting the results of Chapter 5. Finally, Section 4.4 showcases notable difficulties encountered when implementing RL and Section 4.5 ends the chapter by providing a conclusion of the agent's design.

4.1 ML-Agents RL framework

The usage of the Unity game engine's ML-Agents framework is described in this section. As mentioned previously, the ML-Agents library was chosen over a native implementation to prevent implementation bugs from occurring and to ensure that the performance results and design process are more reproducible (uses standardized implementation) [7][61].

Figure 4.1 illustrates a high-level overview of the interface between the serious game and the ML-Agents framework. It should be noted that the purple-colored blocks of Figure 4.1 indicate functions that are performed by the ML-Agents framework.

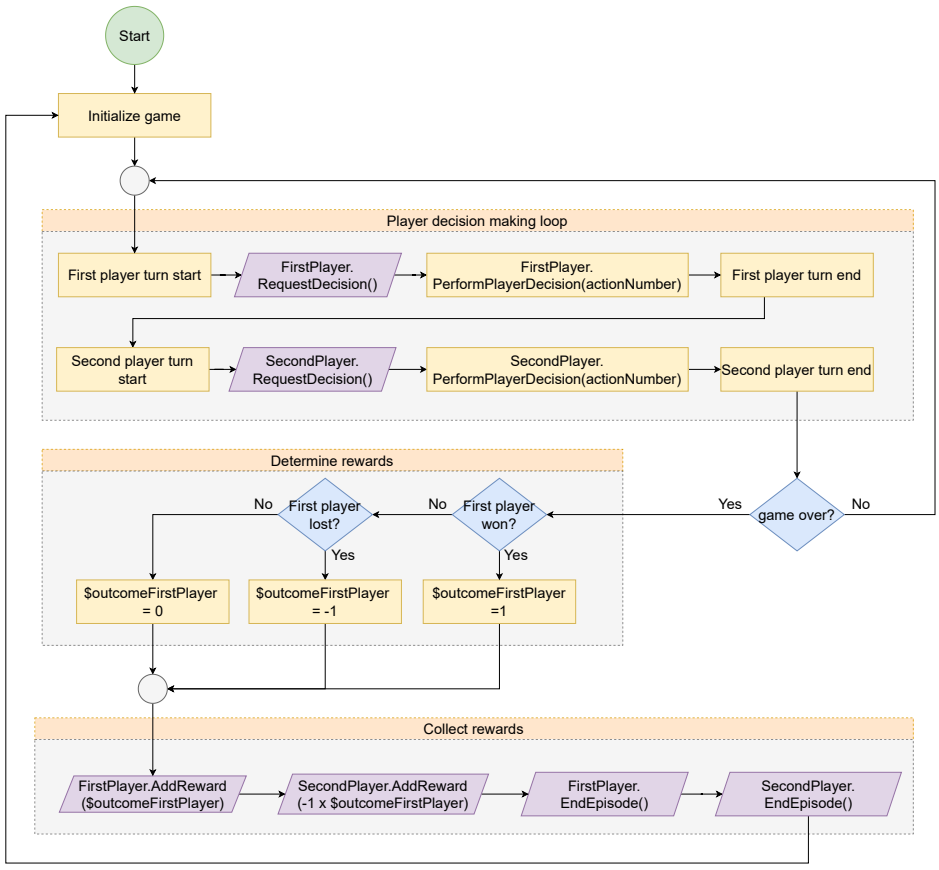


Figure 4.1: Framework of ML-Agents in this study

The general program flow of Figure 4.1 expresses that an episode begins with the game being initialized (starting cards are drawn for both sides), the players playing the game/making decisions until the game/episode ends, rewards being calculated for the episode that took place and then finally the rewards being distributed to the agents so that they can adjust their policy before commencing with the next episode. Subsequently, the purple instruction blocks indicate that the ML-Agents framework provides the interfaces for requesting a decision, dispensing rewards to an agent, and declaring the end of an episode.

The "RequestDecision" function block of Figure 4.1 can be investigated in greater detail by turning to Figure 4.2. It is shown in Figure 4.2 that a DecisionRequest() call consists of three sub-systems, namely obtaining the observations (gathering the states) with CollectObservations(sensor), writing an action mask (prevent certain actions from being taken) with WriteDiscreteActionMask(actionMask), and actually taking the action suggested by OnActionReceived(actions). The ML-Agents framework is only capable of masking discrete actions (cannot mask continuous actions) and requires that at least one action is unmasked.

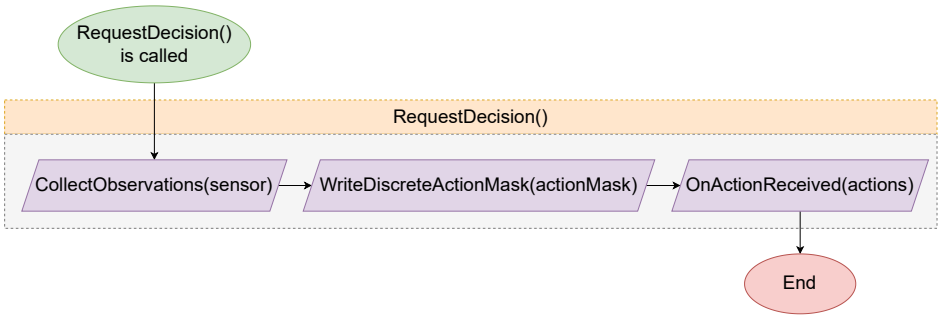


Figure 4.2: RequestDecision() overview

However, the simplicity suggested by Figure 4.2 can be somewhat misleading since it is not guaranteed that the order of the sub-systems will be CollectObservations(sensor) followed by WriteDiscreteActionMask(actionMask) and OnActionReceived(actions). The arbitrarily execution order is due to the ML-Agents library being geared more towards real-time video games that constantly provide states and actions on each game frame with little differences between subsequent frames. Instead, turn-based games (such as *Digit Draw*) must wait for the next *fixed update* of the Unity game engine before proceeding to the subsequent sub-system of Figure 4.2. Synchronizing the subsystems of fig. 4.2 with the fixed updates of the Unity game engine ensures that the sub-systems of Figure 4.2 are executed in order.

The software version of ML-Agents and its dependencies were another source of agony since incompatible dependency versions would lead to unstable training. It is therefore important that the software versions described in Appendix J are used to prevent instabilities from creeping into the training process. These difficulties give reason to first make smaller demo games before applying ML-Agents to large complex games. For example, small demo games that were made to practice the framework include the Google Chrome dinosaur game (single player game), Pong (real-time two player adversarial game) and tic-tac-toe (turn-based two player adversarial game).

Lastly, it is noteworthy to mention that the agent trains with online data by playing against itself. The trained agent's policy is stored and executed offline. Using an offline agent model means that the human player plays against a static offline RL agent and

that no further agent training occurs in the production version of the game. Instead, different offline models are stored and used to simulate a variety of opponent difficulties (see Chapter 5).

4.2 State, action and reward design

This section describes the state, action and reward representation of the serious game. The section consists of four subsections, with the first subsection (Section 4.2.1) describing how the state of a mathematical function can be represented, and the second subsection (Section 4.2.2) showcasing the complete state composition for the whole game. Section 4.2.3 presents the complete action representation of the game, and Section 4.2.4 shows the reward attributions of the game. Finally, Section 4.2.5 mentions how the state search space is reduced by having the agent only train on levels/scenarios that are in the game.

4.2.1 State representation of mathematical functions

The primary hurdle in this study was finding a suitable state representation for mathematical functions of the game's cards and game board. The main trouble was obtaining a state representation that is versatile enough to account for players being able to build their own general mathematical functions (see Section 3.5).

Figure 4.3 shows some characteristics of a scalar function. A scalar function has two primary features, namely the input (usually denoted as x) and the output (usually denoted as y). However, other latent features can also exist within a function such as the function's seasonality/periodicity or trends of increasing/decreasing. A function's state representation is further complicated by being defined on a continuous/analogue interval, while computers commonly make use of discrete/digital data.

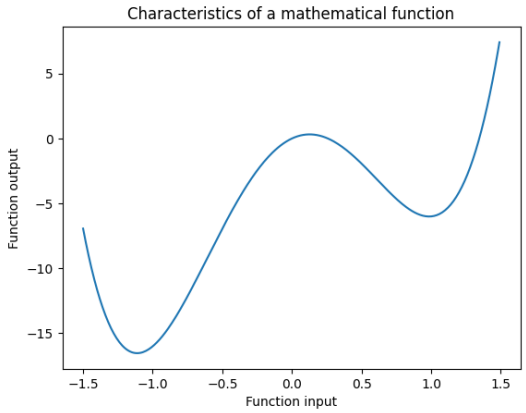


Figure 4.3: Example of an ordinary function

Methods that come to mind for representing the function could be to use one-hot encoding for each function (can take the hash of function samples) or perhaps even the string of the function itself (such as "$x^2 + 3$"). Unfortunately, these representations are not very scalable due to the curse of dimensionality (see Section 2.3.1) of basically having infinite one hot function representations or the difficulty of comparing strings to check if they are the same (must know that the string "$x^2 - 5x^2 + 3$" is the same as the string "$5 - 2 - 4x^2$").

Instead, the simplest solution is to directly sample a function and use the samples of the function for the state representation. The function sampling state representation method seems the most appropriate since it has increased potential of being generalizable. For instance, the agent can learn that adding a positive number card will increase the affected sample values or that subtracting a positive number card will decrease the affected sample values.

Gathering the samples proves tricky since the function's output values can have an infinite range (ANNs like having their input normalized between -1 and 1 to improve generalization [70]), and the choice of sampling intervals for the function's input values is function dependent. The following two subsections address these inconveniences and suggest solutions to them.

Restricting the output range of sampled values with clamped dynamic range compression

It is required to map the sampled output values of a mathematical function, which has an infinite range, to a range between -1 and 1 in order to improve the performance/generalization of the ANN [70]. The process of mapping values from a large range to a smaller range is widely done in the audio industry with a process known as *dynamic range compression* (DRC) [71].

Unfortunately, performing DRC is not as simple as dividing the samples by their maximum possible values since the samples are stored as floats (loses numerical precision on smaller numbers when dividing by a large number) and the maximum possible value for a sample is infinity (any real number divided by infinity is zero). Instead, thresholds are set to vary the compression's gain ratios [71]. Figure 4.4 shows that the gain adjustment can be sudden between thresholds (*hard knee*) or change evenly (*soft knee*). Soft knee DRC is preferred in the audio industry to prevent audible distortion in the output signal [71]. However, this study uses hard knee DRC due to its implementation simplicity and reduced computational requirements [71].

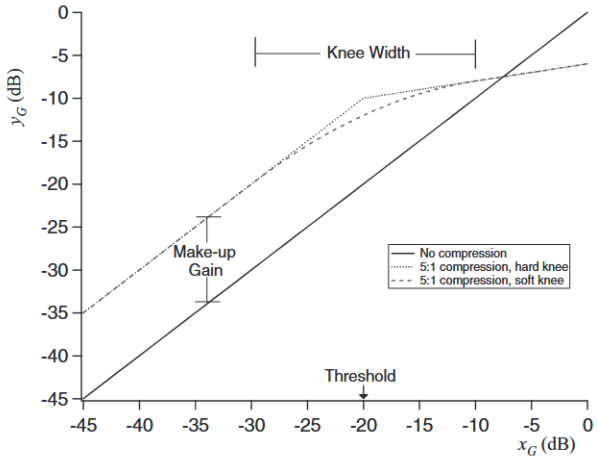


Figure 4.4: Example of hard knee and soft knee DRC [71]

Of course, DRC only aids with numerical precision, but does not aid in converting infinite values to a value between -1 and 1. *Clamping* is therefore added by passing the DRC output through the hyperbolic tangent function ($tanh(x)$) to limit the clamped DRC output values to be between -1 and 1, as shown in Figure 4.5.

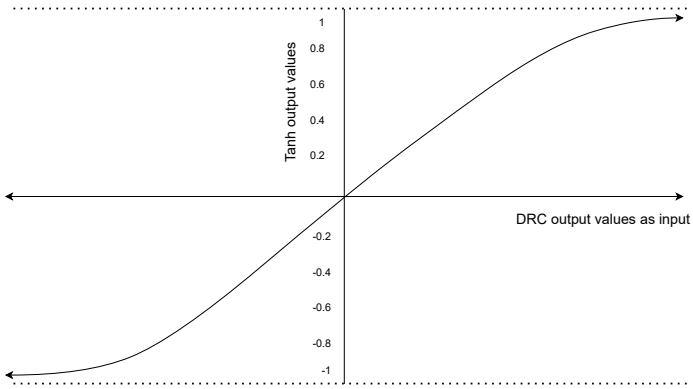


Figure 4.5: Clamped DRC

In a sense, this allows one to allot a percentage of the mathematical function output range to a certain clamped DRC output range. For instance, 10% of the clamped DRC output values (values between 0 and 0.2) can be attributed to mathematical function output values that are between 0 and 1, while only 2.5% (values between 0.95 and 1) may be given to mathematical function values between 1000 and ∞. Being able to specify the regions where the input values should be clamped is an important property since it enables us to give the agent higher resolution (allot more of the clamped DRC output percentage) to values between -1 and 1 (the agent requires the additional resolution to know that multiplying with an absolute number smaller than one is the same as dividing) while a small percentage of clamped DRC output values should be given to input values

larger than 1000 (it is less likely that a player will be able to forge cards that fall within this output resolution, meaning that this output resolution is better spent on a different range).

Table 4.1 shows the absolute clamped DRC ranges used in this study. For example, if a function $f(x) = x^2$ is sampled at $x = 2$ (yielding $y = f(2) = 4$), then the clamped DRC value will be between 0.1 and 0.35 according to Table 4.1. Negative clamped DRC is symmetric with the positive clamped DRC, meaning that a clamped DRC value of -4 will be between -0.35 and -0.1 correspondingly. The clamped DRC allocation percentages of Table 4.1 were chosen arbitrarily and can be treated as a hyperparameter if further performance enhancements are desired.

Table 4.1: Clamped DRC ranges used for agent in this study

Absolute mathematical function value	Absolute clamped DRC value	Percentage of DRC range
$0 \leq \|x\| \leq 1$	$0 \leq \|y\| \leq 0.1$	10%
$1 \leq \|x\| \leq 10$	$0.1 \leq \|y\| \leq 0.35$	25%
$10 \leq \|x\| \leq 100$	$0.35 \leq \|y\| \leq 0.65$	30%
$100 \leq \|x\| \leq 1000$	$0.65 \leq \|y\| \leq 0.9$	25%
$1000 \leq \|x\| < \infty$	$0.9 \leq \|y\| < 1$	10%

Choosing which mathematical function input values to sample

This section investigates the second dilemma of sampling a function, which is to choose what function input values to sample and the number of values to sample. Usually, the number of values to sample depends on the highest frequency component of a periodic signal so that the Nyquist sampling theorem is satisfied [72]. However, the game's cards in Section 3.4 show that the game contains no periodic functions and only consists of polynomials.

The fact that the game primarily relies on polynomials means that the number of samples for the function should rather be based on the degree of the polynomial, as is done with polynomial interpolation. For instance, an n^{th}-degree polynomial requires $n + 1$ samples in order to be accurately fit to the data [73]. It is difficult to determine the largest degree polynomial that can be built in the game, since a player can in an unlikely scenario forge a 36th degree polynomial by multiplying three x variable cards, play two quadratic functions on the result and cubing the final answer. Therefore, it has been chosen to sample 41 values per function since it yields reasonable training performance and is expected to fit the majority of functions encountered in the game.

Choosing which input values should be sampled can be done in a variety of ways, with a naive implementation being the uniform sampler of Figure 4.6. The uniform sampler has a fixed sample rate, meaning that samples are taken in uniform intervals. The uniform sampler is favourable for its simplistic implementation and the ability to compare the sampled values of different sampled functions without needing to know the intervals at which they are sampled (as long as the samples are taken at the same function inputs).

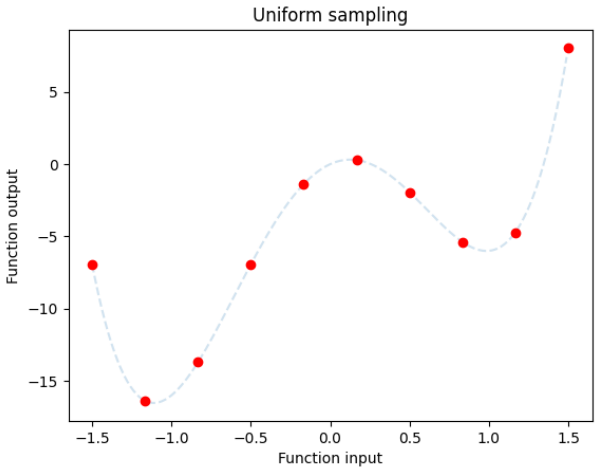


Figure 4.6: Uniform sampling illustration

However, a shortcoming of the uniform sampler is that interesting features of a function do not necessarily occur at uniform intervals. The sampler might miss these features if sampling too slow, or store irrelevant samples when sampling too fast. An alternative is to sample at variable non-uniform intervals so that samples are strategically taken where interesting characteristics of a function occur. Figure 4.7 shows an example of a variable non-uniform sampler where samples are specifically taken at the local maxima and minima of the function.

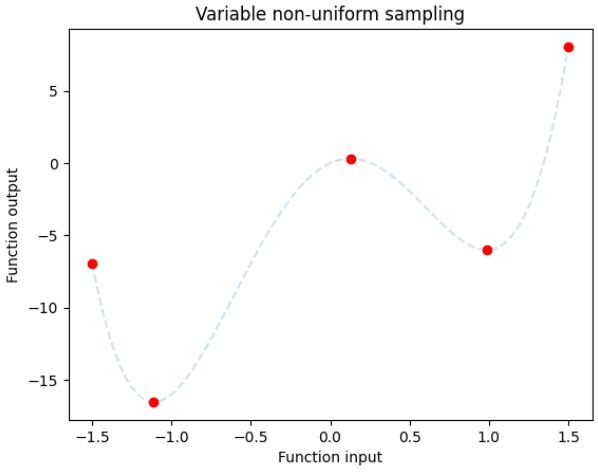


Figure 4.7: Variable non-uniform sampling illustration

Unfortunately, the variable non-uniform sampler requires double the number of features of the uniform sampler since the variable non-uniform also needs the intervals where samples have been taken in order to compare the values of two sampled functions. Needing to also know the intervals where samples have been taken means that an agent employing a variable non-uniform sampler will only have half the number of samples to base its decision on when compared to a uniform sampler (if the same number of input features are used for both samplers). Also, an informal experiment showed that a PPO RL agent was unable to make any meaningful learning progress (ELO score remained the same) when using a variable non-uniform sampler that sampled local maxima and minima of a function.

Therefore, a mixture between uniform sampling and variable non-uniform sampling yields the constant non-uniform sampler. The constant non-uniform sampler always samples at the same intervals, but the distance between sample intervals can be different (non-uniform). Figure 4.8 shows that the same number of samples can be taken with the constant non-uniform sampler as with the uniform sampler. The number of samples required by the non-uniform sampler and the uniform sampler is the same due to the constant non-uniform sampler being able to compare sampled functions with each other without knowing where they were sampled (as long as they were sampled at the same locations).

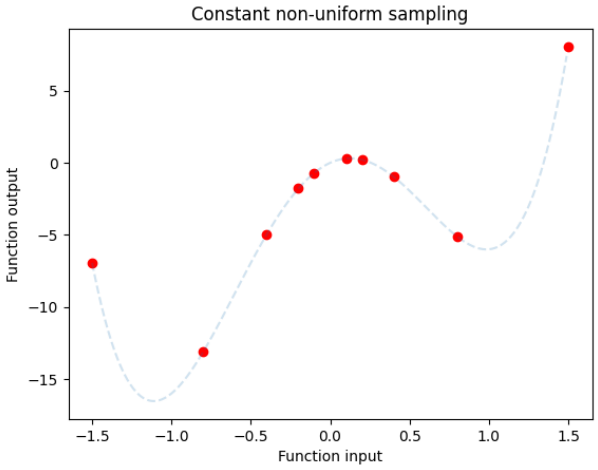


Figure 4.8: Constant non-uniform sampling illustration

The intervals of a constant non-uniform sampler can be chosen so that more samples are taken at smaller input values (such as $-1 < x < 1$, since multiplication turning into division in this interval) and fewer samples are taken at larger inputs (such as $|x| > 100$, since polynomial functions are many times monotonic at larger absolute input values). For this study's agent, all mathematical functions are sampled at the 41 values of Equation 4.1.

The sample interval values were arbitrarily chosen so that more samples are taken between smaller absolute function inputs and less within large absolute function inputs.

$$\begin{aligned} functionInputSamples \in \{&-1000.0, -800.0, -600.0, -400.0, -200.0, -100.0, -80.0, -60.0, \\ &-40.0, -20.0, -10.0, -8.0, -6.0, -4.0, -2.0, -1.0, -0.8, -0.6, \\ &-0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1.0, 2.0, 4.0, 6.0, 8.0, 10.0, 20.0, \\ &40.0, 60.0, 80.0, 100.0, 200.0, 400.0, 600.0, 800.0, 1000.0\} \end{aligned} \quad (4.1)$$

4.2.2 Total game state representation

This section presents the total state representation used by the agent for the serious game. The mathematical function representation, discussed in Section 4.2.1, is used for all function representations of this section. Figure 4.9 presents an overview of the total state representation. From Figure 4.9 it can be seen that the agent has a total of 439 features that are divided into several categories, such as the *round of game*, *order of turn*, *board comparator sign*, *operator deck draw chance*, *question deck draw chance*, *variable deck draw chance*, *equation panels*, and *sorted card in hand*.

The *round of game* is a single float value that indicates the progress, as a percentage, till the battle round/game over (0.0 is a new game and 1.0 is a finished game). *Order of turn* indicates whether the agent is the first or the second player (first player = 1.0, second player = 0.0 and battle round = -1.0). *Board comparator sign* tells the agent what its board comparator sign is (¡ is -1.0 and ¿ is 1.0).

Operator deck draw chance consists of four floats (one for each operator card) that tell the agent the chance of drawing a specific operator card. The value of these four floats will always be between 0.0 and 1.0 (with the sum of them equal to 1.0). A similar approach is used to obtain the draw chances of the question deck and variable deck by using the *question deck draw chance* and *variable deck draw chance* features respectively.

The *sorted card in hand* feature group represents the state of the eight cards that a player can hold in his hand. Each card consists of 42 float features, with the first feature representing the card's type and the rest of the features being the samples of the card's function (as discussed in Section 4.2.1). The card's type can either be a variable (-0.5), a number (-0.25), or equal to the order of the card in any other case.

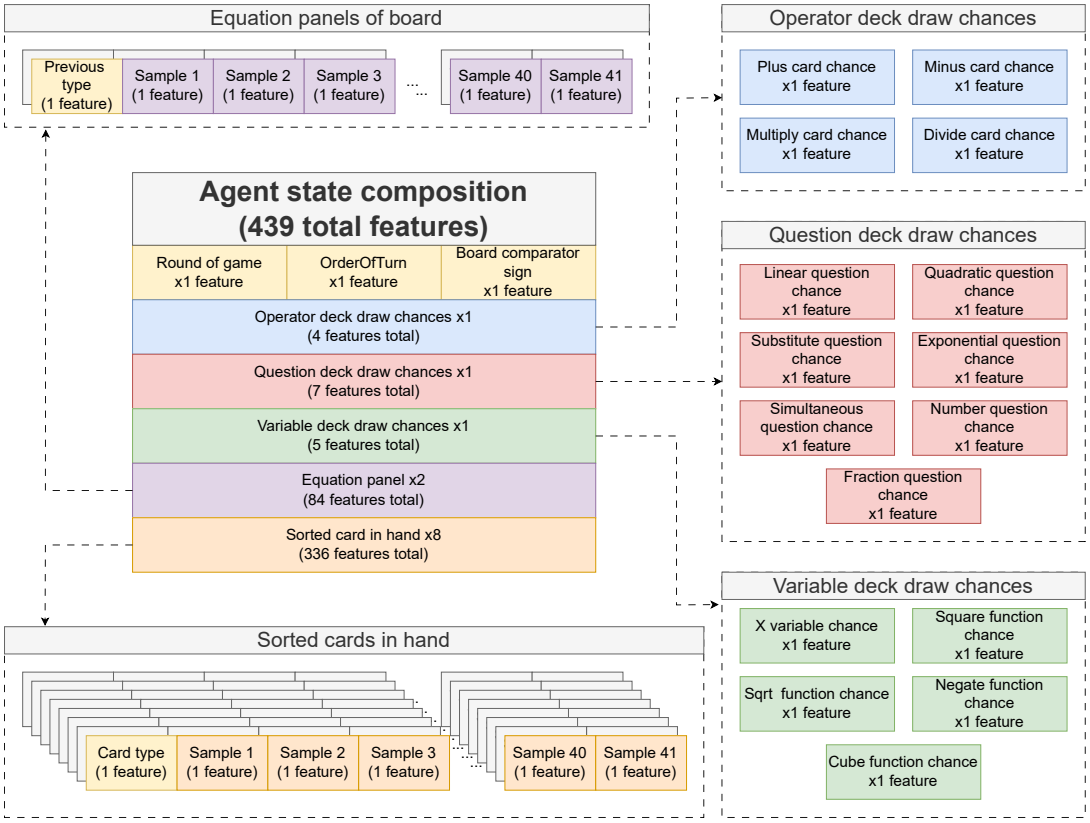


Figure 4.9: Agent state composition

The order of a card is a property of a card that is used to sort a player's hand so that the player's hand has a unique representation. For instance, a player can have a number, variable and operator card in his hands, which means that there are six possible card combinations, but only one possible sorted combination (if the order property of the cards is unique). The float number representing the card's order can be predetermined if the card is static (such as a square function card) or dynamically generated, using a hash of the card's function samples, when the card is dynamic (like variable cards that are forged). Sorting the player's hand before sampling reduces the state space of possible player hands. A sorted hand improves the training efficiency of the agent since some of the "different" states are treated as the same state once sorted, thereby avoiding the need of spending training steps to learn that some states are the same once sorted.

Lastly, the *equation panels* feature group presents the state of the game's two equation panels. The features of a panel are very similar to that of a card, with the *previous type* feature storing the card type that was previously played on the equation panel and the other 41 features being samples of the panel's mathematical equation. The game's left equation panel is always sampled before the right equation panel, irrespective of the value of the comparator sign.

The observant reader might have noticed that the agent's state representation can be simplified by removing the *comparator sign* input feature and always sampling the equation panel that the agent attempts to minimize before the equation panel that the agent tries to maximize (instead of always sampling the left panel before the right panel). Although this suggestion would lead to a smaller state space to search through (increased performance), the added likelihood of breaking core functionality when implementing such a fundamental change was too great for the time available. The complexity of synchronizing RL agent changes with foundational game changes is regrettably one of the disadvantages of using RL (discussed in Section 4.4). The next subsection investigates the action representation of the agent.

4.2.3 Total game action representation

The action representation of the agent is shown in this subsection. The agent can only take a single action per decision requested. Figure 4.10 shows that the single action can be one of 411 possible choices. The # symbol in Figure 4.10 indicates the action number of a specified action. Actions are classified into three primary groups, namely *play card in hand*, *press buttons*, and *forge sorted cards*.

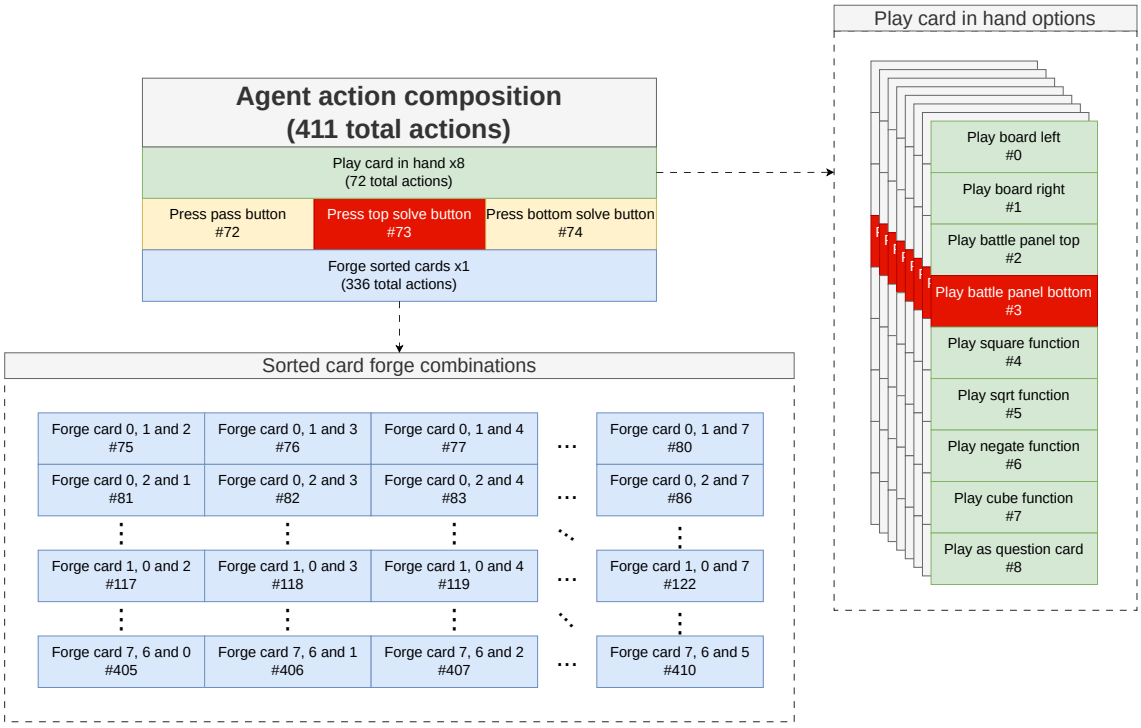


Figure 4.10: Agent action composition

The *play card in hand* group conveys the actions that can be taken on each card of the player's hand. The possible actions available to a card include playing it on the left or

right equation panel, playing it in the battle round, playing it with one of the four function cards, or playing it as a question card (if the card is a question card). For example, an action number of zero states that the first sorted card in the player's hand should be played on the left equation panel, whereas an action number of 9 states that the second sorted card should be played on the left equation panel, and an action number of 18 states that the third sorted card should be played on the left equation panel.

An action number of 72 states that the agent want to press the "pass turn button" while an action number of 73 states that the top agent/player of the game board want to press the solve button in the battle round. An action number of 74 states that the bottom agent/player of the game board want to press the solve button in the battle round. Figure 4.10 shows that the actions highlighted in red can be removed to improve performance since the distinction of top and bottom player is irrelevant to the agent. The agent can only be either the top or the bottom player. Therefore, stating that the agent want to press the solve button automatically means that it wants to press *its* solve button. The top/bottom distinction was required for a mechanic used in a previous version of the game (the mechanic has been removed in the final game version). Again, the distinction has been kept to reduce the risk of introducing glitches when attempting to remove the distinction.

Finally, action numbers 75 to 410 are used by the agent to indicate which cards should be forged together. As an illustration, from Figure 4.10 it can be seen that an action number of 75 requires the zeroth, first and second sorted cards in the player's hand to be forged together, while an action number of 406 requires the seventh, sixth and first sorted cards to be forged together (in that specific order). It is therefore apparent that the majority of actions are dedicated to describing the forge process ($\approx 82\%$).

However, it should be remembered that not all forge actions are possible to do, since the forge specifically requires two variable/number cards on the sides and an operator card in the middle. The invalid forge combinations are masked/disabled in the decision-making process (see Section 4.1). Disabling some of the actions means that the action space is likely smaller than it appears.

4.2.4 Reward shaping

Reward shaping is an integral part of agent design since the rewards ultimately convey the goal to the agent [7]. The process of reward shaping generally requires adjusting rewards with normalization, clipping, transformations or encoding the reward using a model [7].

Luckily, the self-play framework of Section 2.3.3 already shapes the reward by simply attributing reward at the end of the episode/game according to the game outcome. The winning player obtains a reward of 1.0, the losing player receives -1.0, and a draw yields

a reward of 0.0 for both players. This reward policy defines the only reward that is given to the agents during training. The simplicity of this reward structure, together with:

$$\text{Probability of winning in state} = \frac{ExpectedReturnState + 1}{2} \times 100\% \tag{4.2}$$

allows one to interpret the expected return of a state as the probability of winning in that state.

4.2.5 State exposure

It was shown in Section 2.3.1 that the state space size of an environment directly influences the performance/training efficiency of an agent. It is therefore advantageous if the state space size can be reduced so that an agent only learns the states that are directly relevant to the problem. Luckily, Section 3.6.2 shows that one method of reducing the state space size is to limit the card distributions of the decks used during training to reflect only those that actually appear in the levels of the game.

Therefore, training is done by randomly choosing a deck composition from one of the game's levels to use at the start of a new episode. Taking random card decks from real game levels helps to generalize the agent to play the different game levels without requiring the agent to learn *ALL* the possible tactics that appear with completely random deck generations. Many of the completely random deck distribution's states will never be encountered in the game's levels.

This study gave each level's deck distribution an equal chance of being selected, but an interesting alternative would be to increase/decrease the probabilities of choosing a deck distribution for an episode based on the perceived difficulty of a level. It can then be evaluated if the non-uniform selection improves the agent's performance compared to the uniform selection.

4.3 RL algorithms used

Section 4.1 asserts that the ML-Agents framework of the Unity game engine is used to implement the RL agents. This section discusses two RL algorithms, Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC), that are native ML-Agents algorithms [58]. PPO and SAC are used for implementing this study's agents. These two algorithms were chosen due to them satisfying the requirements of the agent design (continuous state space with discrete action space), providing state-of-the-art results, and both algorithms are policy gradient methods (don't need as much feature engineering and have natural stochastic actions that are likely optimal for MARL, as seen in Section 2.3.3) [7][58].

Furthermore, PPO and SAC have their own learning styles that influence the policy that they adopt [7]. PPO attempts to improve the stability of its policy by not making too large adjustments to policies that are known to work [7][74]. On the other hand, SAC is very curious and will choose options that have surprising outcomes, which again leads to states with even more surprising options to choose from [7][52]. The algorithms responsible for these behaviours are given in the following two subsections. It is shown that the two algorithms are mostly defined by their loss/objective function since both algorithms rely on policy gradient methods [7]. The hyperparameters for the RL algorithm implementations of the ML-Agents framework are given in Appendix D.

4.3.1 Proximal policy optimization

PPO stems from the trusted region policy optimization (TRPO) policy gradient algorithm, which seeks to improve the stability of gradient updates by constraining how much the updated policy may differ from the old policy [7]. The loss function of TRPO is minimized by enforcing a limit on the magnitude of the KL-divergence[1] between the updated policy and the old policy [7]. Unfortunately, enforcing TRPO's KL-divergence constraint is quite intricate since one cannot use normal stochastic gradient descent when there is a constraint imposed on the gradient calculations [7]. Instead, TRPO requires the use of conjugate gradient descent, which entails complex calculations [7].

Alternatively, Schulman et al. propose to approximate the constraint of TRPO by simply adding or subtracting functions to the loss function [7][74]. The loss function of PPO therefore consists of three sub-functions, as seen in:

$$L^{CLIP+VF+S}(\theta) \doteq \mathbb{E}\left[L^{CLIP}(\theta) - C_1 L^{VF}(\theta) + C_2 \mathscr{H}[\pi_\theta](s)\right] \tag{4.3}$$

where $L^{CLIP}(\theta)$ is the clipped loss function, $L^{VF}(\theta)$ is the value loss function, and $\mathscr{H}[\pi_\theta](s)$ is the entropy bonus function [7][74]. The hyperparameters C_1 and C_2 are weights that convey the importance of the $L^{VF}(\theta)$ and $\mathscr{H}[\pi_\theta](s)$ terms respectively [7][74]. The definition of the clipped loss function ($L^{CLIP}(\theta)$) is given by:

$$L^{CLIP}(\theta) \doteq \mathbb{E}[\min(\ r(\theta)A(s,a),\ \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{a}(s,a)\ )] \tag{4.4}$$

with $r(\theta) \doteq \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$ being reminiscent of the importance sampling ratio (discussed in Section 2.3.1), and $A(s,a)$ is the advantage function (see Section 2.3.2) [7][74]. Schulman et al. indicate that a suitable approximation for the advantage function of PPO is [74]:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \tag{4.5}$$

$$\text{where} \quad \delta_t = r_t + \gamma V(s_t + 1) - V(s_t)\ . \tag{4.6}$$

[1] KL-divergence is a measurement used to determine the difference between the probability distributions of two policies [7].

T in Equation 4.5 refers to the number of time-steps that $\pi_{\theta_{old}}(a|s)$ should be run for before computing the advantages and updating the loss function [74]. The value for T must be much less than the episode length [74]. It should be noted that PPO has an implicit T step delay and is an on-policy algorithm (must follow the $\pi_{\theta_{old}}(a|s)$ policy) when using this implementation [7]. Luckily, multiple agents can perform $\pi_{\theta_{old}}(a|s)$ in parallel to increase the tempo of experiences generated, thereby improving the algorithm's wall-clock time [74][7].

An intuitive feel for the clipped loss function (Equation 4.4) is gained by inspecting Figure 4.11 [74]. The figure showcases the clipped loss function at various importance sampling ratios and is separated according to positive/negative advantages. The ϵ hyperparameter determines when the loss function is clipped [74]. Positive advantages (left graph) indicate an improvement and clip the loss at $1 + \epsilon$ to prevent overly optimistic updates [74][7]. A negative advantage (right graph) indicates suboptimal decisions and is clipped at $1 - \epsilon$ to prevent overly optimistic improvement of the loss function when the probability of the new inferior policy is reduced ($r(\theta) < 1$) [74][7]. Therefore, PPO prefers a pessimistic approach with the clipped loss function so that the worst possible loss for the parameters is obtained [74][7]. This pessimistic approach ensures more stable updates [74][7].

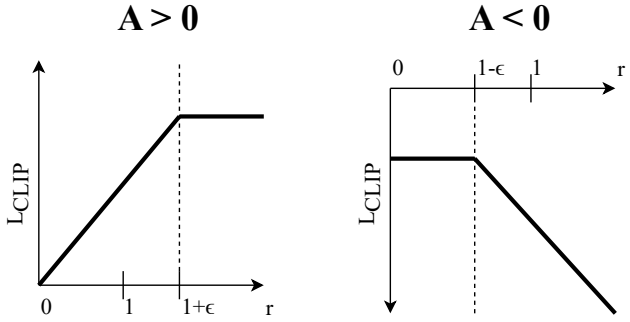


Figure 4.11: PPO's clipped objective [6]

Returning to Equation 4.3, the value loss function is defined to be the square difference between the behaviour and target value functions ($L^{VF}(\theta) \doteq (V_\theta(s) - V^{target})^2$) [7][74]. The value loss function is included in Equation 4.3 to enable DL frameworks to learn the state-value function at the same time as the policy [7][74].

The entropy function $\mathscr{H}[\pi_\theta](s)$, also known as *Shannon entropy*, is added to encourage exploration by motivating the agent to visit states with a *higher* entropy [7]. The Shannon entropy of a random variable (X) is defined as the expected information ($I(X)$) of that random variable [7]. The mathematical equation for Shannon entropy is given by:

$$\mathscr{H}(X) \doteq \mathbb{E}[I(x)] = -\sum_{x \in X} p(x) log_b(p(x)) \tag{4.7}$$

with the base of the logarithm usually taken as $b = 2$ so that the output unit of entropy is in *bits* [7]. Notice from Equation 4.7 that the entropy of a random variable is influenced by the number of values that a random variable can assume, the probability of the variable being a specific value, and the information ($log_b(p(x))$) contained within that specific value. Therefore, the entropy function of Equation 4.3 is increased by policies that have many possible states and a high chance of visiting each one of those states.

The entropy function's maximization nature may come at somewhat of a surprise[2] to readers from a data compression background. The data compression field often desires to have data with a *low* entropy, because it is then theoretically easier to compress. Nevertheless, a higher entropy is desired in RL since it will lead the agent to explore states that have increased exploration potential [7]. Entropy discourages premature deterministic policies [7].

Hyperparameters used in the PPO implementation of the ML-Agents framework are given in Table D.2 of Appendix D. The subsequent subsection investigates the Soft Actor-Critic algorithm, which takes the entropy concept to a new level by making it its primary objective to obtain a policy that maximizes entropy [7].

4.3.2 Soft actor-critic

This subsection presents the soft actor-critic (SAC) RL algorithm. The SAC algorithm is special due to it being an off-policy policy gradient RL algorithm with no hyperparameters [7][52]! The soft in the name of SAC emphasizes that the algorithm relies on non-deterministic behaviour [7]. SAC's primary objective is to obtain the policy parameters that maximize the Shannon entropy of the policy [7]. The entropy used by SAC's objective function is the same Shannon entropy that was discussed in Section 4.3.1. Also, a side effect of maximizing the entropy is that the agent attempts to visit *all* states and actions, which can yield policies that are more robust against adversarial attacks [7].

Therefore, SAC's goal of maximizing the entropy of a policy can mathematically be written as:

$$\pi^* \doteq \underset{\pi}{\operatorname{argmax}} \sum_{s \in \mathscr{S}, a \in \mathscr{A}} \mathbb{E}\left[r + \alpha \mathscr{H}(\pi(a|s))\right] \tag{4.8}$$

where α in Equation 4.8 is known as the *temperature* [52][7]. The temperature determines the influence of entropy on the agent's objective [7]. Setting the temperature too high can cause the agent to value entropy above actually obtaining the maximum long-term reward [7]. Contrarily, having no entropy in an RL algorithm will yield deterministic/hard policies [7].

[2]The pun is intended

Using Equation 4.8 as motivation, Haarnoja et al. made use of KL divergence and Stein variational gradient descent to obtain a loss function for the policy's parameters [7][52]. The loss function for the policy parameters is provided as [7][52]:

$$L_\pi(\phi) = \mathbb{E}_{S_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}[\alpha \log \pi_\phi(f_\phi(\epsilon_t; S_t)|S_t) - Q_\theta(S_t, f_\phi(\epsilon_t; S_t))] \tag{4.9}$$

$$\text{with} \quad A_t = f_\phi(\epsilon_t; S_t) \ . \tag{4.10}$$

Notice from Equation 4.9 that $\alpha \log \pi_\phi(f_\phi(\epsilon_t; S_t)|S_t)$ is the actor (parametrized by ϕ) and the $Q_\theta(S_t, f_\phi(\epsilon_t; S_t))$ term serves as the critic (parametrized by θ) [7]. $\mathcal{D}$ in Equation 4.9 conveys that the distribution of the states should match that of the replay buffer or previously sampled states [52]. ϵ_t is input noise that can be distributed according to a fixed spherical Gaussian distribution [52]. Equation 4.10 conveys that the actions (A_t) of the agent are parametrized with a reparametrization trick to yield a lower variance estimator ($f_\phi(\epsilon_t; S_t)$) [52].

Subsequently, the parameters of the action-value function $Q_\theta(S_t, f_\phi(\epsilon_t; S_t)$ can be obtained by minimizing the soft Bellman residual loss function that is given as [7][75]:

$$L_Q(\theta) = \mathbb{E}_{(S_t,A_t)\sim\mathcal{D}}\left[\frac{1}{2}(Q_\theta(S_t, A_t) - \hat{Q}(S_t, A_t))^2\right] \tag{4.11}$$

$$\text{with} \quad \hat{Q}(S_t, A_t) = R_t + \gamma\mathbb{E}_{S_{t+1}\sim\rho}[V_{\bar{\theta}}(S_{t+1})] \ . \tag{4.12}$$

Equation 4.12 presents an approximation of the action-value function by using the Bellman equation (see Equation 2.9) [75]. The parameters of the action-value function are exponentially averaged according to $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$ [75]. $\tau \in [0, 1]$ determines the weight of new observations on the exponential average [75]. The state-value function in Equation 4.12 aims to improve entropy (encourage exploration) and is consequently given by Haarnoja et al. as [7][75]:

$$V(S_t) = \mathbb{E}_{A_t\sim\pi}[Q_{\bar{\theta}}(S_t, A_t) + \alpha\mathscr{H}(\pi_\phi(A_t|S_t))] \tag{4.13}$$

$$= \mathbb{E}_{A_t\sim\pi}[Q_{\bar{\theta}}(S_t, A_t) - \alpha \log(\pi_\phi(A_t|S_t))] \ . \tag{4.14}$$

Equation 4.14 expands on Equation 4.13 by substituting the definition of Shannon entropy (Equation 4.7) [7].

Finally, Haarnoja et al. found that the SAC algorithm can be turned into a hyperparameter-free RL algorithm[3] by obtaining a loss function for the temperature parameter [7][75]. SAC is turned into a hyperparameter free algorithm by using the loss function

$$L_\alpha = \mathbb{E}_{a_t\sim\pi_t}[-\alpha \log(\pi_t(A_t|S_t)) - \alpha\overline{\mathscr{H}}] \tag{4.15}$$

to constrain the current policy to have a higher entropy than the expected minimum entropy of the solution [7][75]. $\overline{\mathscr{H}}$ in Equation 4.15 represents the expected entropy

[3]The ANN of SAC still has hyperparameters. It is just that the SAC algorithm itself has no real hyperparameters [7].

and is defined as the negative of the number of possible actions ($\overline{\mathscr{H}} = -dim(\mathscr{A})$) [7]. Therefore, the SAC algorithm can be implemented by minimizing the loss functions of Equation 4.9,Equation 4.11 and Equation 4.15 with automatic differentiation (see function approximation methods in Section 2.3.2). It should be noted that the naive SAC algorithm shown in this subsection only works with continuous actions (due to the Gaussian action assumption of the reparametrization trick in Equation 4.10) [7]. Minor alterations must be made to enable the usage of SAC with discrete action spaces [7].

Hyperparameters used by the SAC implementation of the ML-Agents framework are given in Table D.3 of Appendix D. P. Winder mentions that the starting, stopping, and annealing rate parameters of SAC are important since they influence the trade-off between robust policies (attempts to learn from all states/possibilities) and optimal policies (attempts to learn by exploiting the optimal known policy) [7].

4.4 Notable RL implementation difficulties

A quick note on RL implementation difficulties is presented in this section. The main cause for implementation difficulties is the long-running nature of training an RL agent, and the agent attempting every possibility in every state. For instance, a small memory leak can easily lead to out-of-memory errors when the agent is trained for hours and the leak piled up.

Also, the long-running nature of training an agent can cause a bug with a 0.001% chance of occurring, to consistently occur by just training long enough. This challenging training environment means that libraries used in the RL project must be bug-free, otherwise consistent crashes *WILL* occur. The computer algebra system (CAS) library used in the game is an example of a library where the agent ran into edge cases that the library didn't account for and resulted in complete training freezes. Additional effort must then be exerted to fix the 0.001% bugs that cause the entire training process to halt.

Adding additional functionality to the game can also lead to supplementary 0.001% bugs that must once again be tracked and fixed. Added functionality must be incorporated with the RL agent's logic and can lead to further code complexity. The sorted card indexes used by the agent in Section 4.2.2 are an example thereof, where a record must be kept of the card's index in the player's hand (what the player visually sees) and the card's sorted index (what the agent perceives as the state).

Furthermore, each and every change made to the game can have a performance impact. Minor performance regressions have a serious impact on the wall time of agent training, since tight program loops are repeated much more often and over longer periods when compared to normal gameplay. The long-running nature of the training environment

means that even a 100 ms additional delay in the core game loop can cause the training time to prolong by hours.

Ultimately, the worst implementation difficulty is thinking that a 0.001% bug has been fixed only to discover that it still persists once an hour of training is halted. Having bugs with a low probability of occurring means that it was sometimes necessary to wait for hours to track game bugs and implement potential fixes. However, it is observed that the SAC algorithm often finds the 0.001% bugs quicker than the PPO algorithm. SAC's effectiveness in finding bugs is likely due to SAC's exploratory nature.

4.5 RL design summary

This chapter presented the design of the RL agent by introducing Unity's ML-Agents framework and how the state of a mathematical function can be represented. The state of a mathematical function can be obtained by simply sampling the function. Important aspects of sampling a function include the normalization of sampled values (clamped DRC) and choosing which values to sample (constant non-uniform sampling).

The total state and action composition of the agent was also given. The agent supplies a single discrete action by depending on a continuous state representation. Furthermore, reward shaping of the agent and state exposure was explored to improve agent performance/training efficiency.

Two RL algorithms applicable to the agent (PPO and SAC) were mentioned and briefly investigated. The PPO algorithm is known to stay with policies that are identified to be good (improves stability), while SAC is more curious and prefers actions that lead to states with even more options (explores). It should be noted that the state, action and reward representation of an agent can have a more profound impact on performance than the RL algorithm itself [7].

Lastly, the chapter concluded with the difficulties of implementing RL algorithms. The most notable difficulty is that infrequent bugs have a high chance of occurring during RL training due to the training being a long-running process. These bugs can completely halt training and can be difficult to track down due to their infrequent appearances.

RL has some advantages, such as automatically play testing the game and automatically adapting to game changes if the state, action, and reward representation remains the same. The next chapter investigates the performance of the SAC and PPO algorithms by using the state, action and reward representations that were developed in this chapter.

Chapter 5

RL agent performance

The performance of the RL agent designed in Chapter 4 is evaluated in this chapter. In particular, this chapter aims to answer the study's research question that asks "which reinforcement learning algorithm is more applicable to the serious game?". Chapter 4 proclaimed PPO and SAC as the contending RL algorithms of this study that need to be compared.

It should be remembered that RL is notorious for producing noisy results [7]. The noisy results mean that the simulations in this chapter had to be run a number of times in order to gain certainty in the results. However, the simulations can be time-consuming and only a limited number of simulations could be run with the available computational resources. Statistical methods are therefore employed to define the uncertainties in the stochastic nature of the results.

The chapter commences with Section 5.1 which investigates the hyperparameter tuning of the two algorithms. Section 5.2 uses the tuned hyperparameters to evaluate the performance of the RL algorithms. The RL algorithms' performance evaluation is done by comparing the ELO scores obtained by the agents (see Section 2.3.3 for a background on ELO). The conclusion reached by Section 5.2 is validated in Section 5.3 with a tournament in which the agents battle against each other to show that the actual outcomes of the tournament can be predicted with ELO scores. Finally, Section 5.4 investigates the behaviour of the agents in order to determine how the RL algorithms' policies differ, and how it influences their ELO scores. A conclusion to this chapter is given in Section 5.5.

5.1 Hyper parameter tuning and final agents

The hyperparameter values of an RL algorithm can drastically affect the performance of an agent, even to the extent where an agent can become unstable when using the wrong

hyperparameter values [7]. It is therefore paramount that the optimal hyperparameters of the RL algorithms are used in order to make a fair comparison between the performance of PPO and SAC.

Hyperparameter tuning can be done manually by changing the hyperparameters, observing the effect thereof, and repeating the process until a satisfactory hyperparameter combination is found. However, manually tuning is time-consuming and makes it difficult to substantiate that the same amount of tuning effort has been invested in both RL algorithms. Manual tuning makes it difficult to ensure that both algorithms are tuned equally well. The inconsistencies of manual tuning establish the need of using automatic hyperparameter tuning.

The three main methods used for automatic hyperparameter tuning are grid search, random search, and Bayesian optimization [53]. Grid search uses a brute force approach by evaluating all specified combinations of hyperparameters and noting which combination produces the optimal results [53]. Random search attempts to reduce the search space by randomly choosing hyperparameter combinations and relying on chance to land on a hyperparameter combination that is near-optimal and better than the other random combinations that were attempted [53]. It is apparent that grid search and random search are simple to implement, but somewhat inefficient since they do not take into account the results of previous hyperparameter attempts to guide subsequent tuning attempts.

On the other hand, Bayesian optimization uses the results of past hyperparameter combinations to construct a *surrogate function* that predicts the performance that a combination of hyper-parameters will yield [76]. The surrogate function is much quicker/less computationally expensive to evaluate than repeatedly retraining an agent on different hyperparameters [76]. Using a surrogate function means that it is feasible and quick to directly search the surrogate function's predictions for optimal hyperparameter combinations whereas it is computationally impractical to do the same with continuously retrained RL agents. The fit of the surrogate function can be improved by choosing another set of hyperparameters (x_t) and observing the agent's retrained performance response (y_t). The new observations can be added to the observation history ($\mathcal{H}$) that is used to fit an improved surrogate function. Retraining an agent on a new set of hyperparameters in order to capture the actual training performance response is known as running a tuning *trial*. The choice of which hyperparameters to sample for a trial is done according to an *acquisition function*, such as expected improvement (EI) [76][77]. EI works by balancing the next sample choice between regions that are thought to be the new best response (exploit), and regions where there is high uncertainty in the value of the surrogate function (explore) [76][77].

The methodological and more objective approach to hyperparameter tuning makes Bayesian optimization highly desirable in this study, although it can be quite complex to implement compared to other methods. Implementation associated risks are mitigated by using a trusted hyperparameter tuning library. P. Winder recommends the *Optuna* library for hyperparameter tuning [7]. This study used Optuna's Tree-structured Parzen Estimator (TPE) sampler for the Bayesian optimizer with no pruning to stop underperforming trials early.

The number of trials available for hyperparameter tuning was constrained by the computational resources available (see Appendix L for the system used to train the agent). It was seen from experimentation that the agents' training starts to stabilize after 1M training steps, and it takes about a minute to perform 5k training steps. This training time is achieved by already using ML-Agent optimizations such as a dedicated Unity build with 12 areas running in parallel. Therefore, running 100 tuning trials with 1M training steps each, resulted in a tuning time of approximately two weeks. The 100 tuning trials are the bare minimum number of trials that are recommended by the Optuna library for the TPE-sampler [78].

Table 5.1 shows the results of the tuned RL algorithms. It can be seen from the table that the tuned PPO algorithm outperforms the tuned SAC algorithm in terms of ELO scores after 1M training steps. SAC also took longer to train compared to the PPO algorithm. The table shows that both agents found their optimal hyperparameters near the end of tuning (at trial 91 and 92 respectively).

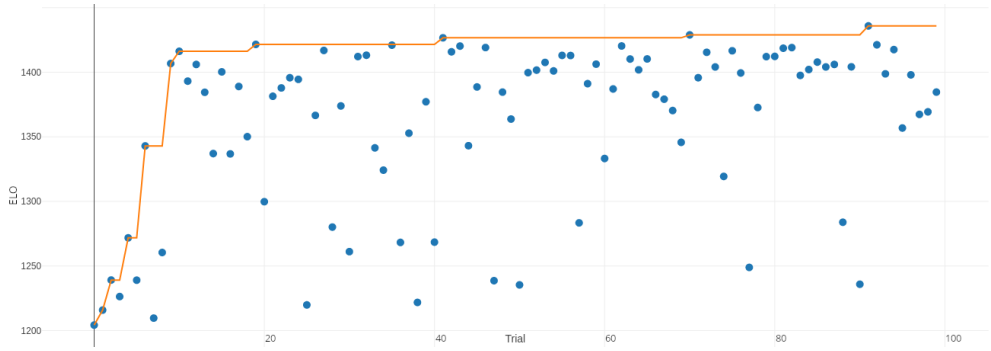
Table 5.1: Hyperparameter tuning statistics for 100 trials with 1M training steps each

Attribute	PPO	SAC
Highest ELO score	1435.78	1374.52
Trial number of highest ELO score	91	92
Tuning time start	2024-06-27 09:52:31	2024-06-27 09:52:45
Tuning time end	2024-07-11 05:21:27	2024-07-17 07:30:40
Tuning duration	331:28:56	477:37:55

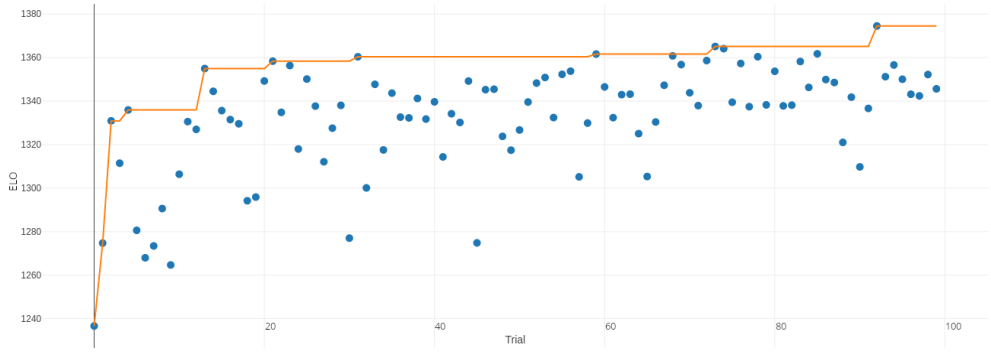
The ELO obtained versus tuning trials is shown in Figure 5.1a and Figure 5.1b. These figures provide validation that the Optuna library worked since the ELO generally increased as the trials progressed. The orange trace shows the maximum ELO achieved by an agent at a specific tuning trial. Trials well below the orange trace are indicative of exploration that was conducted by an agent in search of a more optimal hyperparameter configuration.

It is further observed that 100 trials are an acceptable number of trials to obtain satisfactory hyperparameters for the game's agents, since the ELO improvements near trial 100 are not as pronounced as the ELO improvements that occurred at the start of the

hyperparameter tuning. It is still possible to obtain more optimal hyperparameters than those that are used in the study, but it is likely not worth the added wall time since the study's parameters already took two weeks to obtain.



(a) PPO ELO obtained versus trial number



(b) SAC ELO obtained versus trial number

Figure 5.1: ELO obtained by RL agents during hyperparameter turning

The optimal hyperparameters found in the tuning process are given in Table 5.2. It must be emphasized that the number of hyperparameters and their tune-able ranges have a dramatic effect on the quality of the hyperparameter tuning (number of minimum trials required by the tuner) [78]. More hyperparameters with large tunable ranges cause the search space to exponentially increase [78]. Therefore, the tune-able ranges of the hyperparameters were set according to the suggested ranges of the ML-Agents library given in Appendix D.

The number of tuned hyperparameters was reduced by only tuning the parameters shown in Table 5.2, and using the default values of parameters in Appendix D that do not appear in Table 5.2. Also, all parameters related to self-play were hard set to ensure that the agents train in similar environment setups. The environment's self-play settings were made as diverse as possible in order to generalize the agents better to human players. The hyperparameters used by self-play are: save_steps = 50000, swap_steps = 10000, team_change: = 200000, play_against_latest_model_ratio = 0.5, and window = 30.

Table 5.2: Tuned hyperparameter values

Hyperparameter	PPO	SAC
time_horizon	1449	230
learning_rate	0.0005494	0.0003969
batch_size	467	423
buffer_size	14146	895001
hidden_units	501	447
num_layers	3	2
gamma	0.9086	0.9413
beta	0.004954	
epsilon	0.2395	
beta_schedule	constant	
epsilon_schedule	constant	
lamd	0.9168	
num_epoch	5	
buffer_init_steps		9749
init_entcoef		0.4856
tau		0.005761
steps_per_update		20

The importance of each hyperparameter in the tuning process is given in Figure 5.2 for PPO and Figure 5.3 for SAC. The Optuna library determines the importance of a hyperparameter by using a technique known as functional analysis of variance (fANOVA) [79]. fANOVA values a hyperparameter according to the fraction of variance that it explains [79].

Figure 5.2 shows that the buffer size of PPO had the largest effect on the agent's performance. The chosen buffer size for PPO is small in comparison to the available tuning range, which means that the environment was stable enough for the agent to make frequent updates for a faster learning experience. The importance of SAC hyperparameters in Figure 5.3 illustrates that the number of hidden layers, time horizon, and buffer size had the largest effect on SAC performance. SAC using fewer hidden layers, compared to PPO, might suggest that SAC was prone to overfitting with more hidden layers. The SAC overfitting observation is supported by SAC having a large buffer size (compared to its tunable range) since it means that SAC required a large collection of decorrelated experiences to ensure that model updates are advantageous for most states and not only for recently encountered states. The small time horizon of SAC also shows that the tuner attempted to reduce the variance of estimates (even though it introduces a bias) [58].

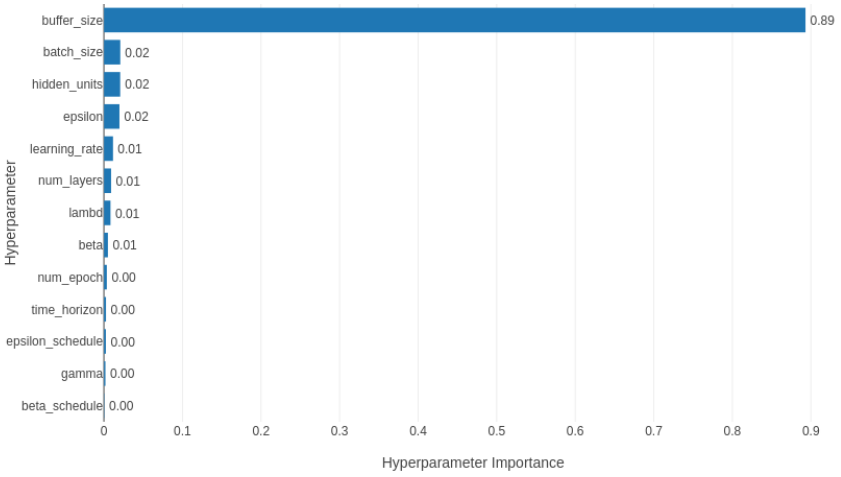


Figure 5.2: PPO hyperparameter importance

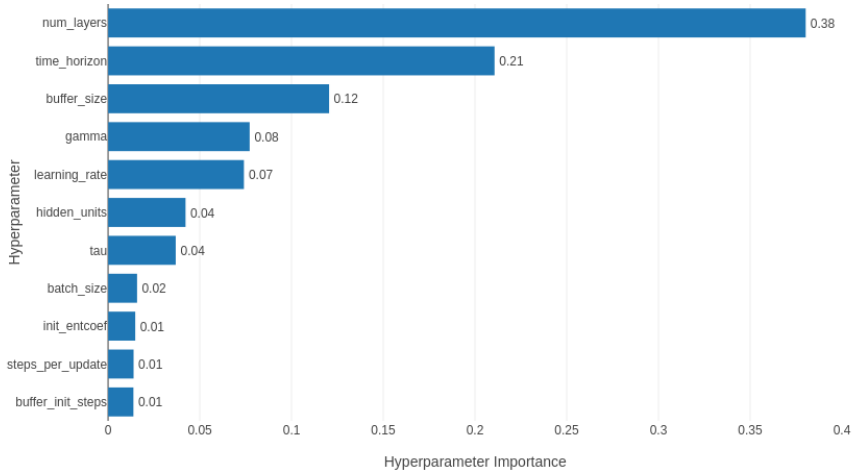
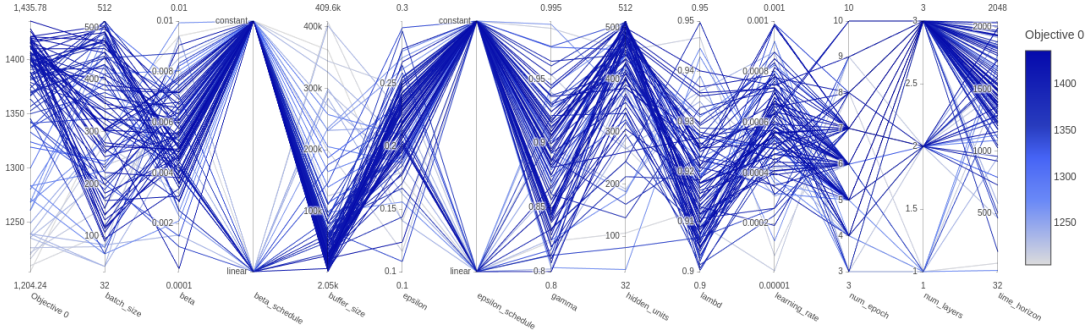
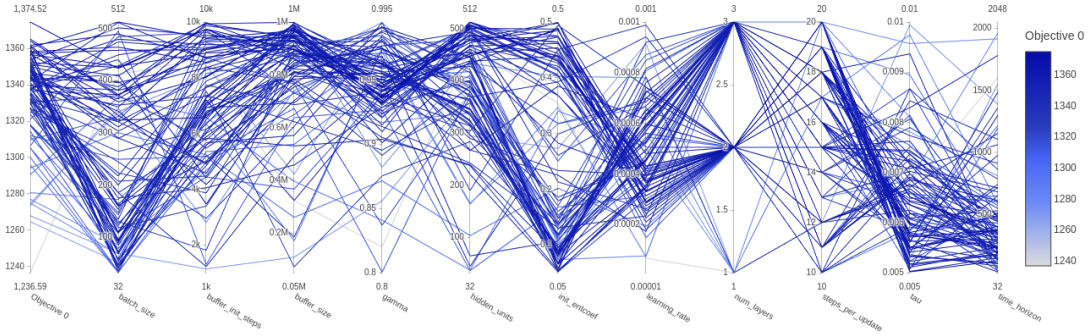


Figure 5.3: SAC hyperparameter importance

Finally, Figure 5.4 shows parallel coordinate charts that indicate the different hyperparameter combinations that were tested, and the results that were produced by the combinations. The charts show that a wide variety of combinations were tested and that some hyperparameters had clear optimal parameter values. The ELO score is labeled as "objective 0" in the charts. The next section compares the performance of the tuned hyperparameter agents.



(a) PPO hyperparameter relationships



(b) SAC hyperparameter relationships

Figure 5.4: Parallel coordinates charts of hyperparameter tuning

5.2 Algorithm performance

The ELO score comparison of PPO and SAC was done by training both agents for 5M steps and comparing their final ELO obtained. The 5M training step count was chosen since it provides a convenient wall time of approximately one day to train the agent (more specifically 22 hours for SAC and 17 hours for PPO). Figure 5.5 shows the average ELO obtained by SAC and PPO. The results are averaged over five PPO runs and five SAC runs. Also, 95% confidence intervals are drawn in a lighter shade by using a one-sample t-distribution and assuming that the runs are normally distributed [51].

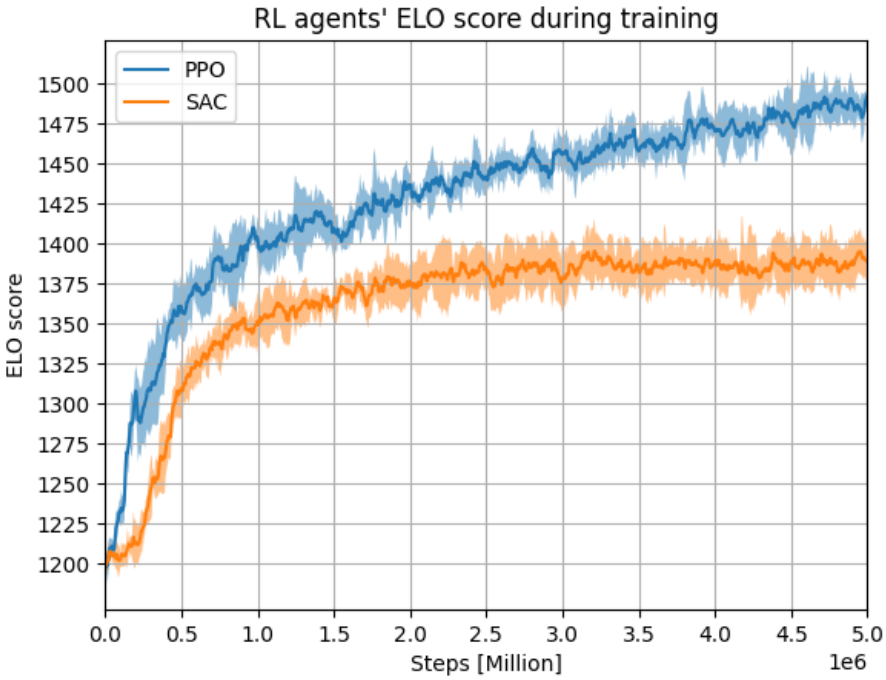


Figure 5.5: Comparison of RL agents' ELO over training steps with tuned hyperparameters

Figure 5.5 clearly shows that PPO outperforms SAC during the 5M training steps since there is almost no overlap in the algorithms' confidence intervals, and PPO's ELO is consistently larger than SAC's ELO. Moreover, the results seem predictable since the width of the confidence intervals remains mostly consistent (uniform variance). The hyperparameter tuning seems to have worked well with both algorithms since they exhibit the characteristic shape of a smoothed step response that is often seen in RL results (such as Figure 2.22). PPO's sudden dip in ELO score near 200k training steps is likely due to the first agent swap that takes place during self-play. The agent swap also explains why the 200k region of PPO has a higher variance than the other regions (suddenly has to adapt to a new opponent for the first time).

Furthermore, Figure 5.5 exhibits an increasing trend for the PPO trace, which can lead to the hypothesis that PPO will continue to improve if it is left to train longer. Therefore, both SAC and PPO have been left to train again for 20M steps in order to test this hypothesis. Surprisingly, Figure 5.6 shows that longer training does not yield the expected ELO increase that an extrapolation of Figure 2.22 suggests, but only a minor ELO improvement over the 5M step training.

The decrease in the ELO improvement rate when using a longer training period is likely due to the learning rate of the PPO agent that is set to decrease linearly. The linear learning rate decrease of the longer training period just takes longer to do the same thing as a linear learning rate decrease of a shorter training period. The 20M training step PPO

outcome was repeated two times and obtained identical results. The SAC agents reached a constant maximum ELO likely due to them using a constant learning rate (as advised by the ML-Agents documentation) [58]. It can therefore be interesting to do future research on the effects that different types of learning rate decreasing functions have on training performance/efficiency.

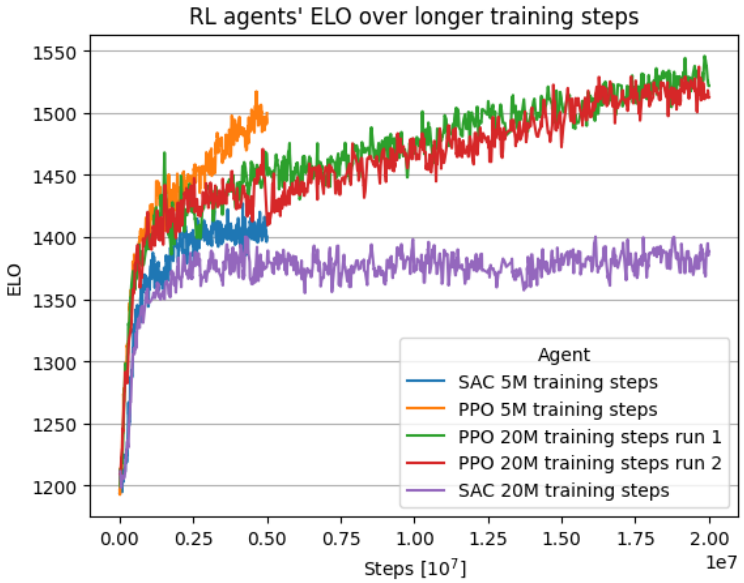


Figure 5.6: Comparison of RL agents' ELO over longer training steps

5.3 Tournament validation of agents

Opponents of the serious game (see Section 3.6.2) were chosen based on Figure 5.5 so that the easiest opponent is PPO at 200k steps (≈1300 ELO), the medium opponent is SAC at 5M steps (≈1400 ELO), and the hardest opponent is PPO at 5M steps (≈1500 ELO). Comparing the ELO scores of the agents (using Equation 2.49) states that the hardest opponent should win against the medium opponent 64% of the time, and the easiest opponent 76% of the time.

This section validates the ELO results by hosting a round-robin tournament in which the three agents of the game duel each other. The tournament is conducted by assigning an agent as the bottom player of the screen (which would normally be the human player) and making it play against all the other agents (including himself) on every level of the game for 100 times on each level. For instance, the easy opponent plays 100 games on level 1 against the hardest opponent, then 100 games on level 2 against the hardest opponent, and so forth until the easiest opponent has played 100 games on each level against every opponent.

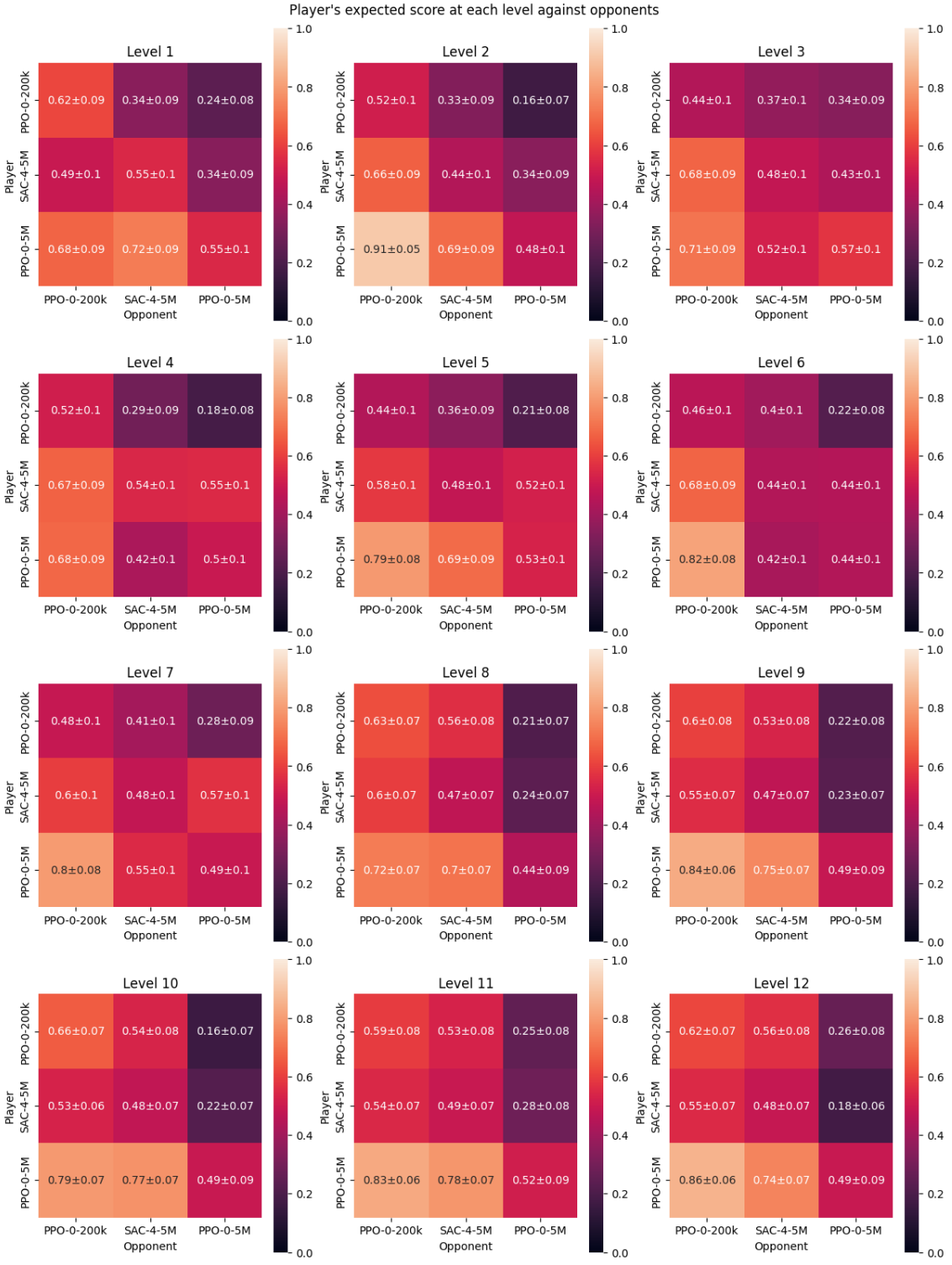


Figure 5.7: RL tournament results

Figure 5.7 shows the outcome of the tournament simulation. The win rate of the matrix is attributed to the row player against the column opponent. For example, the level 2 matrix says that PPO with 5M training steps (row player) has a win rate/expected score

of 91% against the PPO opponent with 200k training steps, a win rate of 69% against the SAC opponent with 5M training steps and a win rate of 48% against itself. The win rate is calculated by averaging wins (coded as a 1), draws (coded as 0.5) and losses (coded as 0) of the 100 games played against an opponent on a level of the tournament. Confidence bounds (95%) were created with the normal distribution, since the central limit theorem applies (100 samples were used per confidence interval, which is more than the 40 samples required by the central limit theorem) [51].

Subsequently, Figure 5.7 proclaims that the PPO agent with 5M training steps is the best (has the highest win rate for most of the levels), followed by the SAC agent with 5M training steps and finally the PPO agent with 200k training steps being the worst agent. It can further be seen that the majority of main diagonal values of each matrix are more or less 50%. The main diagonal having a 50% win percentage is expected since the main diagonal games are the games where the agent played against itself. However, some of the levels where the 200k PPO agent played against itself show that the agent in the row position had a higher win rate than the agent in the column position. The win rate discrepancy is most likely due to the 200k PPO agent having less training experience in playing the top/column position of the board.

Lastly, it can be somewhat surprising that the ELO scores were able to predict that PPO plays better than SAC, even though the agents never played a game against each other. However, both agents initially played against the same baseline agent, which is the opponent that makes random choices. This baseline/random move agent is the reason why the ELO scores can be compared, even though the agents never directly played against one another. The following section seeks explanations for PPO's superior performance in comparison with SAC.

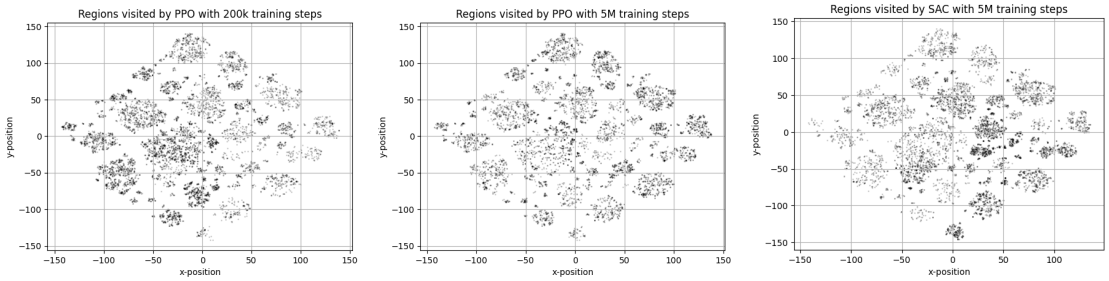
5.4 Explainability of policies

The previous section established that PPO is capable of achieving a higher ELO score than SAC. This section investigates why this is the case by making use of t-distributed stochastic neighbor embedding (t-SNE) plots. The t-SNE algorithm is a dimensionality reduction algorithm that is used to transform high dimensional data into a lower dimensional (often 2D or 3D) representation which can be visually shown [80]. The t-SNE algorithm does this visualization by taking proximal close high dimensional data and arranging the data so that it is also proximal close in the lower dimension [80]. Using t-SNE to create lower dimensional visualizations of high dimensional data means that t-SNE can provide a convenient way of grouping similar high dimensional states (439 features described in Section 4.2.2) of the agent together so that a 2D representation of proximal close states can be created. The t-SNE plot can be used to determine the general actions that are taken in similar states by the agent.

Agent observations for the t-SNE plot were gathered by following a similar setup to the tournament conducted in Section 5.3. The three agents of Section 5.3 played against each other in 100 games on the last level of the game. The last level was chosen due to it containing/testing all the features that are present in the game. Observations of the states and actions performed by the bottom player (the position that the human player normally plays in the game) were logged to be used for the t-SNE plots. Logging the observations of the bottom player resulted in 8036 observations by SAC with 5M training steps, 7666 observations by PPO with 5M training steps, and 8797 observations by PPO with 200k training steps.

The total number of observations (24499) was given to the t-SNE algorithm to reduce the dimensionality (arrange similar states to be in close proximity) to 2D so that the proximal states can be graphed. However, the original t-SNE implementation does not scale well with many samples since it is a $O(N^2)$ algorithm, with N being the number of samples given to the t-SNE algorithm [80]. Instead, a faster implementation was used, known as FIt-SNE, that provided a good approximation to t-SNE, but ran in $O(\ 2pN)$, where p is the number of interpolation nodes used by the algorithm [80]. The plots generated in this section fixed the FIt-SNE hyperparameters to max_iter=3000, perplexity=25 and seed=3. The rest of the hyperparameters were left at their default values.

Figure 5.8 presents a t-SNE plot of the states visited by each agent. It must be emphasized that the samples generated by the three agents were first united together, then run through the FIt-SNE algorithm (to position similar samples close to each other), and afterwards split again to produce Figure 5.8. The figure is used to determine which agent contributed to which state area of the t-SNE plot.

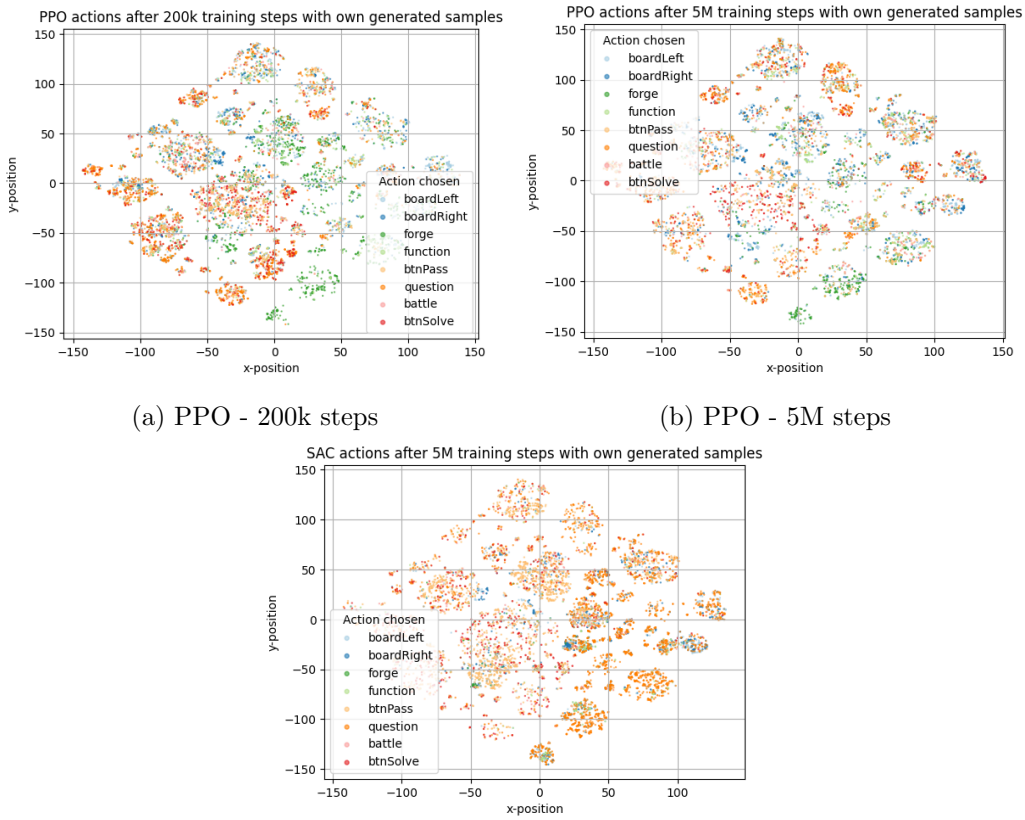


(a) PPO - 200k steps (b) PPO - 5M steps (c) SAC - 5M steps

Figure 5.8: t-SNE plot of states visited by agents

Care should be taken when reading the t-SNE plots since the separation distance between clusters and the size of the region occupied by a cluster has no meaning [81]. Instead, a cluster only indicates the grouping of similar states without telling how similar those states are to the next nearest cluster. Subsequently, Figure 5.8 shows that the distribution of the type of states visited by each agent differs. The easiest opponent (PPO with 200k training

steps) had insufficient training, since it attempts to visit/explore a variety of states (dark sample densities in plenty of clusters). Contrastingly, PPO with 5M training steps seems more set in its ways by having fewer clusters with dark cluster densities. The reduced cluster densities of the 5M PPO clusters means that PPO with 5M training steps has only a few states which it highly prefers to be in. Lastly, the SAC agent seems to also enjoy exploration, just like the PPO agent with 200k training steps, but prefers to visit different state regions compared to PPO with 200k steps.



(a) PPO - 200k steps

(b) PPO - 5M steps

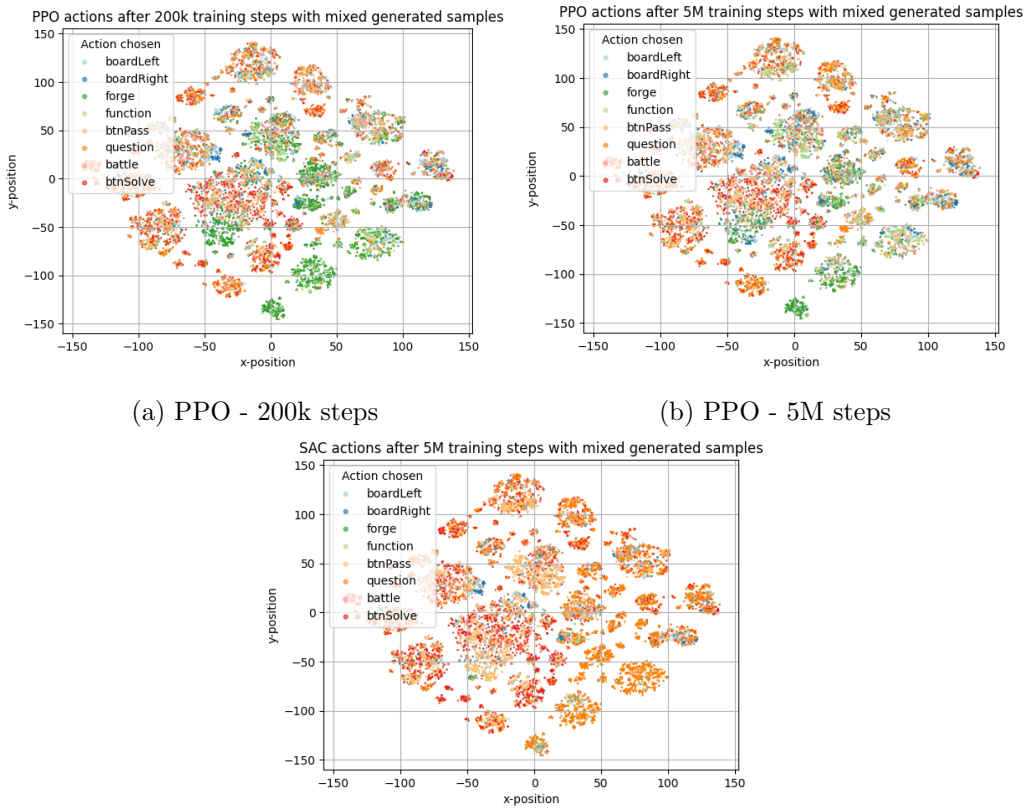
(c) SAC - 5M steps

Figure 5.9: t-SNE plot of agent actions with own visited states

Figure 5.9 shows the actions performed by the agents on their own generated data. It can be seen that a state cluster contains a variety of actions and not just a single action. The variety of actions in a single cluster is likely due to some high-dimensional data features being lost, causing the lower-dimensional features to appear blended. It is further recommended to use t-SNE as a data visualization algorithm instead of a clustering algorithm.

Finally, Figure 5.10 presents the t-SNE plots of each agent being exposed to the states collected by all the agents (the united observations that were originally given to the FIt-SNE algorithm). Comparing Figure 5.10 with Figure 5.9c shows that each agent continues

to take the same type of actions in each cluster and that the added unseen samples of each agent generalize quite well to the seen samples. Figure 5.10 further shows that there are certain clusters where all three agents take more or less the same actions, but that many of the clusters also differ between PPO and SAC. SAC seems to be much more eager in playing question cards and passing turns while PPO prefers to forge and play function cards. It can also be seen between Figure 5.10a and Figure 5.10b that the PPO agent with 200k training steps takes more or less the same actions as the PPO agent with 5M training steps, with the biggest difference being that the PPO agent with 200k training steps is more likely to forge cards.



(a) PPO - 200k steps

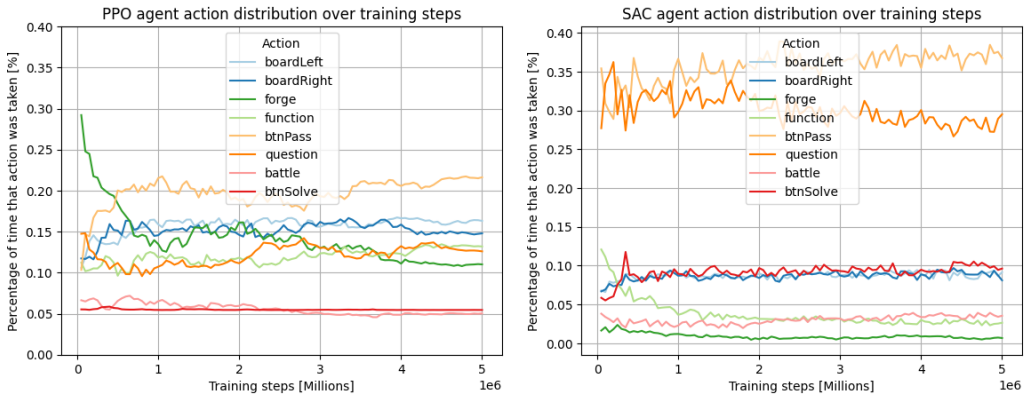
(b) PPO - 5M steps

(c) SAC - 5M steps

Figure 5.10: t-SNE plot of agent actions with mixed visited states

In fact, a plot of the action distribution over training steps for the SAC and PPO agents is provided in Figure 5.11. It is clear from Figure 5.11a that the PPO agent initially had a high preference for forging cards, but later decided to pass turns and forge less. The decrease in forging moves as PPO trains is likely due to the PPO agent learning that some forge combinations are not as good as other combinations, and that it might be worthwhile to wait for the ideal cards to forge. SAC, on the other hand, succumbed to its curiosity by continuously waiting and playing question cards in order to achieve the highest entropy, instead of learning how to manipulate the game board.

It makes sense that question cards yield a high entropy, since they provide a random number card to the agent when played. Therefore, the skill difference between SAC and PPO can be explained by SAC wasting most of its training steps to pass turns and play question cards instead of learning how to forge cards or to manipulate the game board.



(a) PPO (b) SAC

Figure 5.11: Agent action distribution over training steps

5.5 Summary of RL performance

This chapter aimed to answer the research question asking "which reinforcement learning algorithm is more applicable to the serious game?". It is shown in Section 5.1 and Section 5.2 that PPO is a better opponent than SAC and that PPO requires less wall time to train when using the setup of Appendix L. Care was taken to ensure that both algorithms had properly tuned hyperparameters in order to facilitate a fair comparison.

The conclusion of PPO being more applicable to the serious game than SAC was validated in Section 5.3 by means of a tournament. The tournament results confirmed that PPO trained at 5 million training steps won most of the time against the SAC agent that trained for 5 million training steps. It is concluded that the ELO score can fairly predict the performance of two RL agents that never played against each other, since they shared/played against the same baseline opponent (first opponent that only took random actions).

The policies of the PPO and SAC agents were investigated in Section 5.4 to seek a reason for agents' performance difference. It was found that SAC spent most of its time seeking entropy by playing question cards instead of manipulating the game board to be in its favour. PPO showed a strong preference in forging cards, but forged fewer cards as training continued since it likely started to learn the more optimal/advantageous combinations.

Therefore, this chapter showed that RL agents, especially PPO, were able to successfully master the serious game by being able to consistently defeat an opponent that makes

random choices. Moreover, this means that the agent was able to "learn" mathematical concepts such as addition, subtraction, multiplication, division, functions, and variables since mastery of these concepts is required to outwit the opponent of the game. The next chapter investigates the response of real grade 9 learners to the game, in order to determine if the learners react as positively as the RL agents to learning mathematical concepts with the serious game.

Chapter 6

Response of school learners to the game

This chapter investigates the response of real grade 9 learners to the serious game designed in this study. The primary objective of the results in this chapter is to answer the research questions pertaining to learners' responses, which include:

1. Does the serious game improve the algebraic mathematical abilities of grade 9 learners?
2. How long are grade 9 learners motivated to play the serious game?
3. How long does it take for grade 9 learners to master the serious game?

The questions are investigated by means of a field study that utilizes three data sources. The first question is answered with an adapted Trends in International Mathematics and Science Study (TIMSS) test which is used as a pre-test and post-test. The pre-test and post-test are used to determine if there is any mathematical ability improvement when playing the game.

The second and third questions are answered in a more exploratory manner by using a survey together with game analytics of passively captured gameplay data. The field study's ethical clearance can be found in Appendix I, and the informed consent form is given in Appendix H.

Therefore, this chapter consists of five subsections to describe the experimental setup and results obtained. Section 6.1 commences with the experimental methodology that was followed and the reasoning for using a field study structure. Section 6.2 describes how the samples (schools and learners) for the experiment were chosen, while Section 6.3 provides more details of the instruments used in the experiment (adapted TIMSS test, survey and

game analytics). Finally, Section 6.4 provides the results of the experiment and Section 6.5 finishes the chapter with the key experimental takeaways that answer the above-mentioned questions.

6.1 Experimental methodology

The study embarked with the vision of creating a serious game that grade 9 learners can play in their free time to improve their algebra abilities while also nurturing a mathematical interest within them. The results of the study are subsequently used to determine the feasibility of the game and suggest improvements that can be made in future iterations.

It is of cardinal importance that the experimental protocol must not introduce an observer-expectancy effect with the way that learners interact with the serious game. The observer-expectancy effect is often the case with the strict conditions imposed by laboratory experiments [82][83]. Furthermore, Section 2.1.2 shows that the serious game should be played over a long time-span so that the learners can assimilate what they have learned from the game. Providing learners with enough time to assimilate knowledge means that the game cannot be played and tested in a short afternoon, but rather requires a few weeks of play in order to have any chance of success.

Consequently, this study adopted the field study methodology where participants of the experiment played the serious game on their own phone whenever and wherever they wanted to over the span of a few weeks. It is precisely this real-world evaluation of the research that hopes to contribute to societal change and have a lasting impact [83]. The main advantages of a field study include the ability to track change over time and the improved generalization of results (occurs in a natural setting with minimal influence from the experimenter) [82]. However, some unfortunate disadvantages of a field experiment include the reduced control over external variables/influences, and real-world data being especially messy and effortful to obtain (gatekeepers and ethical dilemmas) [82][83].

It was subsequently decided to introduce an experimental and control group to the experiment in order to reduce and gauge the effect of external variables on the study's results [83]. The experimental group received the serious game designed in this study while the control group received the game discussed in Appendix E. Participants of the experiment were unaware that they were placed in an experimental or control group.

The usage of a control group adds some ethical dilemmas to the study since the control group is expected to remain the same while the experimental group is expected to improve due to the intervention [83]. Expecting the experimental group to show an improvement over the control group can sometimes be interpreted as unfairly withholding the control group from a treatment that can benefit them [83].

However, the control group's game in this study was chosen with care and has the potential benefit of improving its players' arithmetic skills. The benefits of the control group's game do not interfere with the objectives of this study since this study aims to improve algebra skills and not arithmetic skills. Ensuring that the control group's game is also a mathematical game has the added benefit of making participants less suspicious as to why there are two groups since both are tasked to test a mathematical game. Therefore, cross-contamination between the groups is minimized by making learners less curious in what the other group is doing and more focused on playing their own game.

A visual summary of the experiment's methodology is provided in Figure 6.1. It can be seen from the figure that the experiment consists of four main phases, namely obtaining informed consent from the participants' parents, writing the pre-test phase, playing the game in the field experiment phase, and finally completing the post-test phase. The exact same test is written in the pre-test as well as the post-test. Using precisely the same test for the pre-test and post-test ensures that the pre-test can serve as a baseline for a participant while the post-test indicates whether there are any changes, compared to the baseline, as a result of the received treatment. Details surrounding the test used for the pre-test and post-test are given in Section 6.3.1.

It must be reiterated that most of the experiment is conducted in a digital manner, with participants using their own Android smartphone to participate. The experiment assumed that participants didn't have access to a mobile internet connection. Therefore, a local server (laptop connected to a WiFi router) was brought to the schools during the pre-test and post-test so that learners can connect to the server and download the games at no cost to them. The apps/games downloaded from the local server needed to be side loaded on the participants' phones, which means that only Android smartphones could participate in the experiment since Apple smartphones did not support the feature.

Making the majority of the experiment digital had the advantage of ensuring that passively captured gameplay data came back during the post-test phase when participants connected with their mobile phones to the local server to write the post-test. Alternatively, the captured gameplay data was also received when a participant played the games with an active internet connection. The local server data transfer was used as a backup for the case where a participant played the games without any internet connection.

Finally, participants were treated fairly at the end of the experiment by giving all participants access to both the experimental and control games for an unlimited period without any conditions. The games given at the end of the experiment had their gameplay capture and experimentation test functionality removed. Participants also received a soft drink and chips after the pre-test and post-test sessions to show appreciation for their time.

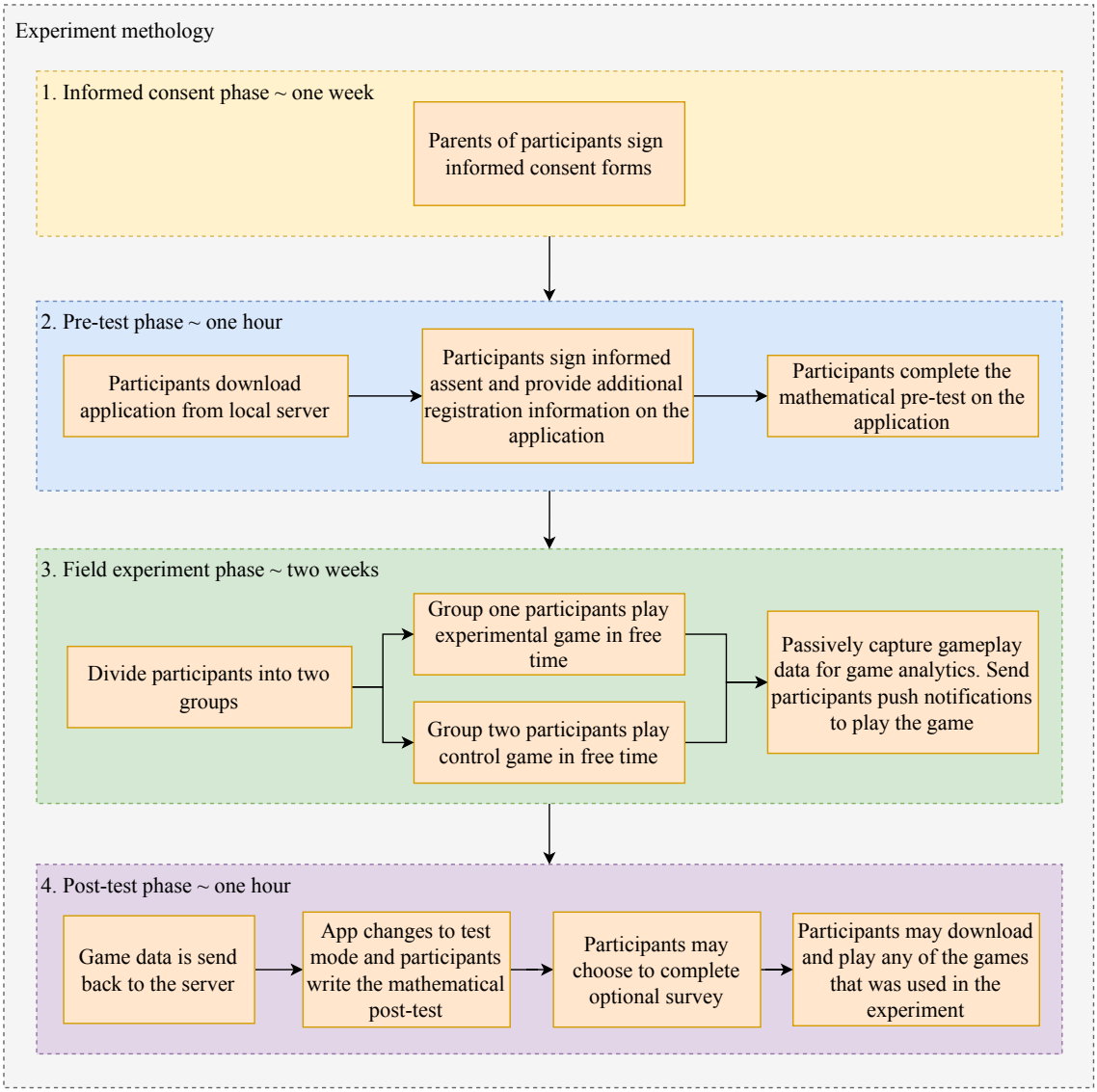


Figure 6.1: Illustration of experiment methodology

It should be noted that the duration of the experimental phases given in Figure 6.1 was somewhat optimistic in the sense that visits could only be made to the schools when it suited the schools and the learners. The participating schools were very helpful and accommodating, but had unforeseen delays such as needing more time to encourage learner participation, learners having tests to write in certain weeks, and school holidays taking place during the experiment. These additional delays meant that the planned field study of 3.5 weeks rather ended up taking 2.5 months. Section 6.2 discusses the schools sampled in the experiment, choice of sample sizes, and dates when each phase took place at the different schools.

6.2 Sampling procedure

The study drew samples from two schools, one school in Potchefstroom (quantile 4 school with 205 grade 9 learners) and one school in Ikageng (quantile 3 school with 230 grade 9 learners). These two schools were chosen based on the feasibility of performing the experiment at the schools. Feasibility factors include the schools having enough learners for a large sample size, schools agreeing to participate in the experiment, learners having access to smartphones, schools allowing learners to bring smartphones to school, and the schools being close to the university in order to provide immediate assistance if a problem occurs. The contrasting socioeconomic statuses between the Potchefstroom and Ikageng schools also helped to expose the games to different grade 9 audiences and evaluate their reactions.

The ideal sample size for the experiment was based on the statistical methods employed in the analysis of the adapted TIMSS test instrument (see Section 6.3.1). The main statistical method used by the adapted TIMSS test instrument is to compare the score between the pre-test and post-test with the paired two-sample t-test and to compare the scores between the experiment and control groups with the independent two-sample t-test. Ideally, these t-tests assume that the samples are normally distributed [51]. The normality assumption can be assured with the central limit theorem, but requires a large sample size [51]. Typically, a large sample size means that the control and experimental group of each school should each have at least 40 participating learners (i.e. at least 80 learners per school) [51].

However, it was anticipated that it could be difficult to reach the ideal sample sizes and that some learners might abandon the study during the experiment. Therefore, informed consent forms (signed by parents) were given to all the grade 9 learners of a school and not just to 80 learners. It was clearly communicated to them that participation in the experiment was completely voluntary. The Potchefstroom school took ≈176 consent forms and had 21 - 30 August 2024 to have their parents sign the form and bring it back to school. The Ikageng school took ≈230 forms and had 16 - 28 August 2024 to sign and bring back the forms.

There were much fewer consent forms received back from the learners than were originally distributed. The Ikangeng school yielded 108 signed consent forms while the Potchefstroom school only had 20 signed consent forms. Surprisingly, the Potchefstroom school's teacher asked the learners why they didn't participate and a large percentage of learners replied that they only own an iPhone. Therefore, some learners at the Potchefstroom school were unable to participate due to the platform of the games (only made for Android smartphones).

The Potchefstroom school wrote the pre-test on the 3rd of September 2024 (after school) with 13 learners attending the session. Unfortunately, one learner could not participate due to her bringing an iPhone to the session, while three other learners could not participate due to family link being enabled on their Android phones. Family link is a parental control feature that prevents the installation of side-loaded apps. One of the three learners with family link issues was given one of our phones to write the pre-test on. Therefore, there were 10 learners at the Potchefstroom school that wrote the pre-test.

The Ikageng school started with the pre-test on the 10th of September 2024, with 59 learners writing the pre-test. About two to four learners at the Ikageng school were unable to install the game due to their phone not meeting the system requirements of the game (either phone storage or Android version being too low). There were no family link issues at the Ikageng school, but there were one or two learners that could not participate since they brought an iPhone to the session.

Participants of the pre-test were randomly divided into the control group or experimental group. The group assignment was done by alternating the players assigned to a group as they connected to the local server for the first time. For instance, the first learner that connected to the local server would be assigned to the experimental group, the second learner to connect would be in the control group, the third learner would again be in the experimental group, the fourth learner in the control group and so forth. The group assignment was done in this manner to increase the likelihood of having equal group sizes compared to generating a completely random number that indicates to which group a player belongs.

However, group sizes obtained with the alternating assignment method can still be uneven since some participants downloaded the game, but were unable to install the game. Therefore, the Potchefstroom school had 6 experimental group participants and 4 control group participants that wrote the pre-test. The Ikageng school had 28 experimental group participants and 31 control group participants that wrote the pre-test.

Finally, 4 learners (2 experimental and 2 control) from the Potchefstroom school came back on the 9th of October 2024 (37 days later) to write the post-test. Similarly, 28 learners (13 experimental and 15 control) of the Ikageng school came back to write the post-test on the 15th of October 2024 (36 days later). Around 5 learners from the Ikageng school had to re-install their games during the post-test (need game to write the test) since they uninstalled their games in the field experiment phase. Gameplay data of these $\approx$5 learners were lost as a result of the uninstall if they did not play their games with an active internet connection. The field experiment phase (originally planned to be 2 weeks) was prolonged for both schools due to the school holiday that took place between 20 September 2024 and 1 October 2024.

It is interesting to observe that approximately half of the pre-test participants did not participate in the post-test at both of the schools. The Potchefstroom school's learners did not provide a reason for the reduced participation. However, the Ikageng school's data showed that only one of the school's classes was able to attend the post-test session. The other classes of the Ikageng school were most likely not informed about the post-test session. Also, the majority of participants that wrote both the pre-test and post-test were female (75% were female at the Potchefstroom school, and 69% were female at the Ikageng school).

6.3 Experiment instruments

This section presents the three instruments that were used to obtain the learner response data. Section 6.3.1 presents the pre-test and post-test that are used to determine if the game improves learners' algebra skills by using formal statistical hypothesis tests. Section 6.3.2 presents the survey that is used to gather insight into the experience that the participants had while Section 6.3.3 presents the gameplay data that is used for game analytics. The survey and game analytics use an exploratory approach to explain the results obtained by the pre-test and post-test, determining how long students are motivated to play the games, and investigating how long it takes participants to master the games.

6.3.1 TIMMS test

It was mentioned in Section 6.1 that the experiment required participants to write two tests. The first test is known as the pre-test and sets a baseline for a student's mathematical score. The second test, known as the post-test, was written after the students were exposed to the field experiment phase. The post-test score is compared with the pre-test score to determine the effect that the serious games had on the participants' mathematical abilities. Furthermore, participants were also assigned to either the control group or experimental group to determine if external validation threats had an impact on the experiment.

More specifically, the significance of differences between pre-test and post-test scores of participants can be ascertained with a paired t-test hypothesis test [51]. The hypothesis test is performed by grouping the pre-test and post-test scores of each participant as a pair so that the difference in test scores of a participant is represented by:

$$D_j = X_{1j} - X_{2j}; \quad j = 1, 2, ..., n \tag{6.1}$$

where D_j represents the difference in score of the j^{th} participant, X_{1j} is the post-test score of the j^{th} participant, X_{2j} is the pre-test score of the j^{th} participant, and j indicates which

one of the n sampled participants is used for the calculation [51]. Consequently, the test statistic (T_0) of the paired t-test is calculated using:

$$T_0 = \frac{\bar{D}}{S_D/\sqrt{n}} \tag{6.2}$$

with $\bar{D}$ being the average test score difference of the sampled participants and S_D portraying the standard deviation of the sampled participants' test score difference [51]. The degrees of freedom (v) for the paired t-test is $v = n - 1$. The test statistic is consequently used in a two-sided hypothesis test with a significance level of 5% ($\alpha = 0.05$) to determine the merit of the null hypothesis (mean difference between tests is zero) or alternative hypothesis (mean difference between tests is not zero). A successful improvement in the participants' mathematical abilities is indicated by a rejection of the null hypothesis.

On the other hand, the ordinary independent two-sample t-test is used to determine if the differences in test scores are dissimilar between the experimental group and the control group. The null hypothesis states that the mean test score difference between the two groups is the same, while the alternative hypothesis states that they are different. The ideal case is for the null hypothesis to be rejected, since it means that the experimental group showed improvements that are not present in the control group. Therefore, the test statistic for the independent two-sample t-test with unknown and unequal variances is given as:

$$T_0^* = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}} \tag{6.3}$$

with $\overline{X_1}$ and S_1^2 representing the mean and sample standard deviation of the experimental group's test score differences [51]. $\overline{X_2}$ and S_2^2 are the mean and sample standard deviation of the control group's test score differences. The degrees of freedom for the independent two-sample t-test with unknown and unequal variances is calculated with [51]:

$$v = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{(s_1^2/n_1)^2}{n_1 - 1} + \frac{(s_2^2/n_2)^2}{n_2 - 1}} \ . \tag{6.4}$$

The above-mentioned independent two-sample t-test with unknown and unequal variances is also performed as a two-sided hypothesis coupled with a 5% significance level. The samples taken from the experimental and control group are assumed to be independent of each other.

However, a fundamental assumption of the independent two-sample t-test and the paired t-test requires the samples to be normally distributed [51]. The normality requirement means that great care should be taken with the pre-test and post-test to ensure that a standardized test is used in order to obtain scores that are likely normally distributed. Furthermore, the tests must measure the aptitude/insight that a participant gathered

from playing the serious game instead of measuring textbook-specific knowledge that the participant retained/memorized (as is often done with conventional school assessments).

Therefore, three potential standardized mathematical tests for grade 9 learners were found that can fulfil the requirements of the previous paragraph, namely the standardized college admission test (SAT) of the United States, the annual national assessments (ANA) conducted in South Africa, and the Trends in International Mathematics and Science Study (TIMSS) test that is used to benchmark the mathematical and science performance of countries internationally.

The TIMSS test was chosen for this study since TIMSS provides a standardized international benchmark for grade 9 mathematics, does not rely on textbook-specific knowledge, and the authors of the test granted permission to use their test in this study (reference IEA-24-027) [84]. Many countries choose to evaluate their grade 8 learners with the TIMSS test, but South Africa distributes the test to grade 9 learners [84].

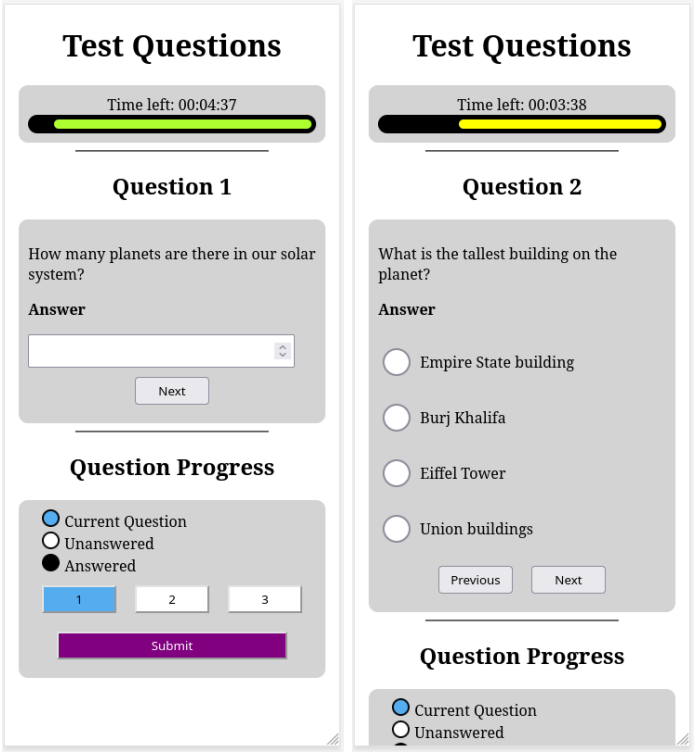
It should be noted that the permission granted by the authors of TIMSS had some conditions imposed. For one, the questions used in this study may not be published or shared with any third parties, other than the participants that wrote the test. The distribution restriction is due to TIMSS sometimes re-using old questions in future tests, and TIMSS does not want test candidates to specifically practice for their tests. A second condition is that the results of this study may not be compared to the results of the official TIMSS study, since the execution of the studies differs.

For instance, one aspect in which the studies differ is the choice of questions that were included in the pre-test and post-test. The mathematical section of the standard 2015 TIMSS test consists of two blocks (a block being a set of 12-18 questions) that must be completed by the test candidate within 45 minutes [84]. In total, the 2015 study had 14 mathematical blocks, with each test candidate receiving a different combination of blocks to answer [84]. As an example, one candidate might receive blocks M01 and M02 to answer, while another candidate could get blocks M12 and M13 to answer. The standard 2015 TIMSS test designed each block so that the questions are distributed to include 30% number, 30% algebra, 20% geometry, and 20% data and chance questions [84]. Furthermore, a block's questions are arranged to include approximately 35% knowing, 40% applying, and 25% reasoning problems [84].

However, this study aims to evaluate if the serious game improved the algebra skills of learners. Therefore, the majority of the study's test should consist of algebra questions in order to increase the sample size of algebra questions answered. Questions stemming from the numbers, geometry, and data and chance categories are still included to provide additional validation in the test, since the experimental game is expected to only improve participants' algebra skills and not necessarily the skills required for the other categories. A change in the other skills may indicate that the test has been compromised.

The authors of TIMSS granted access to use questions from block M01, M05 and M07 of the 2015 TIMSS test. Most of the algebra questions from these blocks were added to this study's test, and effort was made to only include multiple choice or single answer questions in order to ease the marking of the test. The adapted TIMSS test questions are presented in Appendix F, where it is seen that the participants had 32 minutes to answer 15 algebra questions, 5 number questions, 3 data and chance questions, and 2 geometry type questions. The order of the questions was randomized with a random number generator and kept the same in the pre-test and post-test.

The pre-test and post-test of the experiment are exactly the same test (shown in Appendix F), meaning that no alternations or differences have been made between the two tests. Participants of the Potchefstroom school wrote the Afrikaans test while participants of the Ikageng school wrote the English test. Visuals of the digital test platform are given in Figure 6.2. The practice questions of Figure 6.2 were used to show learners how the digital testing platform worked before they commenced with the pre-test and post-test. The next subsection discusses the survey given to participants after writing the post-test.



(a) Fill in question (b) Multiple choice question

Figure 6.2: Examples of question formats found in the digital test

6.3.2 Survey

The survey shown in Appendix G was administered as a hard copy to participants after completing the post test in order to obtain the subjective opinion of participants on the serious games. The survey contains three Likert scale questions, namely how fun the game was to play, how much the participants thought the game helped them, and the difficulty of understanding the game. The survey also contained 4 open questions that asked what the participants liked about the game, what they hated about the game, at what time they normally played the game, and what improvements they would like to see in a future version of the game. It was completely voluntary for participants to complete the survey.

6.3.3 Passive game analytics

Gameplay data was passively captured of participants in both the experimental and control game as they played their games. The data contained no personally identifiable information. Data captured by the games includes the times when games were played, levels that were played, in-game rewards obtained (such as pets rescued), notifications that fired in order to motivate participants to play, gameplay moves that were made in the game, and interactions that participants had with in-game menus. The captured data is granular enough so that the gameplay of a participant can be completely reconstructed.

The gameplay data is analysed in an exploratory manner in order to explain the results that are observed by the other data sources (pre/post tests and survey). The gameplay data was automatically sent to our online server when a participant had their mobile data switched on while playing the game. Alternatively, the gameplay data was received by our local server when a participant wrote the post-test at the end of the experiment.

6.4 Experiment results

The results of the experiment are conveyed in this section. This section is divided into 4 subsections, namely Section 6.4.1 that examines the pre-test and post-test scores of participants, Section 6.4.2 that investigates the duration that participants played the game, Section 6.4.3 that inspects the difficulty that players had in playing the game, and Section 6.4.4 that provides suggestions from the survey on how the serious games can be improved. Furthermore, Section 6.4.2 aims to answer the research question pertaining to the game's ability to improve grade 9 learners' algebra skills, Section 6.4.2 answers the question of how long players are motivated to play the game, and Section 6.4.3 finally mentions how long it takes for participants to master the serious games.

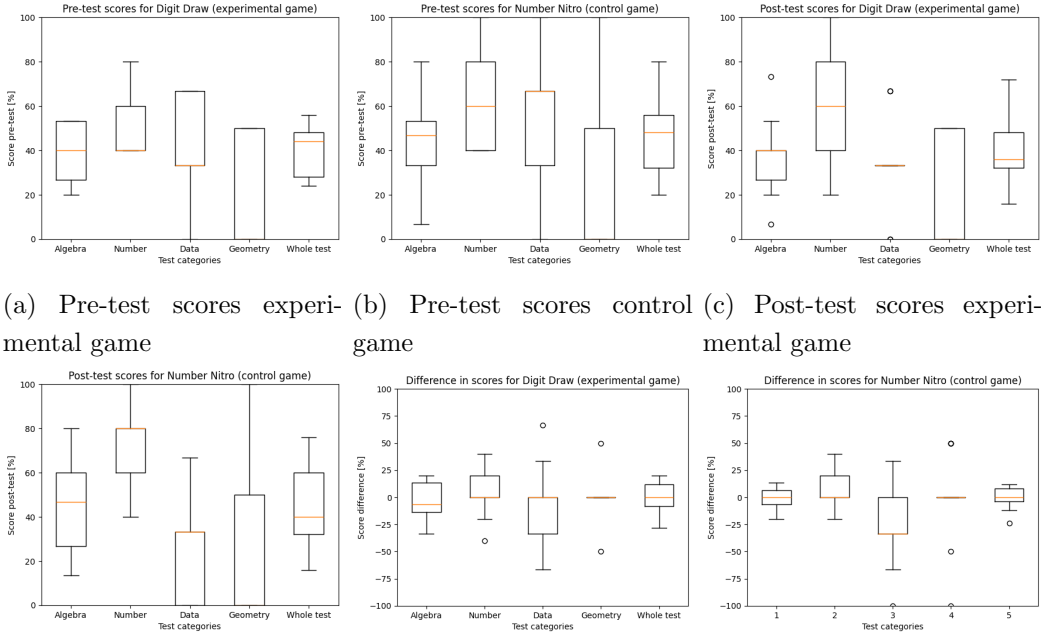
6.4.1 Test scores

Test scores obtained by the participants in the pre-test and post-test are discussed in this subsection. Only the test scores of participants that wrote both the pre-test and post-test are analysed. Using the scores of participants that wrote both tests means that the test scores of participants who only wrote the pre-test are discarded. Therefore, Section 6.2 mentioned that only 4 learners from the Potchefstroom school completed the pre-test and post-test while 28 learners from the Ikageng school completed the pre-test and post-test. Therefore, the sample size of the Potchefstroom school is a bit small for statistical analysis (has only two learners in each group). Statistical analysis is, however, performed on the test scores of the Ikageng school.

First off, outliers were removed from the Ikageng school's data. Two outlier participants were immediately identified in the test scores of the control group. One participant did not answer any question in the pre-test and post-test (obtained zero for both tests), while the other participant stopped with the post-test halfway through (post-test score decreased by 60%). The experimental group showed no obvious outliers.

Box plots are drawn below to inspect outliers that are beyond 1.5 times the interquartile range. Figure 6.3c, Figure 6.3e, and Figure 6.3f show that there are some outliers beyond 1.5 times the interquartile range. However, these outliers were still kept since there are not many samples to work with (only 13 participants per group) and the outliers are not consistently outliers for every test category (a sample is an outlier for one category, but acceptable for all the other categories). A larger sample size could perhaps show that the outliers are not real outliers. The box plots further show that the scores obtained between the experimental and control groups are very similar.

The normality of the data is tested in Table 6.1 by using the Shapiro-Wilk statistical test. The null hypothesis of the test states that the data is normally distributed while the alternative hypothesis concludes that the data is not normally distributed. The red cells of Table 6.1 show the categories that rejected the assumption of normality with a 5% significance level. The Shapiro-Wilk test accurately concludes that the geometry questions are not normally distributed since there were only two geometry questions in the test. Rejecting the normality assumption in the case of two variables means that the geometry questions likely have a binomial distribution.



(a) Pre-test scores experimental game

(b) Pre-test scores control game

(c) Post-test scores experimental game

(d) Post-test scores control game

(e) Score differences experimental game

(f) Score differences control game

Figure 6.3: Box plots of test scores at Ikageng school

However, it comes as a surprise that most of the experimental group's question categories are not normally distributed according to the Shapiro-Wilk test. Unable to ascertain the normality assumption is unfortunate since the planned procedure of using the paired t-test, described in Section 6.3.1, seeks to use normally distributed data. An alternative could be to use a non-parametric test, such as the Wilcoxon rank-sum test [51]. The disadvantage of the Wilcoxon rank-sum test is that it has its own assumption, such as the shape and spread of the compared samples being the same [51]. The similarity assumption of the distribution shape and spread between the experimental group's pre-test and post-test is doubted since Table 6.1 it fails to reject that the post-test of the experimental group is normal, while the normality of the pre-test is rejected.

Therefore, given that the majority of scores in Table 6.1 is normally distributed and that the t-test is relatively insensitive/robust to the normality assumption, the choice has been made to continue with the procedure that was set out in Section 6.3.1 [51]. The effect of this choice is that the hypothesis test may exhibit a larger significance level (such as 7% or 8%) instead of the advertised 5% [51].

Table 6.1: Testing normality of Ikageng tests with the Shapiro–Wilk test (5% significance level)

Test	Category	Experimental game p-value	Control game p-value
Pre-test	Algebra	0.04	0.84
	Number	0.00	0.03
	Data and Chance	0.01	0.08
	Geometry	0.00	0.00
	Whole test	0.11	0.85
Post-test	Algebra	0.73	0.50
	Number	0.32	0.14
	Data and Chance	0.32	0.00
	Geometry	0.00	0.00
	Whole test	0.84	0.69
Difference in test scores (post minus pre scores)	Algebra	0.33	0.21
	Number	0.22	0.04
	Data and Chance	0.36	0.36
	Geometry	0.00	0.01
	Whole test	0.54	0.36

The two-sided two sample paired t-test (Equation 6.2) is used together with a 5% significance level in Table 6.2 to evaluate if there is a difference between the pre-test and post-test scores of the experimental game. The two-sided critical value test-statistic for the t-test is therefore $t_{\alpha/2,n-1} = t_{0.025,12} = 2.179$ [51].

Table 6.2 consequently shows that the null hypothesis was failed to be rejected for all cases since their P-values are larger than 0.05 in all instances. Regrettably, this means that there are no statistically significant differences between the scores of the pre-test and post-test of the experimental game. The low power of the test further indicates that the probability of correctly rejecting the null hypothesis is very small, meaning that it is unlikely that a rejection of the null hypothesis would be correct [51].

Table 6.2: Statistical differences between pre-test and post-test scores of Ikageng experimental group

Category	Mean pre-test score [%]	Mean post-test score [%]	Paired t-Statistic [T_0]	P-value	Power
Algebra	38.46	36.41	-0.44	0.66	0.07
Number	52.31	60.00	1.24	0.24	0.21
Data and Chance	38.46	33.33	-0.52	0.61	0.08
Geometry	15.38	15.38	0.00	1.00	0.05
Whole test	39.38	39.08	-0.07	0.94	0.05

Again, applying the same methods of Table 6.2 to Table 6.3 allows for the statistical comparison of the pre-test and post-test scores of the control group. The majority of Table 6.3 also failed to reject that there are no statistically significant differences between the pre-test and post-test scores of the control group. The only exception is the "Data and Chance" category, which is 28.21% lower in the post-test compared to the pre-test.

However, the "Data and Chance" category represented only three questions in the entire test. Having only three "Data and Chance" questions means that there were not many questions in the test to gain an accurate evaluation of a participant's "Data and Chance" mastery. The "Data and Chance" and "Geometry" categories were only added to monitor if the test was compromised. A compromised test would be indicated by many participants suddenly obtaining full marks in the "Data and Chance" and "Geometry" categories since the games of the experiment did not aim to improve these skills. Luckily, the scores of the categories in Table 6.3 suggest that the test was not compromised. It is hypothesized that the school holiday of 20 September 2024 to 1 October 2024 led to the decline in post-test performance.

Table 6.3: Statistical differences between pre-test and post-test scores of Ikageng control group

Category	Mean pre-test score [%]	Mean post-test score [%]	Paired t-Statistic [T_0]	P-value	Power
Algebra	44.62	44.62	0.00	1.00	0.05
Number	61.54	70.77	1.58	0.14	0.31
Data and Chance	53.85	25.64	-2.86	0.01	0.75
Geometry	23.08	23.08	0.00	1.00	0.05
Whole test	47.38	45.85	-0.55	0.59	0.08

Finally, Table 6.4 uses the independent two-sample t-test of Section 6.3.1 to determine if there are any statistical differences between the test scores obtained by the experimental and control groups. Table 6.4 shows that all categories fail to reject the similarity between the mean score differences of the experimental group and control group on a two-sided 5% significance level. The "Number" category shows an improvement for both the experimental and control groups, although it is statistically insignificant according to Table 6.2 and Table 6.3. The difference in scores being statistically insignificant is likely due to a few participants obtaining worse scores for the number category in the post-test. The tendency for many categories to show a statistically insignificant decline in the post-test scores is likely due to the school holiday that took place between 20 September 2024 and 1 October 2024.

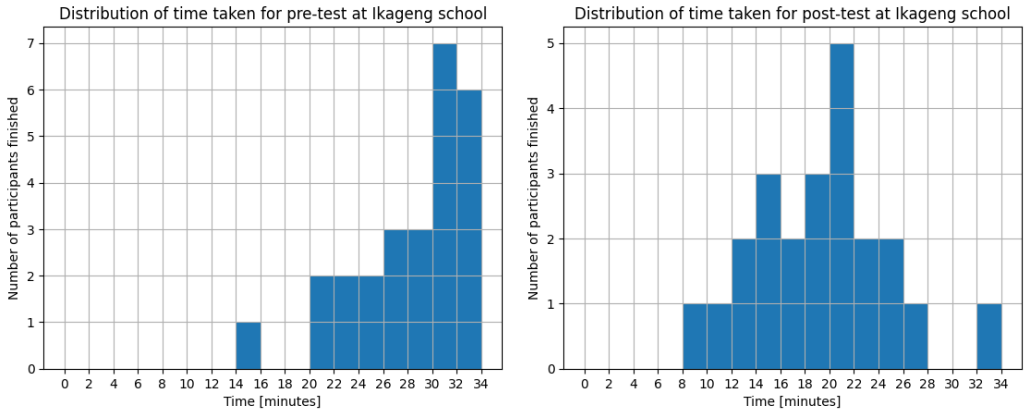
Table 6.4: Statistical differences between Ikageng experimental and control groups

Category	Mean experimental game score difference [%]	Mean Control game score difference [%]	Independent two sample t-statistic [$t_{\alpha/2,v}$]	Absolute critical value t-statistic [T_0^*]	P-value	Power
Algebra	-2.05	0.00	-0.38	2.086	0.71	0.08
Number	7.69	9.23	-0.18	2.069	0.86	0.05
Data and Chance	-5.13	-28.21	1.65	2.069	0.11	0.34
Geometry	0.00	0.00	0.00	2.110	1.00	0.05
Whole test	-0.31	-1.54	0.24	2.086	0.81	0.06

Therefore, the conclusion drawn from Table 6.2, Table 6.3 and Table 6.4 is that there are no significant differences between the scores of the pre-test and post-test for both the control and experimental groups of the Ikagang school. Luckily, these results seem to be trustworthy since the validity of the test is assured by the "Geometry" and "Data and Chance" question categories that do not have substantially higher post-test scores. The normality assumption of the t-test also seems to not have a great influence on the hypothesis test, since most of the p-values of Table 6.2, Table 6.3 and Table 6.4 are fairly high. The only p-value that is somewhat low and of interest is the number category of Table 6.3, although even it requires a significance level of 14% in order to reject the null hypothesis.

The only real difference between the pre-test and post-test results is the amount of time it took for the Ikageng school to complete the tests. Figure 6.4 shows that many of the participants finished the pre-test just before the time was up, but were able to complete the post-test in less time. It could be that the participants became more familiar with the test and the electronic testing platform, but this does not seem to have an effect on the scores obtained in the post-test. Also, the sum of participants in Figure 6.4a correctly adds to 26 while the sum of participants in Figure 6.4b only adds to 23. The difference in participant count is because three participants did not finish the post-test in time. The data of these participants were still kept since their post-test scores indicate that they completed the post-test, but forgot to press the submit button.

The test scores of the four participants of the Potchefstroom school are provided in Table 6.5. This table shows a similar trend as Ikageng, where some participants did better in the post-test and other participants did worse. Likewise, the Potchefstroom participants finished the post-test in less time compared to the pre-test time. It similarly appears that neither the experimental group nor the control group of the Potchefstroom school had any improvements in their mathematical abilities due to the serious games.



(a) Time taken on pre-test

(b) Time taken on post-test

Figure 6.4: Time taken to write tests at the Ikageng school

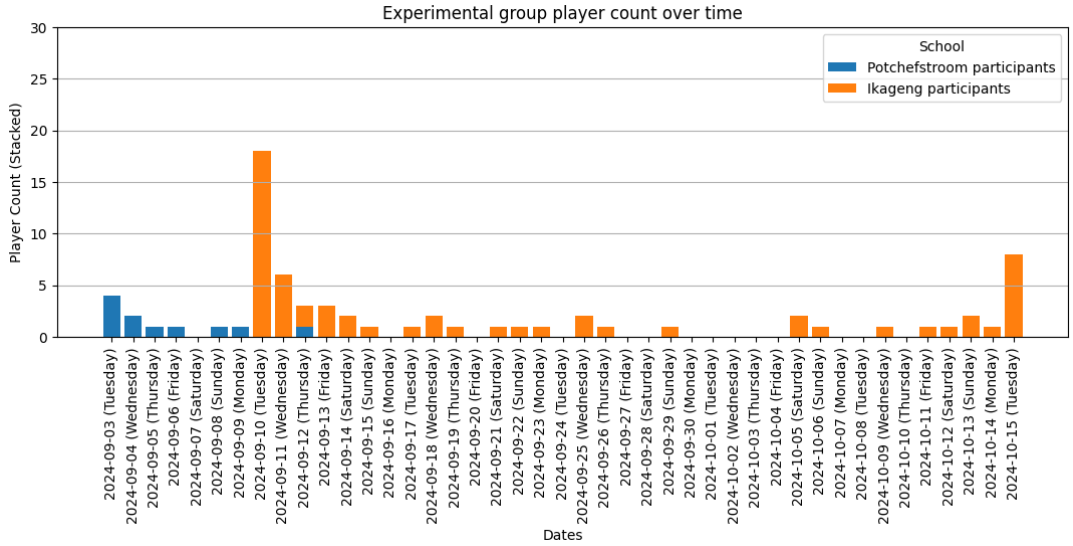
Table 6.5: Test scores of participants from Potchefstroom school

Game group	Player ID	Write time [minutes]		Algebra [%]		Number [%]		Data [%]		Geometry [%]		Whole test [%]	
		Pre	Post	Pre	Post	Pre	Post	Pre	Post	Pre	Post	Pre	Post
Experiment	b854f41502	18.38	16.97	66.67	80.00	100.00	100.00	33.33	66.67	100.00	50.00	72.00	80.00
Experiment	c7e517fcaf	19.03	12.88	86.67	80.00	100.00	80.00	100.00	100.00	100.00	100.00	92.00	84.00
Control	53177eb04b	16.75	10.73	93.33	66.67	80.00	100.00	100.00	33.33	100.00	100.00	92.00	72.00
Control	90a0e1a483	13.05	10.70	73.33	86.67	100.00	100.00	100.00	100.00	100.00	50.00	84.00	88.00

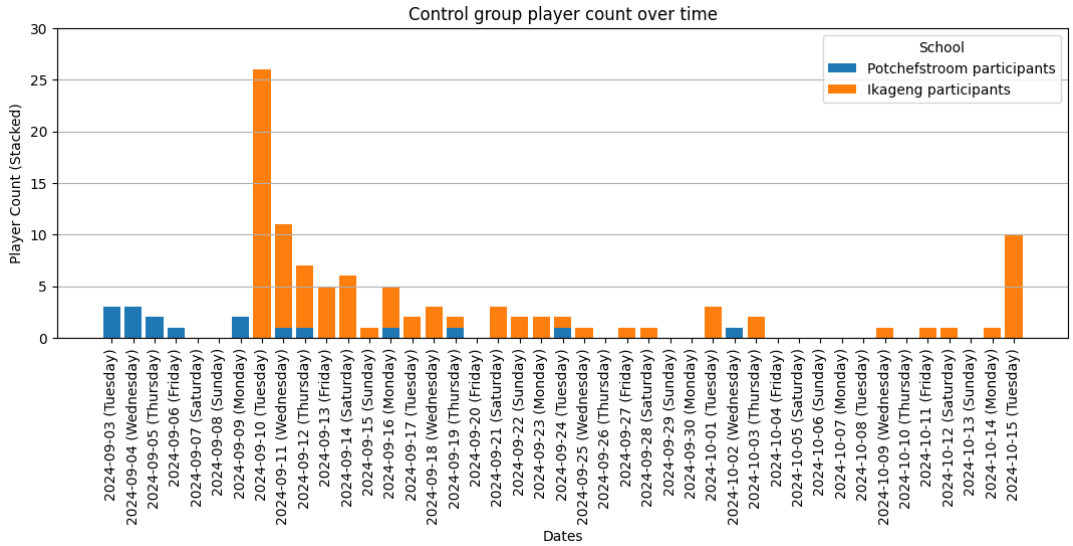
6.4.2 Games play time

This section investigates the times that the participants played their games in order to answer the research question that asks "how long are grade 9 learners motivated to play the serious game?". It should be remembered that significant events affecting these results include the pre-test dates (3 September at Potchefstroom school and 10 September at Ikageng school), the school holiday (20 September - 1 October), and the post-test dates (9 October at Potchefstroom school and 15 October at Ikageng school).

Figure 6.5 illustrates the number of players for both groups at both schools over time. A participant is counted as a player when the participant played at least one game on the specified date. It is interesting to observe that some participants who wrote the pre-test did not play any games on the day of the pre-test. For instance, Section 6.2 mentioned that 28 learners wrote the pre-test and received the experimental game at the Ikageng school, but Figure 6.5a indicates that about 18 of these learners actually played a game on the pre-test date (10 September). It could also be possible that the data was never received from these participants (didn't play games while connected to the internet or didn't participate in the post-test).



(a) Experimental group



(b) Control group

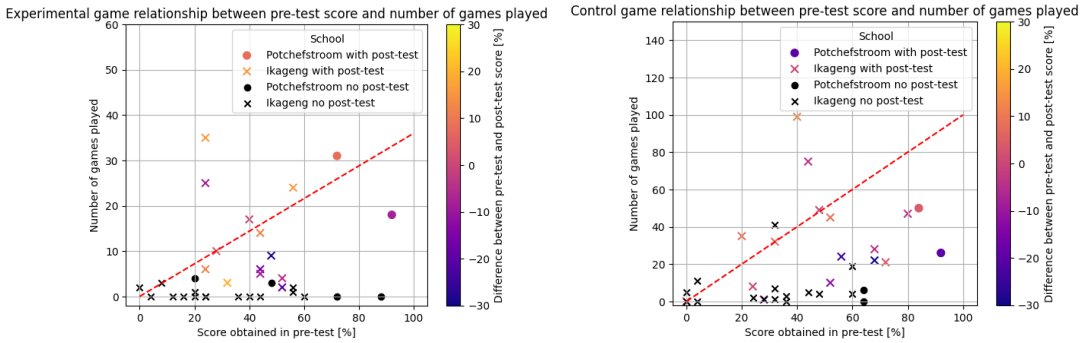
Figure 6.5: Player count over dates

Figure 6.5 further shows that the number of players per day decayed exponentially in all cases. The figure illustrates that the players per day statistic decayed in the majority of cases within 4-5 days after distribution, which also coincides with the first weekend that took place after distribution. The players per day statistic never fully recovered after the decay, but the Ikageng school continuously had one or two players playing in both groups. The experimental group at the Potchefstroom school reached a 100% mortality 10 days after distribution.

Luckily, Figure 6.5 shows that both groups of the Ikageng school had many players that chose to play their game on the 15th of October when we came to administer the post-test.

Therefore, in the case of the Ikageng school, it seems that learners are encouraged to play the games when it is made part of a class/extracurricular activity where there is someone to share the experiences with or to provide some guidance.

Scatter plots presenting the relationship between pre-test scores obtained, number of games played, and the difference between pre-test and post-test scores for participants of each group are given in Figure 6.6. The plots show no real relationship between these variables. Therefore, it appears that there are other factors that influence the number of games played by each participant. However, the plots do show that there are a number of participants in each group that played more than 10 games and that it is not only one or two players that played the games. It can also be seen that most of the participants who did not write the post-test, did not really play the serious games or their gameplay data was never received (when playing without internet and not writing the post-test).



(a) Experimental group (b) Control group

Figure 6.6: Relationship between pre-test score and number of games played

Cumulative plots of the number of games played by each group over time are given in Figure 6.7 and Figure 6.8. Both figures show that the cumulative play time has the shape of a smooth stepped response where the cumulative play time plateaus after a while.

The cumulative plots can also be used to determine how long it took to play 80% of the games by ignoring all games played on the post-test days since they cause a bias in the results. The 80% game mark is indicated in each figure by the dashed line. It is seen from Figure 6.7 that 80% of the games played by the Potchefstroom experimental group occurred within 7 days after receiving the game while the Ikageng experimental group played 80% of the experimental group games within 12 days after receiving the game.

In the case of the control game, about 80% of the games played by the Potchefstroom school occurred within 10 days after receiving the game and 80% of the games played by the control group of Ikageng took place within 9 days of receiving the game. Therefore, the Potchefstroom group was motivated to play 80% of its games for around 8 days after receiving the games while the Ikageng school was motivated to play 80% of its games for

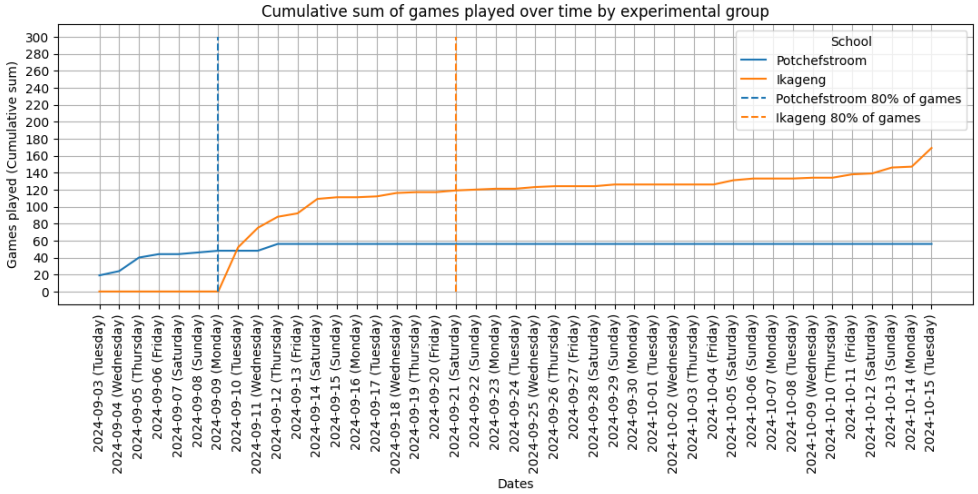


Figure 6.7: Cumulative games played over dates by experimental group

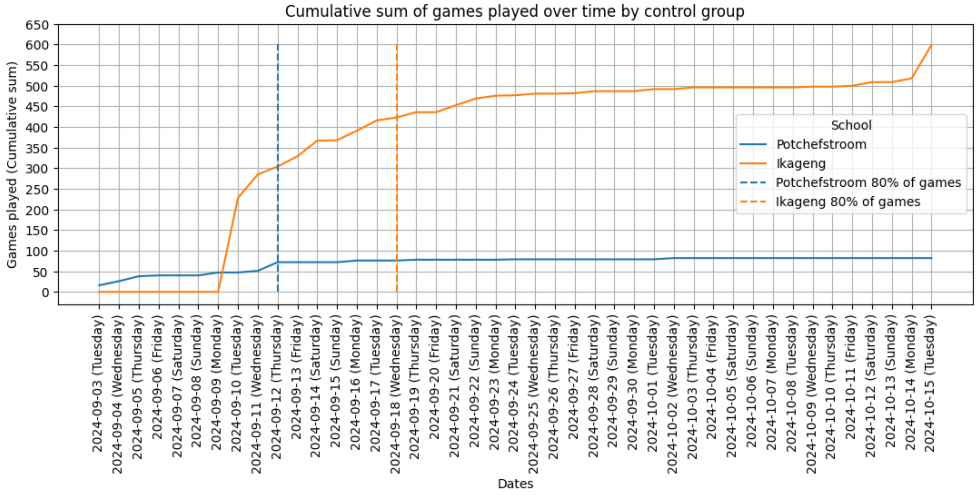


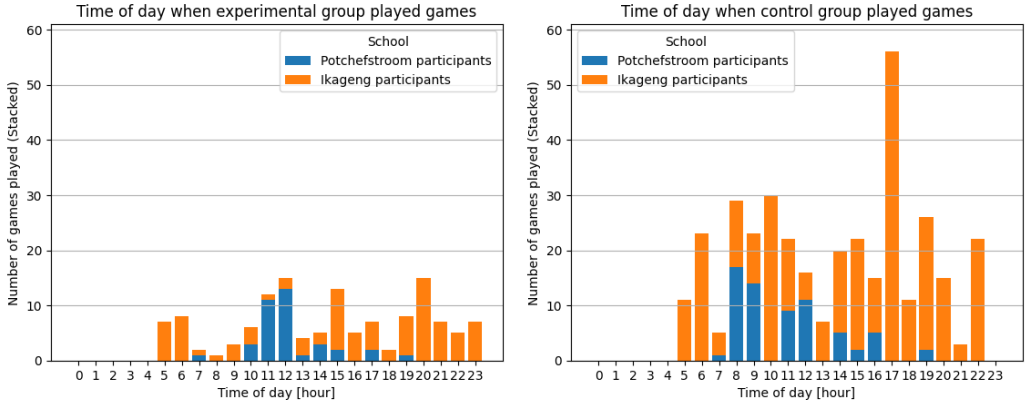
Figure 6.8: Cumulative games played over dates by control group

around 10 days after receiving them. It can be seen that the control group of both schools played many more games than the experimental group.

Figure 6.9 investigates the time of day when the participants played their games. Games played on pre-test and post-test days are not included since they biased the results towards the time when the tests were taken. Interestingly, participants from the Potchefstroom school played their games the most during school hours, while the participants of the Ikageng school played their games any time of the day, from early in the morning to late at night.

Additional insight can be gained into the results of Figure 6.9 by considering question 2.3 of the survey in Appendix G. Question 2.3 asked participants what time of day they played the games and why. None of the participants responded that they play during school

hours, but a few (n = 5) responded that they play in the afternoon when their homework is finished. Other (n = 5) participants responded that they play at night since it is peaceful and there is nobody to disturb them. One participant mentioned that they play in the evening since their mother uses the phone at other times. There are n=9 participants that state that they play when they have free time, which is often random/unpredictable. Two participants (n=2) stated that they play in the morning (one participant said it is due to feeling "fresh" in the morning), another (n=2) stated that they play on weekends (since it is then their free time) and (n=1) said that they play during meals since it is their free time.



(a) Experimental group (b) Control group

Figure 6.9: Time of day when games were played

Therefore, the survey results indicate that some participants and their parents saw the serious games as entertainment games since they could only play the game after their homework was finished. Viewing the game as a form of entertainment is a good thing since it means that the serious games have the appearance of being a fun activity. The game being viewed as a form of entertainment also necessitates a paradigm shift within parents and participants that games can be didactic and contribute to mastery of educational content. It is unfortunate to discover that many of the participants could only play at random times since they already have a busy schedule.

Finally, the effect of sending notifications to participants (see Section 3.6.4) in order to motivate them to play is given in Figure 6.10 and Figure 6.11. Notifications are sent to players on day 0, 2 and 4 after the first notification was fired. Section 3.6.4 shows that notifications had a greater influence in the control game than in the experimental game, and that the Ikageng school was more responsive to the notifications than the Potchefstroom school. It can also be seen that the first notification (day 0) has the largest effect and that subsequent notifications became less effective.

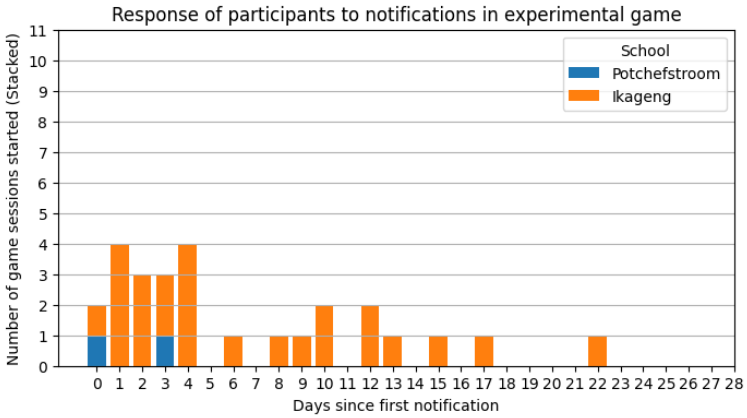


Figure 6.10: Response of experimental group to mobile notifications

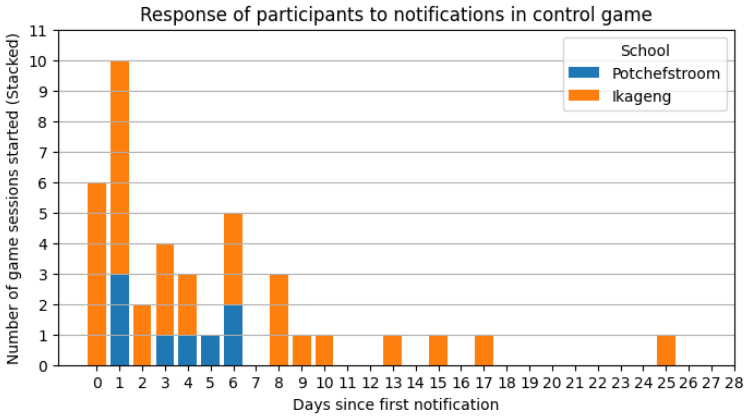


Figure 6.11: Response of control group to mobile notifications

6.4.3 Gameplay difficulty

The difficulty that participants experienced in playing the serious games is assessed in this section. The objective is to answer the research question that asks "how long does it take for grade 9 learners to master the serious game?".

Figure 6.12 and Figure 6.13 show participants' game progression status in the experimental and control games. Figure 6.12 shows the max level reached by the experimental group participants and that the majority of them did not play beyond the first opponent. It should be remembered that the experimental game contains 36 opponents (see Section 3.6.2), which means that the experimental group did not master the game since the majority of players (n=12) got stuck on the first level.

Figure 6.13 conveys the level progression of the control group by stating the amount of players that played a level (not the maximum level reached by a player as with Figure 6.12). It is shown that levels can be a double-edged sword in the sense that the curiosity brought by levels can cause players to play longer, but the players quickly stop playing when a

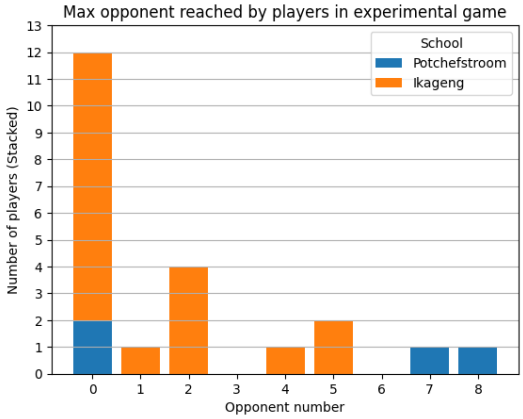


Figure 6.12: Level progress in experimental game

new level is not too different or exciting in comparison with an old level. For instance, Figure 6.13 illustrates that players stopped playing after the second minus level or the boss minus levels since the levels thereafter did not meet the players' expectations (became repetitive or hard/progression blocker).

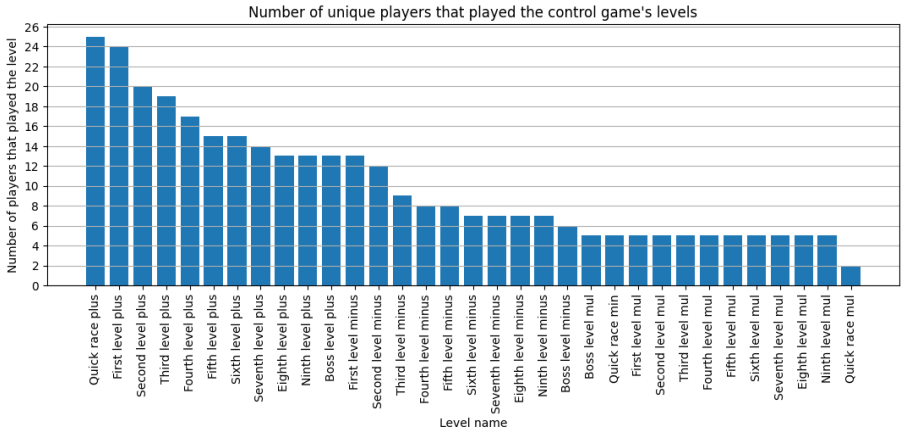


Figure 6.13: Level progress in control game

The difficulty that participants experienced when playing the experimental game is shown in Figure 6.14. It can be seen that many participants struggled to win since they lost most of their games or abandoned them (did not finish). In contrast, Figure 6.15 illustrates that the participants of the control game were able to win most of their games. Participants from the control group also had a substantial number of games that were not finished.

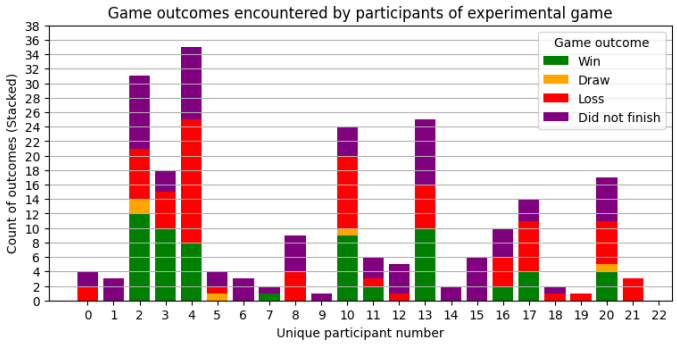


Figure 6.14: Game outcomes in experimental game

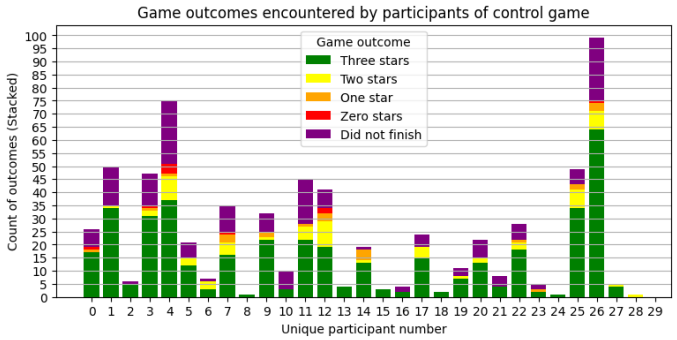


Figure 6.15: Game outcomes in control game

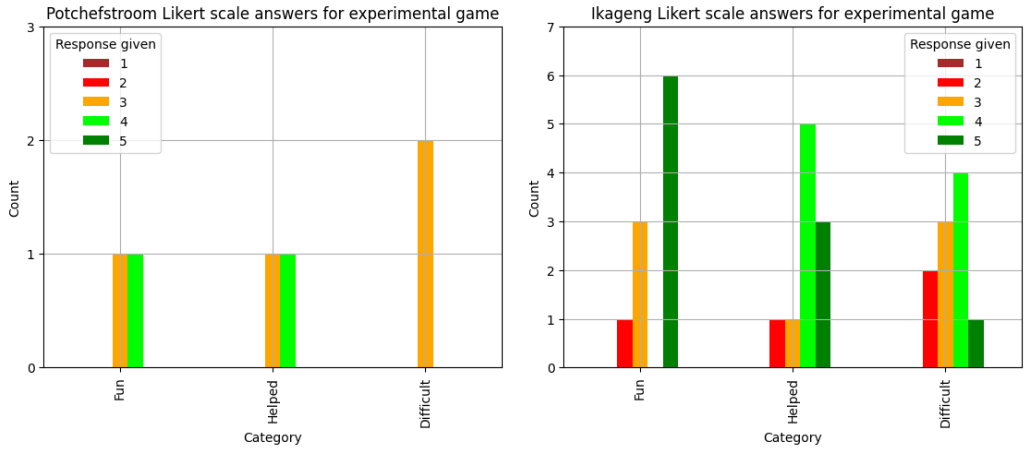
The desirable difficulties implemented in the experimental game as well as the abstract nature of the experimental game can be leading causes for the experimental game to have a much lower win rate compared to the control game. This steep learning curve of the experimental game is precisely the reason why in-game video tutorials were initially developed for the experimental game. Table 6.6 shows that many of the experimental group participants only watched the welcoming tutorial (which covered how to select an opponent), but did not watch the fundamental tutorial that shows how the foundational gameplay works. The win rate/difficulty/learning curve of the game can perhaps be improved by creating in-game tutorials that involve players through active play or by improving the production quality of the video tutorials.

Table 6.6: Experimental game tutorial view statistics

Tutorial description	Number of unique players watched	Number of times watched
TutPlayWelcome	16	35
TutPlayGame Basics	5	10
TutPlayAnimal Rescue	1	1

6.4.4 Participant's feedback for future improvements

This final subsection investigates the feedback given through the survey of Section 6.3.2 in order to determine future improvements that can be made to the experimental game. The experimental group's responses related to the Likert questions of the survey are documented in Figure 6.16. The "category" of Figure 6.16 refers to the different Likert questions asked, namely how fun the game was (score of 5 is plenty of fun), how much the game helped with mathematics (score of 5 helped a lot), and how difficult the game was to play and understand (score of 5 is easy to understand). The categorized bars of the chart indicate the number of participants that gave a particular score for that category.



(a) Potchefstroom school (b) Ikageng school

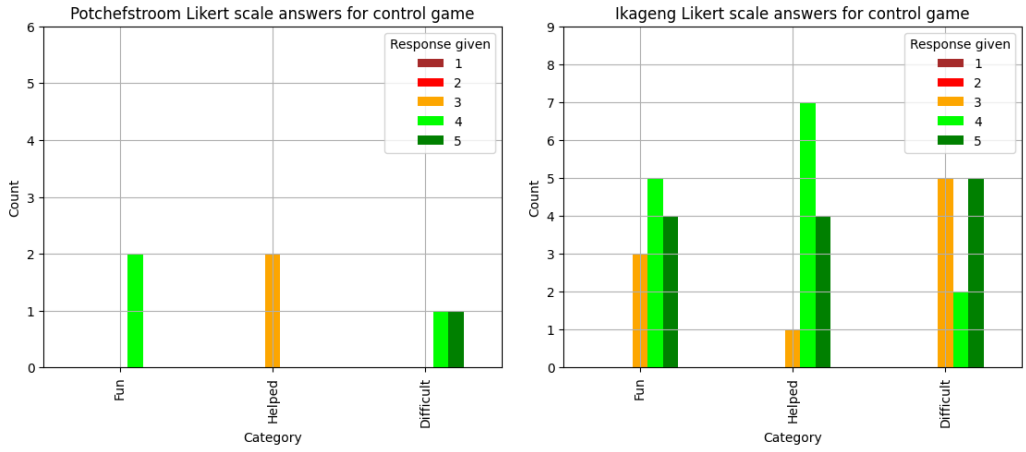
Figure 6.16: Experimental group's Likert responses in the survey

It is observed in Figure 6.16 that the Potchefstroom school found the experimental game moderately fun, while the majority of the Ikageng school stated that the game was very fun to them. This difference in response can justify why the results of Section 6.4.2 indicate that the experimental game was played longer at the Ikageng school when compared to the Potchefstroom school. However, the responses of the Ikageng school can be biased by the observer-expectancy effect since the majority answered that they found the experimental game really fun, but many did not progress past the first opponent/level (see Figure 6.12). It seems unlikely that a participant can enjoy a game, but not progress in it.

The survey responses of Figure 6.16 further show that the learners believed that the game helped them with mathematics. Although Section 6.4.1 indicates that there were no measurable differences in the mathematical abilities of the participants, the participants at least felt that they did mathematics and that there is a possibility of the game helping them. The participants subsequently thought that the game is not too difficult by giving it an average difficulty rating. Striking a balance in the game's difficulty is precisely where flow and desirable difficulties come in (Section 2.2.2 and Section 2.1.2), by making

a game that is not too easy but also not too hard. However, it is still suggested that the game's onboarding/tutorials should be improved so that the win rate of players (shown in Section 6.4.3) can improve. One participant in Figure 6.16b indicated that he didn't enjoy the experimental game and didn't find it helpful.

Figure 6.17 shows the Likert responses of the control game. The responses of the control game are very similar to the experimental game, except for the control group not finding their game as hard as the experimental group found theirs.



(a) Potchefstroom school (b) Ikageng school

Figure 6.17: Control group's Likert responses in the survey

Questions 2.1, 2.2, and 2.4 of the survey were open-ended questions that asked participants what they liked about the game, what they hated about the game, and what improvements they would suggest being made to the game. The majority of the experimental group said that they liked the game due to it helping them with mathematics (n=4), two participants said that they liked mastering the game and the challenging aspect of it, while one participant said that they liked the levels/progression and another said that they liked that the questions don't require the use of a calculator. Participants of the experimental game hated that the game took long to understand (n=5), that the levels and equations are difficult and frustrating (n=3), the feeling of losing for the first time (n=1), and that it was unclear how to rescue animals (n=1). Improvements suggested for the experimental game are to increase the difficulty of opponents (n=4, although participants struggled against the current opponents), adding "solve for x"/algebra questions (is implemented on harder levels that participants didn't play), improving the tutorials/understandability of the game (n=3), seeing maths marks improve (n=2), adding statistics of player performance/progression, playing against friends (multi-player mode), and publishing the game to the Google Play or Apple App Store.

In the case of the control group, the majority (n=7) responded that they liked the control game for improving their mathematics, some responded that they liked that the game did not need a calculator to play (n=3), two participants said that they like challenge/difficulty increases, and one participant said that they like the customization of the car. Participants of the control group hated the hard levels/short time to answer in some levels (n=3), working with large numbers that require a calculator (n=2), progression in the game being blocked by levels (n=2), subsequent levels being mostly the same/monotonous (n=2), the game being easy (n=1), and a bug that makes one fail the race when answering questions too fast (n=1). Improvements suggested by control group participants are to have improved marks in mathematics (n=5), adding a greater variety of questions (exponents, fractions, geometry, algebra) and branching the game out to other subjects, such as chemistry (n=4), increasing the time available to answer questions (n=1), providing extra maths classes/tutor sessions (n=1), fixing a multiplication bug in the game (n=1), and adding more variation between races of different levels (n=1).

Therefore, participants from both groups had a variety of suggestions and advice on what was successful and what failed. The crucial and primary improvement suggested by the participants, as well as the game analytics of Section 6.4.3, is to improve the tutorial/onboarding process of the experimental game. The deficiency of the tutorial videos is also illustrated in Table 6.7 where only four players used/knew about the animal rescue mechanic. Having the animal mechanic largely unused means that the animal rescue mechanic was unable to serve its motivational purpose (discussed in Section 3.6.3). In-game personalization of the control game seemed to interest one participant, but Figure 6.18 suggests that there is not a correlation between personalization mechanic usage and the number of games played in the control group.

Finally, in-game levels are a two-sided coin by providing a natural curriculum and alluring curiosity within the player, but can cause progression blocking or demotivate a participant if there is not enough change between subsequent levels. Multi-player against friends can perhaps help to motivate learners to play by educing positive peer pressure. Figure 6.19 shows that care should be taken to ensure that multi-player is done on a local WiFi/Bluetooth network, since many participants will likely not have mobile data to play online multiplayer. Lastly, the survey responses showed that participants prefer questions that rely on mental arithmetic opposed to using a calculator to solve the problem.

Table 6.7: Experimental game animal rescue statistics

School	Player ID	Rescue date	Rescue chance	Rescue Outcome
Potchefstroom	c7e517fcaf	2024-09-03	0.88	Won
Potchefstroom	c7e517fcaf	2024-09-09	0.6	Won
Ikageng	5f15d87905	2024-09-12	0.83	Won
Ikageng	bfd2bb88f1	2024-09-10	0.15	Lost
Ikageng	bfd2bb88f1	2024-09-10	0.19	Lost
Ikageng	8bb55557e5	2024-10-12	0.17	Lost
Ikageng	8bb55557e5	2024-10-15	0.17	Lost
Ikageng	8bb55557e5	2024-09-10	0.17	Lost
Ikageng	8bb55557e5	2024-09-11	0.09	Lost
Ikageng	8bb55557e5	2024-09-11	0.3	Lost
Ikageng	8bb55557e5	2024-09-11	0.15	Won
Ikageng	8bb55557e5	2024-09-11	0.15	Lost
Ikageng	8bb55557e5	2024-09-11	0.15	Lost
Ikageng	8bb55557e5	2024-10-05	0.17	Lost

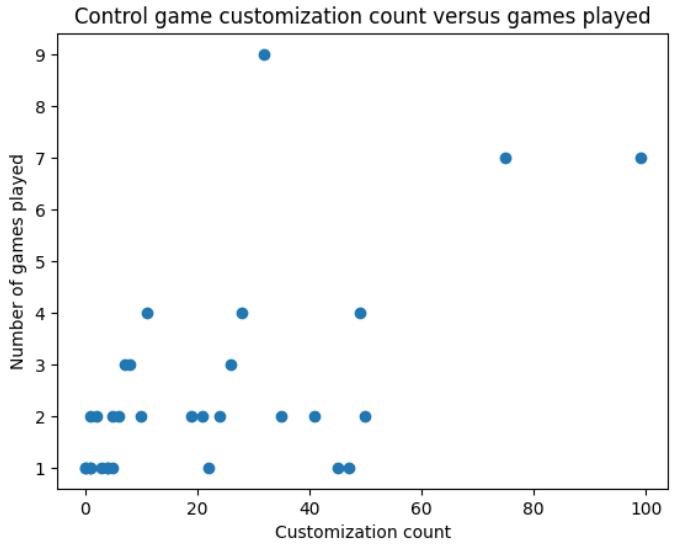
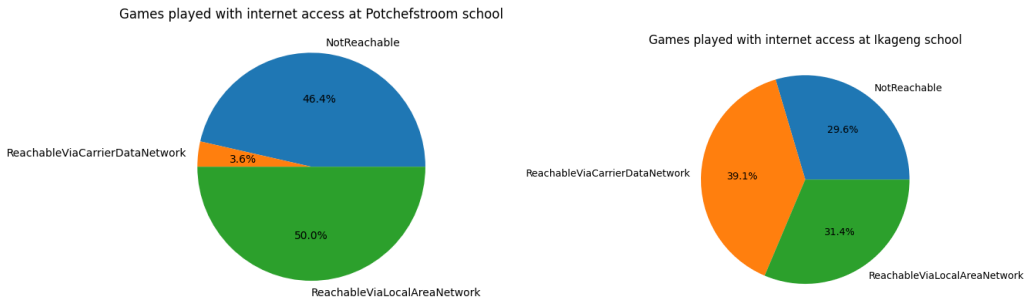


Figure 6.18: Relationship between personalization mechanic and games played of control games



(a) Potchefstroom school (b) Ikageng school

Figure 6.19: Games played with internet access

6.5 Experiment conclusion

This section presented the results of testing the serious game on real grade 9 learners of a Potchefstroom school and an Ikageng school. In particular, the results of this chapter aim to answer the following research questions:

1. Does the serious game improve the algebraic mathematical abilities of grade 9 learners?
2. How long are grade 9 learners motivated to play the serious game?
3. How long does it take for grade 9 learners to master the serious game?

The first research question is answered by means of an adapted TIMSS test that is used for the pre-test and post-test together with an experimental and control group. The results of the tests indicate that the serious game had no effect on the algebra mathematical abilities of the participants. The method of testing seems valid since the non-algebra questions also didn't show any statistically significant increase in post-test scores. An improvement to the adapted TIMSS test would be to ensure that the validation question categories (like geometry) contain at least 5 questions in order to increase the likelihood of the validation questions also being normally distributed. Some test categories showed a statistically insignificant decrease in post-test scores. The insignificant decrease in post-test scores is likely due to the school holiday that took place before the post-test. The school holiday could have caused participants to become out of practice in these categories.

The second research question is answered by observing the game interactions of learners over time. It was observed at both schools that game interactions exponentially decrease after the game is provided to participants. Generally, the majority of participants had interest in the control and experimental games for about a week. Limited gameplay activity was observed after the one-week period. It is informative to observe that the Potchefstroom school mainly played the games during school hours.

The third research question is answered by investigating the maximum game level that was reached by participants. Unfortunately, many participants did not play past the first opponent, and no participant finished the game. The maximum opponent reached by a participant is opponent number 8, while the game has a total of 36 opponents. Therefore, players were unable to master the serious game. It is anticipated that improving the game's tutorials can aid in reducing the game's learning curve, which can also potentially lead to increased player mastery of the game.

However, some participants were able to finish all the levels of the control game. Gameplay data from the control game suggests that game levels are a double-edged sword, since they can pique a player's curiosity but can also demotivate players when subsequent levels

change too slowly. It is therefore possible that the levels of the experimental game changed too slowly, making players think that the experimental game doesn't contain algebra questions.

Lastly, it appears that the survey administered to participants may be biased by the observer-expectancy effect, even though participants were told that they can be critical of the game. Participants indicated in the survey that they enjoyed the game and that they think that the game helped them with mathematics. Unfortunately, this is in contradiction with the quantitative data of the experiment that suggests that the game had no effect on the participants' mathematical abilities and that many participants stopped playing after a week.

Chapter 7

Conclusion

The serious game developed in this study is a two-player zero-sum adversarial card game named *Digit Draw*. The characterizing goal of *Digit Draw* is to aid grade 9 learners in gaining an intuition and motivation for algebraic mathematics. The game's characterizing goal is implemented by playing mathematical cards against an AI opponent that was trained with a machine learning technique known as reinforcement learning.

Subsequently, the study evaluated *Digit Draw* by answering the following research questions:

1. Which reinforcement learning algorithm is more applicable to the serious game?
2. Does the serious game improve the algebraic mathematical abilities of grade 9 learners?
3. How long are grade 9 learners motivated to play the serious game?
4. How long does it take for grade 9 learners to master the serious game?

The first research question compared the SAC and PPO RL algorithms by running training simulations and calculating the ELO scores of the agents. It is found that PPO trained faster on the system of Appendix L and that PPO obtained a higher ELO score than SAC. The results of the ELO scores were verified and confirmed by means of a tournament. It is found that the randomness of the card game was detrimental to the SAC algorithm's performance since SAC favours high entropy actions. The SAC algorithm spent many of its training steps satisfying its curiosity instead of learning the game.

For future work, it can be interesting to evaluate if a value based method (such as Deep Q-learning with a CNN) can perform better than PPO. The merit of this investigation is that certain RL tasks may be better represented with value methods compared to policy

gradient methods [7]. Moreover, the RL agent designed in Chapter 4 can be used for other applications that need to make decisions based on the values of a mathematical function. An example that comes to mind is the implementation of an agent that has a set of basis functions (represented by the cards of the game) that needs to use the basis functions to approximate/fit a given mathematical function as closely as possible. This example is similar to how the Fourier series uses trigonometric functions as its basis functions in order to approximate another function.

The other three research questions considered the response of real grade 9 learners to the serious game. These questions were answered by doing a field study at a Potchefstroom school and an Ikageng school. The field study used an experimental and control group. The game of the control group is given in Appendix E.

The second research question that asks if the game "improves the algebraic mathematical abilities of grade 9 learners" was answered by means of a pre-test and post-test. An adapted TIMSS test, given in Appendix F, was used for the pre-test and post-test. The pre-test and post-test concluded that neither *Digit Draw* nor the control game improved the algebraic mathematical abilities of grade 9 learners at the Potchefstroom and Ikageng schools.

Using an adapted TIMSS test in the field study was an appropriate research method for this study since the validation questions of the adapted TIMSS tests showed that the post-test was not compromised during the field study. However, participatory action research (PAR) can possibly be an alternative research methodology to consider for future research that is similar to this study. PAR differs by actively involving research participants throughout the game design and designing the game specifically for the research participants [82]. The benefits of PAR are an increased chance of the game being adopted by research participants, and design issues are detected as early as possible in the game's design. The weakness of PAR is that the methodology is effortful to conduct (more visits need to be made to the schools) and the study's results are less generalizable [82].

The third research question enquiring "how long are grade 9 learners motivated to play the serious game" was answered by using passively captured gameplay analytics. The analytics showed that player engagement decayed exponentially after the game was distributed, and that most of the players stopped playing after approximately one week. A limited number of participants played the game seldomly after a week.

The Ikageng school showed increased game engagement when the post-test was conducted. Furthermore, the Potchefstroom school mostly played the game during school hours. These play trends identified from the data suggest that the game might achieve improved engagement statistics if the game is used as a class activity or an extracurricular activity that occurs at least once a week. Also, player engagement can possibly be improved by adding multiplayer against human players and relying on positive peer pressure among

learners to motivate each other to play. Enabling multiplayer against human players will create a unique experience of playing against friends at school that cannot be obtained by playing games over the internet or watching TV. Care should just be taken to make sure that the multiplayer of the game is done over local Wi-Fi or Bluetooth since many participants of the experiment played without an internet connection.

Finally, the fourth research question asks "how long does it take for grade 9 learners to master the serious game". The dire conclusion is that participants didn't master the game since no participant managed to play all of the game's levels. Furthermore, many participants were stuck on the first level of the game. The mediocre tutorials of the game are believed to be the leading cause for participants not mastering the game. A future improvement can be to improve the production quality of the tutorials by making the tutorials an interactive game level instead of a video. Adding local multi-player against human players can also serve as a peer-based teaching method to supplement the tutorials.

Survey results from the Ikageng school showed that some of the learners *want* to improve in their academics. Therefore, it is not only the DBE that desires an improvement in school performance, but also the learners that seize every opportunity that is given to them. The serious game itself doesn't necessarily need to be the instrument that directly provides this improvement. The game can alternatively serve as a fun assessment for students to do. Teachers can then investigate the passively collected gameplay data after game sessions to determine what students are struggling with which concepts. This use case is similar to the just-in-time teaching method of Section 2.1.2.

Lastly, the field study showed that the mobile technology is available for educational serious games to be implemented in a South African context. This accessibility to smart mobile technology in South Africa was not the case 15+ years ago [3]. It is anticipated that South Africa's technology adoption will only improve as the DBE advances in its information communication and technology strategies [2]. It can be argued that the future limiting factor for serious game adoption is not to obtain the hardware for the game to run on, but rather to have free (as in libre) hardware and software platforms with no restrictions imposed on the developers or the users.

For instance, some participants could not play the game due to the restrictions imposed by family link (parents tried to provide permission but couldn't) or iPhones not allowing side-loaded apps. The same concept occurs with development programs, such as the Unity game engine, where the owner of the proprietary program has complete say over how the program must be used and what one is allowed to do with it. It is therefore of paramount importance to ensure that future hardware and software platforms that are used by researchers and bought by consumers, allow users to remain the owners of their hardware/software and allow users to modify the platforms as they wish.

List of References

[1] South Africa Department of Basic Education. *2023 NSC Exam Results: School Subject Report*. South Africa Department of Basic Education. URL: https://www.education.gov.za/Portals/0/Documents/Reports/School%20Subject%20Report%202024%20Final%20(1).pdf?ver=2024-01-25-112118-313 (visited on 2024-11-15).

[2] South Africa Department of Basic Education. *Research Agenda 2019-2023*. South Africa Department of Basic Education. July 31, 2019. URL: https://www.education.gov.za/Portals/0/Documents/Publications/Research%20repository/Research%20Agenda_V1.pdf?ver=2019-07-31-135117-690 (visited on 2023-04-19).

[3] G. Spencer-Smith. "A mixed methods investigation into the impact of computers and maths software on mathematics teaching and matric results of high schools in the EMDC East, Cape Town". In: *University of Cape Town* 1 (2010). URL: https://open.uct.ac.za/handle/11427/14754 (visited on 2023-03-10).

[4] A. Kramarzewski and E. De Nucci. *Practical Game Design : Learn the Art of Game Design Through Applicable Skills and Cutting-edge Insights*. Packt Publishing, 2018. ISBN: 9781787121799.

[5] R. Dörner, S. Göbel, W. Effelsberg, and J. Wiemeyer. *Serious Games : Foundations, Concepts and Practice*. Vol. 1. Springer, 2016. ISBN: 9783319406114.

[6] R.S. Sutton and A.G. Barto. *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018. ISBN: 9780262352703. URL: https://books.google.co.za/books?id=uWV0DwAAQBAJ.

[7] P. Winder. *Reinforcement Learning: Industrial Applications of Intelligent Agents*. First edition. Sebastopol, CA: O'Reilly, 2023. ISBN: 9781098114831.

[8] E. Komala, D. Suryadi, and D. Dasari. "Development of Mathematics Didactic Design with Integration of Peer Instruction Regarding Representational Ability of High School Students." In: *JTAM (Jurnal Teori dan Aplikasi Matematika)* 5.2 (2021), pp. 479–487. ISSN: 2597-7512.

[9] DeWolfe and E. Thomas. "Jean Piaget's theory of cognitive development". In: *Salem Press Encyclopedia of Health* (2023).

[10] J. Piaget and B. Inhelder. *The psychology of the child.* Routledge & K. Paul, 1969. ISBN: 0710067518.

[11] A. Kozulin and B.Z. Presseisen. "Mediated Learning Experience and Psychological Tools: Vygotsky's and Feuerstein's Perspectives in a Study of Student; Learning." In: *Educational Psychologist* 30.2 (1995), pp. 67–75.

[12] S. Knight, S. B. Shum, and K. Littleton. "Epistemology, Assessment, Pedagogy: Where Learning Meets Analytics in the Middle Space." In: *Journal of Learning Analytics* 1.2 (2014).

[13] L. Roussouw, S. Rhodes, and I. Christiansen. "Teachers' views on mathematics, mathematics teaching and their existing practices". In: *Primary Mathematics Project, University of the Western Cape.* 1 (1998). URL: https://www.education.gov.za/Portals/0/Documents/Reports/Research%20Repository/Curriculum/Teachers%20views%20on%20maths%20teaching%20and%20practises.pdf?ver=2019-09-09-105511-793&timestamp=1570177024299 (visited on 2023-03-10).

[14] Republic of South Africa department of basic education. "Curriculum and Assessment Policy Statement GRADES 4-6. Mathematics." In: *Department of Basic Education* (2011). URL: https://www.education.gov.za/Portals/0/CD/National%20Curriculum%20Statements%20and%20Vocational/CAPS%20IP%20%20MATHEMATICS%20GR%204-6%20web.pdf?ver=2015-01-27-161430-553 (visited on 2023-03-10).

[15] P. Human et al. *Mathematics - Grade 9 - Book 1 - CAPS - Learner Book.* First edition. Rietondale, South Africa, 2014. ISBN: 978-1-920705-28-2. URL: https://www.siyavula.com/book-downloads/maths/Gr9A_Mathematics_Learner_Eng.pdf (visited on 2024-07-19).

[16] P. Human et al. *Mathematics - Grade 9 - Book 2 - CAPS - Learner Book.* First edition. Rietondale, South Africa, 2014. ISBN: 978-1-920705-29-9. URL: https://www.siyavula.com/book-downloads/maths/Gr9B_Mathematics_Learner_Eng.pdf (visited on 2024-07-19).

[17] Siyavula and Volunteers. *Everything Maths - Grade 10 Mathematics.* Version 1.1. URL: https://www.siyavula.com/book-downloads/maths/Gr10_Mathematics_Learner_Eng_v11.pdf (visited on 2024-07-19).

[18] Siyavula and Volunteers. *Everything Maths - Grade 11 Mathematics.* First edition. URL: https://www.siyavula.com/book-downloads/maths/Gr11_Mathematics_Learner_Eng.pdf (visited on 2024-07-19).

[19] Siyavula and Volunteers. *Everything Maths - Grade 12 Mathematics.* First edition. URL: https://www.siyavula.com/book-downloads/maths/Gr12_Mathematics_Learner_Eng.pdf (visited on 2024-07-19).

[20] Z. Kovacheva, K. Kaloyanova, I. Naydenova, and E. Saranova. "Effective Methods for Teaching Mathematics and Informatics in Higher Education in the Digital World." In: *TEM Journal* 11.2 (2022), pp. 876–881. ISSN: 22178333.

[21] J.W.A. Young. "THE TEACHING OF MATHEMATICS". In: *The Mathematics Teacher* 61.3 (1968), pp. 287–295. ISSN: 00255769. URL: http://www.jstor.org/stable/27957829 (visited on 2023-03-11).

[22] E.L. Castellanos-Leal et al. "Implementation of Just in Time Teaching methodology in mathematics and natural sciences study in high school institutions of Colombia northeast". In: *Journal of Physics: Conference Series* 1161.1 (Jan. 2019). DOI: 10.1088/1742-6596/1161/1/012016. URL: https://dx.doi.org/10.1088/1742-6596/1161/1/012016 (visited on 2023-03-11).

[23] A.J. Bishop. *Mathematical enculturation : a cultural perspective on mathematics education.* Mathematics Education Library: Volume 6. Kluwer Academic Publishers, 1991. ISBN: 9789400926578.

[24] L. Pepkolaj, S. Duraj, V. Toma, and D. Gerbeti. "A collaborative learning experience in high school mathematics." In: *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)* (2020), pp. 1–6. ISSN: 978-1-7281-9352-6.

[25] R.E. Mayer. "Cognitive Learning". In: *Encyclopedia of the Sciences of Learning.* Ed. by Norbert M. Seel. Boston, MA: Springer US, 2012, pp. 594–596. ISBN: 978-1-4419-1428-6. DOI: 10.1007/978-1-4419-1428-6_390. URL: https://doi.org/10.1007/978-1-4419-1428-6_390 (visited on 2023-03-10).

[26] E. Bjork and R. Bjork. "Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning". In: *Psychology and the Real World: Essays Illustrating Fundamental Contributions to Society* (Jan. 2011), pp. 56–64.

[27] Z.R. Mevarech and S. Ben-Artzi. "Effects of CAI with Fixed and Adaptive Feedback on Children's Mathematics Anxiety and Achievement." In: *The Journal of Experimental Education* 56.1 (1987), pp. 42–46. ISSN: 00220973.

[28] E. Tatar. "The relationship between mathematics anxiety and learning styles of high school students." In: *New Educational Review* 28.2 (2012), pp. 94–101. ISSN: 17326729.

[29] H. Pashler, M. McDaniel, D. Rohrer, and R. Bjork. "Learning Styles: Concepts and Evidence." In: *Psychological Science in the Public Interest* 9.3 (2008), pp. 105–119. ISSN: 15291006.

[30] Princeton University. *How People Learn: Common Beliefs Vs. Research.* Princeton University. URL: https://mcgraw.princeton.edu/undergraduates/resources/resource-library/how-people-learn (visited on 2023-03-12).

[31] C. Diemand-Yauman, D.M. Oppenheimer, and E.B. Vaughan. "Fortune favors the (): Effects of disfluency on educational outcomes". In: *Cognition* 118.1 (2011), pp. 111–115. ISSN: 0010-0277. DOI: https://doi.org/10.1016/j.cognition.2010.09.012. URL: https://www.sciencedirect.com/science/article/pii/S001002771000226X (visited on 2023-03-12).

[32] T. Feuerstein. "The theory of structural cognitive modifiability". In: 68 (1990), pp. 109–124.

[33] L.C. Moll. *Vygotsky and Education: Instructional Implications and Applications of Sociohistorical Psychology*. Cambridge University Press, 1990. DOI: 10.1017/CBO9781139173674.

[34] D. Wood, J.S.Bruner, and G.Ross. "THE ROLE OF TUTORING IN PROBLEM SOLVING". In: *Journal of Child Psychology and Psychiatry* 17.2 (1976), pp. 89–100. DOI: https://doi.org/10.1111/j.1469-7610.1976.tb00381.x. eprint: https://acamh.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-7610.1976.tb00381.x. URL: https://acamh.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-7610.1976.tb00381.x (visited on 2023-03-13).

[35] South Africa Department of Basic Education. *Mathematics examination guidelines: Grade 12 2021*. South Africa Department of Basic Education. May 17, 2021. URL: https://www.education.gov.za/Portals/0/CD/2021%20Exam%20Guidelines/Mathematics%20GR%2012%20Exam%20Guidelines%202021%20Eng.pdf?ver=2021-05-17-160015-740 (visited on 2023-04-20).

[36] P.H. Winne and D.L. Butler. "Student cognition in learning from teaching". In: *International encyclopedia of education* 2 (1994), pp. 5738–5775.

[37] E.R. Fyfe and B. Rittle-Johnson. "Mathematics practice without feedback: A desirable difficulty in a classroom setting". In: *Instructional Science* 45 (2017), pp. 177–194.

[38] M.I. Núñez-Peña, R. Bono, and M. Suárez-Pellicioni. "Feedback on students' performance: A possible way of reducing the negative effect of math anxiety in higher education." In: *International Journal of Educational Research* 70 (2015), pp. 80–87. ISSN: 0883-0355.

[39] D. Parlett. *card game*. Britannica. July 12, 2024. URL: https://www.britannica.com/topic/card-game (visited on 2024-07-25).

[40] W.S. Ravyse, A.S. Blignaut, V. Leendertz, and A. Woolner. "Success factors for serious games to enhance learning: a systematic review." In: *Virtual Reality* 21.1 (2017), pp. 31–58.

[41] W. Ortiz, D. Castillo, and L. Wong. "Mobile Application: A Serious Game Based in Gamification for Learning Mathematics in High School Students." In: *2022 31st Conference of Open Innovations Association (FRUCT)* (2022), pp. 1–9. ISSN: 978-952-69244-7-2.

[42] O. Dele-Ajayi, J. Sanderson, R. Strachan, and A. Pickard. "Learning mathematics through serious games: An engagement framework." In: *2016 IEEE Frontiers in Education Conference (FIE), Frontiers in Education Conference (FIE), 2016 IEEE* (2016), pp. 1–5. ISSN: 978-1-5090-1790-4.

[43] I.C. Rønningen, E. årsand, and G. Hartvigsen. "Exploring In-Game Reward Mechanisms in Diaquarium - A Serious Game for Children with Type 1 Diabetes." In: *IWBBIO (2)* (2018), pp. 443–455.

[44] R. Koster. *A Theory of Fun for Game Design*. Vol. 2. O'Reilly, 2014. ISBN: 9781449363215.

[45] G.S. Elias, R. Garfield, and K.R. Gutschera. *Characteristics of Games*. Vol. 1. The MIT Press, 2012. ISBN: 978-0-262-01713-8.

[46] D.W. Sandlin and R.P. Bouchard. *Introducing: FISH GAME - Smarter Every Day 291*. Smarter Every Day. Oct. 29, 2023. URL: https://youtu.be/prRTWThrotU?t=327 (visited on 2024-08-02).

[47] V. Rossano, F. Mangialardo, and T. Roselli. *Math is magic: an adaptive serious game to reinforce math competences*. University of Bari. Sept. 21, 2021. URL: https://www.youtube.com/watch?v=Q2k25-QHqhE (visited on 2023-04-07).

[48] G.G. Barbieri, R. Barbieri, and R. Capone. "Serious Games in High School Mathematics Lessons: An Embedded Case Study in Europe". In: *Eurasia Journal of Mathematics, Science and Technology Education* 17.5 (Apr. 2021). ISSN: 1305-8215, 1305-8223. DOI: 10.29333/ejmste/10857. URL: https://www.ejmste.com/article/serious-games-in-high-school-mathematics-lessons-an-embedded-case-study-in-europe-10857 (visited on 2023-04-17).

[49] M.D. Davis and S.J. Brams. *game theory*. Britannica. July 31, 2024. URL: https://www.britannica.com/science/game-theory (visited on 2024-08-18).

[50] T. Simonini. *ELO Rating System*. Unity-Technologies. June 6, 2023. URL: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/ELO-Rating-System.md (visited on 2024-08-18).

[51] D.C. Montgomery and G.C. Runger. *Applied statistics and probability for engineers*. Seventh edition. Hoboken, NJ, 2018. ISBN: 9781119746355, 1119746353, 9781119585596, 1119585597. URL: https://www.worldcat.org/oclc/1004424860 (visited on 2023-04-23).

[52] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: (2018). arXiv: 1801.01290 [cs.LG]. URL: https://arxiv.org/abs/1801.01290.

[53] A. Zhang, Z.C. Lipton, M. Li, and A.J. Smola. *Dive into Deep Learning*. 2023. arXiv: 2106.11342 [cs.LG]. URL: https://arxiv.org/abs/2106.11342.

[54] V. Mnih et al. "Human-level control through deep reinforcement learning". en. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14236. URL: https://www.nature.com/articles/nature14236 (visited on 2023-07-02).

[55] D. Silver et al. "Mastering the game of Go without human knowledge". en. In: *Nature* 550.7676 (Oct. 2017), pp. 354–359. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature24270. URL: https://www.nature.com/articles/nature24270 (visited on 2023-07-02).

[56] OpenAI et al. "Dota 2 with Large Scale Deep Reinforcement Learning". In: (2019). arXiv: 1912.06680. URL: https://arxiv.org/abs/1912.06680.

[57] T. Bansal et al. "Emergent complexity via multi-agent competition". In: *arXiv preprint arXiv:1710.03748* (2017).

[58] Unity ML-Agents. *Training Configuration File*. Unity-Technologies. Mar. 6, 2024. URL: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Training-Configuration-File.md (visited on 2024-10-13).

[59] Michael L Littman. "Markov games as a framework for multi-agent reinforcement learning". In: *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[60] Aswin. *Why The Retirement Of Lee Se-Dol, Former 'Go' Champion, Is A Sign Of Things To Come*. en. URL: https://www.forbes.com/sites/aswinpranam/2019/11/29/why-the-retirement-of-lee-se-dol-former-go-champion-is-a-sign-of-things-to-come/ (visited on 2023-07-02).

[61] *OpenAI Five*. en-US. URL: https://openai.com/research/openai-five#rapid (visited on 2023-07-02).

[62] *HOW TO PLAY MAGIC: THE GATHERING*. Wizards of the coast. URL: https://magic.wizards.com/en/how-to-play (visited on 2024-08-20).

[63] *Yu-Gi-Oh! Official Rulebook*. Konami. URL: https://img.yugioh-card.com/en/downloads/rulebook/SD_RuleBook_EN_10.pdf (visited on 2024-08-20).

[64] *Color Blindness*. National Eye Institute. Aug. 5, 2014. URL: https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/color-blindness (visited on 2024-08-12).

[65] M. Rosewater. *Lenticular Design*. Magic the Gathering. Dec. 15, 2014. URL: https://magic.wizards.com/en/news/making-magic/lenticular-design-2014-12-15 (visited on 2024-08-11).

[66] L. Von Ahn. *How to Make Learning as Addictive as Social Media*. TED. Oct. 26, 2023. URL: https://www.youtube.com/watch?v=P6FORpg0KVo (visited on 2024-08-18).

[67] K.P. Yancey and B. Settles. "A Sleeping, Recovering Bandit Algorithm for Optimizing Recurring Notifications". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, 3008–3016. ISBN: 9781450379984. DOI: 10.1145/3394486.3403351. URL: https://doi.org/10.1145/3394486.3403351.

[68] K. MacDonald. *Balatro: this fiendish mashup of solitaire and poker has taken over my life*. The Guardian. Feb. 27, 2024. URL: https://www.theguardian.com/games/2024/feb/27/balatro-mashup-solitaire-poker-has-taken-over-my-life-roguelike (visited on 2024-08-21).

[69] *Balatro*. Steam. URL: https://store.steampowered.com/app/2379780/Balatro/ (visited on 2024-08-21).

[70] Unity ML-Agents. *Learning Environment Design Agents*. Unity-Technologies. Oct. 5, 2024. URL: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Learning-Environment-Design-Agents.md (visited on 2024-10-23).

[71] D. Giannoulis, M. Massberg, and J.D. Reiss. "Digital dynamic range compressor design—A tutorial and analysis". In: *Journal of the Audio Engineering Society* 60.6 (2012), pp. 399–408.

[72] S.K. Mitra. *Digital signal processing: a computer-based approach*. McGraw-Hill Higher Education, 2001.

[73] J.D. Faires and R. Burden. *Numerical Methods*. Fourth edition. Cengage learning, 2013. ISBN: 978-0-495-38569-1.

[74] J. Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[75] T. Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. 2019. arXiv: 1812.05905 [cs.LG]. URL: https://arxiv.org/abs/1812.05905.

[76] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. "Algorithms for hyper-parameter optimization". In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. Granada, Spain: Curran Associates Inc., 2011, 2546–2554. ISBN: 9781618395993.

[77] E. Brochu, V.M. Cora, and N. de Freitas. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: (2010). arXiv: 1012.2599 [cs.LG]. URL: https://arxiv.org/abs/1012.2599.

[78] Optuna Contributors. *optuna.samplers*. Optuna. 2018. URL: https://optuna.readthedocs.io/en/stable/reference/samplers/index.html (visited on 2024-11-12).

[79] F. Hutter, H. Hoos, and K. Leyton-Brown. "An efficient approach for assessing hyperparameter importance". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, 754–762.

[80] G.C. Linderman et al. "Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data". In: *Nature methods* 16.3 (2019), 243—245. ISSN: 1548-7091. DOI: 10.1038/s41592-018-0308-4. URL: https://europepmc.org/articles/PMC6402590.

[81] M. Wattenberg, F. Viégas, and I. Johnson. "How to Use t-SNE Effectively". In: *Distill* (2016). DOI: 10.23915/distill.00002. URL: http://distill.pub/2016/misread-tsne.

[82] J. Mouton. *How to Succeed in Your Master's and Doctoral Studies: A South African Guide and Resource Book*. Van Schaik, 2001. ISBN: 9780627024849. URL: https://books.google.co.za/books?id=uX4lAQAAIAAJ.

[83] C. Robson and K. McCartan. *Real World Research*. Wiley, 2016. ISBN: 9781118745236. URL: https://books.google.co.za/books?id=AdGOCQAAQBAJ.

[84] I.V.S. Mullis, M.O. Martin, P. Foy, and M. Hooper. *TIMSS 2015 International Results in Mathematics*. Boston College, 2016. ISBN: 978-1-889938-29-5. URL: http://timssandpirls.bc.edu/timss2015/international-results/.

[85] C. Rüegg et al. *Math.NET Symbolics*. 2021. URL: https://symbolics.mathdotnet.com/.

[86] WhiteBlackGoose and H. Tang. *AngouriMath*. 2022. DOI: 10.5281/zenodo.6799175. URL: https://github.com/asc-community/AngouriMath.

Appendix A

Algorithms used to answer and generate question cards

This appendix presents the technical aspects of the question cards shown in Section 3.4.4. In particular, the method used to validate a player's answer and the algorithms used to generate the questions are given.

Two C# libraries were used to implement the Computer Algebra System (CAS) of the game, namely Math.Net Symbolics and AngouriMath [85][86]. Math.Net Symbolics was used in cases where speed was a priority, and AngouriMath was used when additional functionality, such as expression simplification, was needed.

A.1 Validating the player's answer

Figure 3.18 shows that players must supply the solution to question cards by means of a text box. However, it can be somewhat difficult to determine if the player's answer is correct, since the computer can make rounding errors. For instance, if the question is $x = 4.2 + 0.12$, then a calculated answer of $x = 4.32$ could be wrong since the computer might round it to $x = 4.320001$. A rounding error does not necessarily happen at the last decimals, since an equation like $x = (4.2 + 0.12) * 1000000$ gives an answer of 4320001 instead of 4320000. Therefore, another method is required to validate the answer.

Instead, Figure A.1 shows that generated questions can be interpreted as finding the root of a function. The question $5x = 30$ can be rewritten as $5x - 30 = 0$, or $x^2 = 12x - 36$ is rewritten as $x^2 - 12x + 36 = 0$. These equations show that there are basically two scenarios in finding a root. The first is a straight cut through the x-axis, while the second is a turn on the x-axis (more than one root for the same x-value).

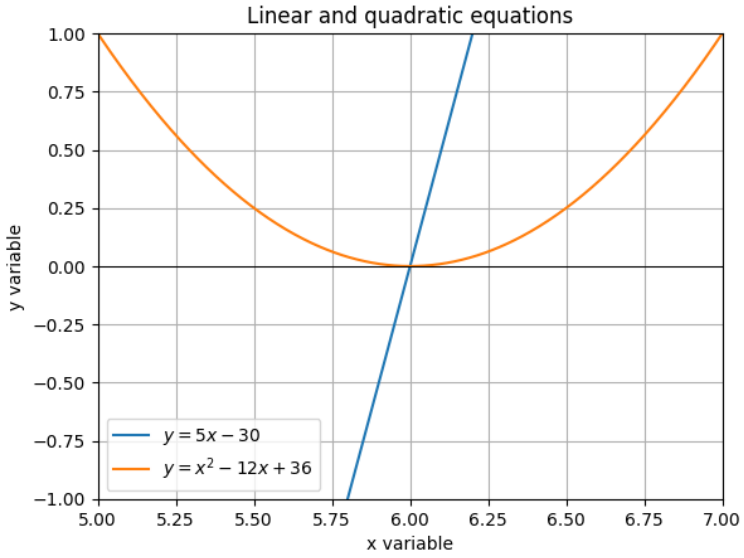


Figure A.1: Linear and quadratic equations

Luckily, the majority of cases will be that of the blue trace in Figure A.1. The approximate linear relationship of the blue trace of Figure A.1 means that a player's answer can be validated by substituting a value just bigger and just smaller than the player's actual answer and checking if the signs differ. For instance, Figure A.1 shows that the question $5x = 30$ has an answer of $x = 6$. Having an answer of x=6 means that the sign of $x = 5.99$ (negative) is different from the sign of $x = 6.01$ (positive) and that the value of $x = 6$ is correct. Unfortunately, this method fails for the case of the orange trace and a regular substitution of the answer is used (the equation must be equal to zero). It is then hoped that a rounding error will not occur.

A.2 Generating questions of question cards

This section covers the algorithms that are used to generate the questions of question cards. The general approach is to start with the answer of the question and add "distractors" to the answer so that a mathematical question is formed. Appendix A.2.1 explains the helper functions that are used in the algorithms, Appendix A.2.2 shows how distractors are generated, Appendix A.2.3 describes the usage of distractors, and Appendix A.2.4 finally provides the pseudocode of the question generating functions.

A.2.1 Helper functions

Helper functions are given below, along with a brief description of what they do. It should be noted that the CAS libraries parse the mathematical expressions by means of nodes.

Nodes of a mathematical expression are combined with operators. For instance, $x + 12$ is an expression with two child nodes, x and 12, which are combined with the "+" operator. Similarly, a more complex equation such as $(12 - 3x) * (4)$ can also be separated into two child nodes, $(12 - 3x)$ and (4), with the "*" operator. The child nodes can again be separated into sub-child nodes until the final level of children only has one node (no operator present in the child).

getFirstChild(equation) Get the first child node. For example, "$(x - 3) * (x + 4)$" returns "$(x - 3)$".

getSecondChild(equation) Get the second child node. For example, "$(x-3)*(x+4)$" returns "$(x + 4)$".

getOperator(equation) Get the operator that combines the outermost nodes. For example, "$(x - 3) * (x + 4)$" returns "$*$".

swapNodes(equation) Swap the outermost nodes so that the first node is second, and the second node is first. For example, $x - (4 + 2x)$ becomes $-(4 + 2x) + x$.

carryOverExpression(fromEq, toEq) Carry one side of the equation to another side. For instance, using carryOverExpression("$12 - x$", "$31 + 5$") becomes "$0 = 31 + 5 - 12 + x$".

getExpandEquation(equation) Expands an equation. For example, "$(x - 3)(x - 2)$" becomes "$x^2 - 5x + 6$"

expandFirstChild(equation) Differs from getExpandEquation(equation) in the sense that it only expands the first child node of an equation.

getSimplifyEquation(equation) Simplifies an equation. For instance, "$x - 3 + 2x$" becomes "$3x - 3$".

getRandomInteger(a,b) Get a random integer that is between a and b (inclusive).

getRandomFloat(a,b) Get a random float that is between a and b (inclusive).

A.2.2 Distractor generators

As mentioned previously, distractors are added to the answer of a question to obfuscate the answer. The obfuscated answer forms the math question. Three types of distractors are presented in this section. The base distractor is called the "miniDistractor" in the code and is used to generate a regular polynomial (such as $5x^3 + 12x^2 - 3$). The "generateDistractorNumber" is a special case of "miniDistractor" that generates only a single number as a distractor. Finally, "generateDistractor" builds a complex distractor by combining "miniDistractor" functions.

Algorithm 2 Generate the base distractor (mini-distractor)

Input: Chance of adding additional terms; max power of distractor polynomial

Output: The mini-distractor's mathematical expression as a string

```
1: function MINIDISTRACTOR(newTermChance, maxPower = 1)
2:     distractor ← ""
3:     if getRandomInteger(0,1) = 0 then
4:         distractor ← distractor + "+" + getRandomInteger(1,29)
5:     else
6:         distractor ← distractor + "-" + getRandomInteger(1,29)
7:     end if
8:
9:     power ← 1
10:    while getRandomFloat(0,1) ¡ newTermChance AND power ¡= maxPower do
11:        term ← getRandomInteger(1,29) + "*xˆ" + power
12:        if getRandomInteger(0, 1) = 1 then
13:            distractor ← "+" + term + distractor
14:        else
15:            distractor ← "-" + term + distractor
16:        end if
17:        power ← power + 1
18:        newTermChance ← newTermChance / 2
19:    end while
20:
21:    return distractor
22: end function
```

Algorithm 3 Generate a distractor that consists of only numbers

Input: void

Output: The distractor number as a string

```
1: function GENERATEDISTRACTORNUMBER( )
2:     return miniDistractor(0,0)      ▷ Zero addition terms and zero degree polynomial
3: end function
```

Algorithm 4 Generate a large distractor from mini-distractors

Input: List of numbers to avoid in distractor solution; Chance of adding another distractor term

Output: A mathematical expression as a string

```
1: function GENERATEDISTRACTOR(avoidNumberList, distractorCombineChance)
2:     distractor ← ""
3:     distractorAcceptable ← false
4:
5:     //Repeat until an acceptable distractor is found
6:     repeat
7:         distractor ← miniDistractor(0.4)
8:         distractorIter ← 0
9:
10:        //Keep adding more mini-distractors and randomly stop
11:        while getRandomFloat(0,1) ¡ distractorCombineChance AND distractorIter
   ¡ 1 do
12:            //Get the operator of the distractor
13:            operatorSymbol ← getRandomInteger(0,2)
14:            if operatorSymbol = 0 then
15:                distractor ← distractor + "+"
16:            else if operatorSymbol = 1 then
17:                distractor ← distractor + "-"
18:            else
19:                distractor ← distractor + "*"
20:            end if
21:
22:            //Append the distractor
23:            distractor ← miniDistractor(0.4)
24:            distractorCombineChance ← distractorCombineChance / 2
25:            distractorIter ← distractorIter + 1
26:        end while
27:
28:        //Check that the distractor is valid and avoids avoidNumberList solutions
29:        distractorAcceptable ← true
30:        if distractor is a solution to avoidNumberList or distractor is invalid  then
31:            distractorAcceptable ← false
32:        end if
33:
34:    until  distractorAcceptable = true
35:
36:    return distractor
37: end function
```

A.2.3 Distractor usage

This section provides two methods that employ the distractors of Appendix A.2.2. Algorithm 5 is used to add distractors to both sides of a mathematical expression (e.g. $x = 5$ becomes $x + 12 * x^2 = 5 + 12 * x^2$). Algorithm 6 is used to add a distractor to a random node (e.g. $x = 5$ becomes $x + \frac{12+x}{12+x} = 5$).

Algorithm 5 Add a distractor to both sides of math equation

Input: left side of equation; right side of equation; list of numbers to avoid in distractor solution

Output: String of modified equation with distractors added

```
1: function addDistractorToBothSides(leftEquation, rightEquation, avoidNumberList)
2:     randomDistractor ← generateDistractor(avoidNumberList, 0.5)
3:     distractOperator ← ""
4:
5:     //Get the operator for the distractor
6:     randomOperator ← getRandomFloat(0,1)
7:     if randomOperator < 0.4 then
8:         distractOperator ← "+"
9:     else if randomOperator < 0.8 then
10:        distractOperator ← "-"
11:    else
12:        distractOperator ← "/"
13:    end if
14:
15:    //Add the distractor to both sides of the equation
16:    if distractOperator = "/" OR getRandomInt(0,1) = 0 then
17:        leftEquation ← leftEquation + distractOperator + randomDistractor
18:        rightEquation ← rightEquation + distractOperator + randomDistractor
19:    else
20:        leftEquation ← randomDistractor + distractOperator + leftEquation
21:        rightEquation ← randomDistractor + distractOperator + rightEquation
22:    end if
23:
24:    return leftEquation + "=" + rightEquation
25: end function
```

Algorithm 6 Add a distractor to a random node

Input: equation to add the distractor to;list of numbers to avoid in distractor solution

Output: String representing modified equation with distractor added to a random node

```
1: function addDistractorToRandomNode(equation, avoidNumberList)
2:     randomDistractor ← generateDistractor(avoidNumberList, 0.5)
3:     nodeToAddTo ← ""
4:     newNode ← ""
5:
6:     //Check which node to add the distractor to
7:     if getRandomInt(0,1) = 0 then
8:         nodeToAddTo ← getFirstChild(equation)
9:     else
10:        nodeToAddTo ← getSecondChild(equation)
11:    end if
12:
13:    //How the random node should be added
14:    operationOption ← getRandomInteger(0,7)
15:    if operationOption = 0 then
16:        newNode ← expandEquation((nodeToAddTo)/(randomDistractor))*(randomDistractor)
17:    else if operationOption = 1 then
18:        newNode ← expandEquation((nodeToAddTo)*(randomDistractor))/(randomDistractor)
19:    else if operationOption = 2 then
20:        newNode ← (randomDistractor)*expandEquation((nodeToAddTo)/(randomDistractor))
21:    else if operationOption = 3 then
22:        newNode ← 1/(randomDistractor)*expandEquation((nodeToAddTo)*(randomDistractor))
23:    else if operationOption = 4 then
24:        newNode ← expandEquation((nodeToAddTo)−(randomDistractor))+(randomDistractor)
25:    else if operationOption = 5 then
26:        newNode ← expandEquation((nodeToAddTo)+(randomDistractor))−(randomDistractor)
27:    else if operationOption = 6 then
28:        newNode ← (randomDistractor)+expandEquation((nodeToAddTo)−(randomDistractor))
29:    else
30:        newNode ← −(randomDistractor)+expandEquation((nodeToAddTo)+(randomDistractor))
31:    end if
32:
33:    equation ← Replace nodeToAddTo of equation with newNode
34:    return equation
35: end function
```

A.2.4 Question generating functions

Finally, the algorithms used to generate the questions are given below.

Algorithm 7 Generate polynomial questions

Input: Answers; complexity/iterations of the question

Output: String representing the polynomial question

```
1:  function GENERATEQUESTIONPOLY(answerList,iterations)
2:      leftEquation ← ""
3:      rightEquation ← ""
4:      baseQuestion ← ""
5:      //Build polynomial with answers as roots
6:      for (every answer in the list of answerList) do
7:          baseQuestion ← baseQuestion + "(x −answer)*"
8:      end for
9:      baseQuestion ← remove last "*" from baseQuestion
10:     //Randomly expand base question
11:     if getRandomFloat(0,1) ¡ 0.7 then
12:         baseQuestion ← getExpandEquation(baseQuestion)
13:     end if
14:     //Generate the left and right equations
15:     if getRandomFloat(0,1) ¡ 0.7 then
16:         leftEquation ← baseQuestion
17:         rightEquation ← "0"
18:     else
19:         leftEquation ← "0"
20:         rightEquation ← baseQuestion
21:     end if
22:     //Random swap of starting equations
23:     if getRandomInteger(0,1) = 0 then
24:         leftEquation ← swapNodes(leftEquation)
25:     else
26:         rightEquation ← swapNodes(rightEquation)
27:     end if
28:     //Random carry over of start equation
29:     if getRandomInteger(0,1) = 0 then
30:         carryOverExpression(leftEquation,rightEquation)
31:     else
32:         carryOverExpression(rightEquation,leftEquation)
33:     end if
```

```
34:     //Mangle the question with distractors
35:     transformCount ← 0
36:     while transformCount < iterations do
37:         randomOperation ← getRandomFloat(0,1)
38:         if randomOperation ¡ 0.2 then                    ▷ Carry over an expression
39:             if getRandomInteger(0,1) = 0 then
40:                 carryOverExpression(leftEquation,rightEquation)
41:             else
42:                 carryOverExpression(rightEquation,leftEquation)
43:             end if
44:         else if randomOperation ¡ 0.4 then                          ▷ Swap nodes
45:             if getRandomInteger(0,1) = 0 then
46:                 leftEquation ← swawpNodes(leftEquation)
47:             else
48:                 rightEquation ← swawpNodes(rightEquation)
49:             end if
50:         else if randomOperation ¡ 0.8 then          ▷ Add a distractor to both sides
51:             DistractorEquation ← addDistractorToBothSides(leftEquation,
    rightEquation, answerList)
52:             leftEquation ← get left math expression from DistractorEquation
53:             rightEquation ← get right math expression from DistractorEquation
54:         else if randomOperation ¡ 0.9 then
55:             if getRandomInteger(0,1) = 0 then ▷ Add an distractor to a random node
56:                 leftEquation ← addDistractorToRandomNode(leftEquation,
    answerList)
57:             else
58:                 rightEquation ← addDistractorToRandomNode(rightEquation,
    answerList)
59:             end if
60:         else                                          ▷ Expand an equation side
61:             if getRandomInteger(0,1) = 0 then
62:                 leftEquation ← expandFirstChild(leftEquation)
63:             else
64:                 rightEquation ← expandFirstChild(rightEquation)
65:             end if
66:         end if
67:         transformCount ← transformCount + 1
68:     end while
```

```
69:     //Simplify one of the board sides for additional equation obfuscation
70:     if leftEquation is more complex than rightEquation then
71:         leftEquation = getSimplifyEquation(leftEquation)
72:     else
73:         rightEquation = getSimplifyEquation(rightEquation)
74:     end if
75:     return leftEquation + "=" +rightEquation
76: end function
```

It should be noted that "generateQuestionPoly" can take a list of answers in its *answerList* argument. Therefore, linear questions are created by only having one answer in *answerList*. On the other hand, quadratic questions are produced by passing two answers to *answerList*.

Algorithm 8 Generate substitute questions

Input: Answer; Complexity of the question

Output: Substitute question

```
 1: function GENERATEQUESTIONSUBSTITUTE(answer,iterations = 4)
 2:     //Get the substitute expression
 3:     subExpr ← "x"
 4:     repeat
 5:         if getRandomInteger(0,1) = 1 then
 6:             subExpr ← (generateDistractorNumber())*x
 7:         end if
 8:         if getRandomInteger(0,1) = 1 then
 9:             subExpr ← subExpr+ generateDistractorNumber()
10:         end if
11:     until subExpr != "x"
12:
13:     //See if nodes of subExpr should be swapped
14:     if getRandomInteger(0,1) = 1 then
15:         subExpr ← swapNodes(subExpr)
16:     end if
17:
18:     //Use the subExpr to get the poly question
19:     subFunction(inputForExpr) ← Turn subExpr into a callable function
20:     subAnswer ← subFunction(answer)
21:     polyExpr ← generateQuestionPoly(subAnswer,iterations)
22:     polyExpr ← Replace all "x" with "a" in polyExpr
23:     return "Solve polyExpr if a = subExpr"
24: end function
```

Algorithm 9 Generate exponential questions

Input: Answer; Complexity of the question

Output: Exponential question

```
1: function generateQuestionExponential(answer,iterations = 3)
2:
3:     //Get the exponential substitute expression
4:     subExpr ← getRandomInteger(2,5) + "^(x)"
5:
6:     //Use the exponential sub expr to get the poly question
7:     subFunction(inputForExpr) ← Turn subExpr into a callable function
8:     subAnswer ← subFunction(answer)
9:     polyExpr ← generateQuestionPoly(subAnswer,iterations)
10:    expExpr ← Replace all "x" in polyExpr with subExpr
11:    return "Solve expExpr"
12: end function
```

Algorithm 10 Generate simultaneous equation questions

Input: Answer of the x variable; answer of the y variable

Output: Simultaneous equation question

```
1: function generateQuestionSimultanious(answerX,answerY)
2:     simultaniousEquation1 ← ""
3:     simultaniousEquation2 ← ""
4:
5:     simultaniousEqLeft ← ""
6:     simultaniousEqRight ← ""
7:     validEquation ← false
8:     repeat
9:         equationOperator ← "-"
10:        if getRandomInteger(0,1) = 0 then
11:            equationOperator ← "+"
12:        end if
13:        simultaniousEqLeft ← generateDistractorNumber() + "*x" +
   equationOperator + generateDistractorNumber() + "*y"
14:
15:        //Random if the x and y variable order should be swapped
16:        if getRandomInteger(0,1) = 0 then
17:            simultaniousEqLeft ← swapNodes(simultaniousEqLeft)
18:        end if
```

19: //Check that the generated equation is valid
20: subFunction($inputX$, $inputY$) ← Turn $simultaniousEqLeft$ into a callable function
21: $simultaniousEqRight$ ← subFunction($answerX$, $answerY$)
22: **if** subFunction($answerX$, $answerY$) is finite and real valued **then** $validEquation$ ← true
23: **end if**
24: **until** $validEquation$ = true
25:
26: //Random if equation should be carried over
27: **if** getRandomInteger(0,1) = 0 **then**
28: **if** getRandomInteger(0,1) = 0 **then**
29: carryOverExpression($simultaniousEqLeft$,$simultaniousEqRight$)
30: **else**
31: carryOverExpression($simultaniousEqRight$,$simultaniousEqLeft$)
32: **end if**
33: **end if**
34:
35: //Get the first simultaneous equation
36: $simultaniousEquation1$ ← ""
37: **if** getRandomInteger(0,1) = 0 **then**
38: $simultaniousEquation1$ ← $simultaniousEqLeft$ + "=" + $simultaniousEqRight$
39: **else**
40: $simultaniousEquation1$ ← $simultaniousEqRight$ + "=" + $simultaniousEqLeft$
41: **end if**
42:
43: //Get the second simultaneous equation
44: **repeat**
45: $simultaniousEquation2$ ← repeat Line 5 to Line 41 but use $simultaniousEquation2$ in all occurrences of $simultaniousEquation1$
46: **until** $simultaniousEquation2$!= $simultaniousEquation1$
47:
48: **return** $simultaniousEquation1$ and $simultaniousEquation2$
49: **end function**

Algorithm 11 Generate number questions

Input: Answers; complexity/iterations of the question

Output: Number question to answer

```
1: function GENERATEQUESTIONNUMBER(answer,iterations)
2:     constructedEquation ← answer
3:     operationList ← new empty stack
4:
5:     for Loop iterations number of times do
6:         //Operation used to make question
7:         calcOperator ← "+"
8:         if getRandomInteger(0,1) = 0 then
9:             calcOperator ← "-"
10:        end if
11:
12:        //Inverse operators used to balance question with answer
13:        equationOperator ← ""
14:        if calcOperator = "-" then
15:            equationOperator ← "+"
16:        else
17:            equationOperator ← "-"
18:        end if
19:
20:        //Add the distractors
21:        distractorNumber ← generateDistractorNumber()
22:        OperationList.Push("equationOperator" + "(distractorNumber)")
23:        constructedEquation ← constructedEquation + "calcOperator" +
   "(distractorNumber)"
24:    end for
25:
26:    //Generate the final question
27:    dispQuestion ← Calculate the numeric value of constructedEquation
28:    for every element in OperationList do
29:        dispQuestion ← dispQuestion + OperationList.Pop()
30:    end for
31:
32:    return "Calculate value of " + dispQuestion
33: end function
```

Fraction questions are generated using the same principles of Algorithm 11. The primary difference is that "+" operations are replaced with "*" operations, and "-" operations are replaced with "/" operations. The number distractor generator of Line 21 is also

replaced with a fraction distractor generator. The fraction distractor operates similarly to the number distractor generator, but generates random valid fractions (ensures that the denominator cannot be zero).

Appendix B

Cards in deck distributions

The tables in this section provide the amount of cards that are in the player's deck, at each level, for each of the three decks. It should be remembered that players always start a game with a random number card in their hands, and usually draw two blue deck cards and three green deck cards at the start of a game. The start of a turn (except for the first turn of the first player) allows players to draw one card from each deck. The first player may not draw cards in his first turn. Players only draw a random red deck card when it is their turn in the battle round.

Table B.1: Blue deck card distribution at each level

Card	lvl 1	lvl 2	lvl 3	lvl 4	lvl 5	lvl 6	lvl 7	lvl 8	lvl 9	lvl 10	lvl 11	lvl 12
Plus	5	3	2	2	2	2	2	2	2	2	2	2
Minus	0	3	2	2	2	2	2	2	2	2	2	2
Multiply	0	0	2	2	2	2	2	2	2	2	2	2
Divide	0	0	0	0	0	2	2	2	2	2	2	2
Cube	0	0	0	0	0	0	0	0	0	0	1	1

Table B.2: Red deck card distribution at each level

Card	lvl 1	lvl 2	lvl 3	lvl 4	lvl 5	lvl 6	lvl 7	lvl 8	lvl 9	lvl 10	lvl 11	lvl 12
Number Question	5	5	3	2	2	2	2	2	2	2	2	2
Fraction Question	0	0	2	2	2	2	2	2	2	2	2	2
Linear Question	0	0	0	2	2	2	2	2	2	2	2	2
Quadratic Question	0	0	0	0	0	0	0	0	4	2	2	2
Substitution Question	0	0	0	0	0	0	0	0	0	4	2	2
Simultaneous Question	0	0	0	0	0	0	0	0	0	0	4	2
Exponential Question	0	0	0	0	0	0	0	0	0	0	0	4

Table B.3: Green deck card distribution at each level

Card	lvl 1	lvl 2	lvl 3	lvl 4	lvl 5	lvl 6	lvl 7	lvl 8	lvl 9	lvl 10	lvl 11	lvl 12
Square	0	0	0	0	5	3	2	2	2	2	2	2
Square root	0	0	0	0	0	2	2	2	2	2	2	2
Negate	0	0	0	0	0	0	3	3	3	3	3	3
Variable card	0	0	0	0	0	0	0	9	9	9	9	9
Cube	0	0	0	0	0	0	0	0	0	0	1	1

Appendix C

Game requirements fulfilled by game's design

Requirements of the game design and the design's ability to fulfill these requirements are given in Table C.1.

Table C.1: Verification of game design requirements

Requirement no.	Requirement outcome	Requirement description	Reason for outcome
1a	Mostly satisfied	Game rules and mechanics shall align naturally with those of mathematics	Every game move follows the rules of mathematics. However, some mathematical rigour has been lost with certain design choices. For instance, the core gameplay allows the player to adjust the board inequality by playing cards. However, normal mathematics would not allow these adjustments to an inequality. Function cards also use the question symbol (?) instead of a proper variable for its argument. Lastly, the graphing mechanic is not able to draw discontinuities, but joins the closest rational values instead (Figure 3.23).

1b	Satisfied	The game shall present the mathematics in a clear and explicit form to the player	Mathematics is explicitly and directly shown to the player. The visual presentation of the mathematics is similar to how it would appear in a textbook.
1c	Satisfied	There shall be 2+ players in the game	The game is a two-player card game
1d	Satisfied	Opponents shall have the same type of goal as the player	The opponents take part in a symmetric game, meaning that the opponent plays cards similarly to the player. The only difference is that the opponent has an opposite comparator sign than that of the player to ensure that there is conflict in the game. The opponent therefore serves as a mentor that the player can learn from while playing the game.
1e	Satisfied	The opponents shall be controlled by the computer (AI)	The reinforcement learning implementation of the AI opponent is given in Chapter 4.
1f	Satisfied	The opponents shall be implemented with reinforcement learning.	The reinforcement learning implementation of the AI opponent is given in Chapter 4.
1g	Satisfied	The game levels shall introduce the different mathematical concepts. Harder levels convey more advanced concepts	Section 3.6.2 shows that progression is implemented using levels. Each level introduces the player to new cards which increase the game's difficulty. Also, the game initially starts with ordinary arithmetic and progresses towards algebraic manipulation with variables.

1h	Satisfied	The game shall not rely on the reflexes or dexterity of the player to win games, but rather on the strategy/understanding that the player has of the mathematics	There are no time-based mechanics in the game. Therefore, players have infinite time in a round to consider options. Question cards also have no time restriction, meaning that players can take as long as they want to solve the questions. The lack of time-based mechanics aids in reducing math anxiety that players might experience (Section 2.1.3).
1i	Satisfied	Game mechanics shall be based on desirable difficulties such as varying conditions of study, spacing the study, interleaving study tasks and using tests to learn the concepts	Desirable difficulties have been implemented throughout the game. Spacing of the content was ensured by having twelve levels that require beating each opponent and gaining enough water before progressing to the next level. Study tasks were interleaved by always adding more cards to the deck without removing old cards, therefore forcing the player to master content in an interleaved manner. Varying conditions of practice are ensured by the random nature of the card game and the questions generated by the question cards(no two games are exactly the same). The questions from the question cards also serve the purpose of using tests to learn concepts.

1j	Moderately Satisfied	The game shall use mediation and scaffolding to provide aid such as attracting players' attention, simplifying tasks by reducing the number of choices, directing players back to the original objective, marking critical aspects of the problem, controlling the frustration of the player, and demonstrating the ideal solution	The AI opponent provides mediation by demonstrating how the player can play the game. Furthermore, the progression and levels of the game slowly introduce new concepts to aid with scaffolding. The different opponents with different skill levels also aim to be a form of difficulty adjustment in the game. However, there are no personalized adjustments or explanations in the game.
2a	Satisfied	The core gameplay shall function without internet	All functionality of the game can be accessed without requiring an internet connection. The player only plays against offline AI opponents.
2b	Satisfied	The game shall consume mobile data sparingly for non-critical functions where there is no other option	The game sometimes uses the internet to send gameplay data for the experiment of Chapter 6. Subsequently, there are no other game functions that require the internet.
2c	Satisfied	The game shall be developed for low performance devices	The game is a card game, meaning that most elements are 2D and don't require the higher performance necessities of 3D games. The game requires devices to meet the minimum system requirements of the Unity game engine. System requirements of the game can be found in Appendix K.
2d	Satisfied	The game's audio shall not be compulsory to play the game	*Digit Draw* features no sound due to development time and budget constraints. The lack of sound is detrimental to the immersive experience of the game, but aids in creating a non-disruptive study environment.

3a	Satisfied	The game shall cater for Bartle's player model (explorers, achievers, killers and socializers)	Explorers might enjoy discovering the mathematics, levels and rescued animals of the game, while achievers will try to rescue all the animals and unlock all the levels. Killers will enjoy defeating the predator animal opponents and obtaining high scores, while socializers could perhaps enjoy sharing what they learned or discovered with other players. Socializers might find satisfaction therein that they build a talking animal community as they rescue animals.
3b	Satisfied	At least three of the four key elements of fun (hard, people, easy, serious) mentioned by Lazzaro shall be present in the game	Hard fun occurs when players beat the difficult AI opponents and easy fun is associated with discovering mathematical rules and forging cards. Serious fun takes place in rescuing animals and playing card games that always have the same gameplay, but different outcomes due to randomness. People fun occurs when players talk to each other about what they learned in the game.
3c	Satisfied	The game shall use input randomization to keep the game interesting	The random cards that players draw use input randomness, which keeps the games interesting. Questions generated by question cards are random and not from a test bank.

3d	Satisfied	The game shall keep output randomization to a minimum	There are no cards that rely on chance to determine if the card play outcome is successful. For instance, question cards do not provide a random number once the card is played. Instead, a few questions are randomly generated, and the player can choose which answer he wants by correctly answering the question. The ability to choose an answer gives the player a sense of control instead of being given a random number.
3e	Satisfied	The game shall provide a variety of choices with no apparent meta strategy. Experience can guide the player on what choices are superior for the situation	The game has some depth since cards can be played in various combinations. The training of the RL agent also revealed no meta strategy to exploit for every situation. There are situations where the optimal choice is apparent, but that choice won't be optimal for every situation. For instance, there are situations where the multiply operator card is very beneficial, but there are also situations where the plus operator card means a lot more to the player.
3f	Satisfied	Choices made by the player shall have an impact on the outcome of the game	Every action taken by the player must be performed by the player. The player will most likely lose a game when playing random cards against the opponents. The skill requirement for the game is illustrated in Chapter 5 where the performance of the random player is inferior to trained policies.

3g	Mostly satisfied	Players shall receive clear feedback/rewards when they perform the correct action	Question cards provide clear feedback when a question is answered incorrectly and reward the player when a question is answered correctly. The end game summary also clearly shows the player that he has won or has been defeated. However, there is not a clear indication if the player played the optimal cards during the game. The player only receives feedback at the end when a game is lost/won. Luckily, games are short (only six rounds) which makes it easier to determine when an inferior move has been played.
3h	Satisfied	The game shall create a stress-free learning environment for the player to experiment/fail in order to learn	There is no penalty for losing games, meaning that players do not lose progress when a game is lost. In fact, players even gain reduced rewards in some cases, such as animal rescue chance and player exp. Players can also choose the difficulty of their opponent, and the stress of losing against an AI opponent is not as much as losing to a human. Players are also encouraged to experiment with strategies in order to obtain a new high score.
3i	Barely satisfied	Game shall provide feedback on how to solve/improve on the mathematical problem. Possible forms of feedback include learning from the opponent's strategy or funnelling the player to the answer/solution.	See the motivation of requirements $1j$. Players are therefore able to learn from the opponent, but there are no mechanics that provide personalized feedback on where the player made an error.

3j	Unsatisfied	The game flow shall not be interrupted by prompts	Question cards cause a completely new question to pop up, which could break the flow of the game. However, the benefits of the question cards are believed to outweigh the drawbacks since the question cards reduce output randomness and enable the players to practice mathematics in a more traditional manner.
3k	Satisfied	The game must prevent users from impulsive clicking without thinking about a solution	A wrong move in the card game is carried over until the end of the game. The fact that a wrong move has consequences means that players should think before adding cards to the board since it can make the game more difficult to win if the move is detrimental to the player. Also, question cards require a numerical input to answer the question (more difficult to guess than multiple choice), and the player can only answer a question card question once.

Appendix D

ML-Agents hyperparameters

This appendix describes the hyperparameters found in the Unity game engine's ML-Agents framework [58]. Only hyperparameters affecting the performance of an RL agent are shown. Therefore, the hyperparameters related to formatting training output, saving the RL models or choosing between the PPO or SAC algorithm are not shown.

Table D.1 provides general hyperparameters that are mostly related to the ANN used in PPO and SAC, while Table D.2 presents hyperparameters that are specifically related to PPO and Table D.3 are the hyperparameters specific to SAC. Table D.4 includes the hyperparameters relevant to self-play.

Table D.1: Hyperparameters shared by PPO and SAC in Unity ML-Agents [58]

Hyperparameter	Description	Default value	Typical range
time_horizon	Number of experience steps to collect per-agent before adding it to the experience buffer. Seems to be similar to the number of steps in n-step methods.	64	32 to 2048
threaded	Allows the environment to step while updating the model. It is recommended to disable threaded when using self-play.	False	True or False
learning_rate	Step size of the gradient descent updates.	3E-4	1E-5 to 1E-3
batch_size	Number of experiences used in each iteration of gradient descent.		32 to 512

buffer_size	The PPO algorithms use this hyperparameter to determine the number of experiences to collect before updating the model. SAC uses it to define the max size of the experience replay buffer, which should always be thousands of times larger than the episode to incorporate old and new episodes.	PPO: 10 240 SAC: 50 000	PPO: 2048 to 409600 SAC: 50 000 to 1 000 000
learning_rate _schedule	Determines how the learning rate should be annealed. Constant does not decrease the learning rate.	PPO: linear SAC: constant	linear or constant
hidden_units	Number of neurons in a hidden layer of the ANN.	128	32 to 512
num_layers	Number of hidden layers in the ANN.	2	1 to 3
normalize	States if the running average and variance of the inputs are used to normalize the inputs. Is advantageous for complex continuous action problems, but may yield negative results for simple discrete problems.	False	True or False
gamma	Discount factor (γ) for future rewards.	0.99	0.8 to 0.995

Table D.2: PPO hyperparameters in Unity ML-Agents framework [58]

Hyperparameter	Description	Default value	Typical range
beta	Weight of the entrophy regularizes (C_2 in Equation 4.3).	5.0E-3	1.0E-4 to 1.0E-2
epsilon	Indicates the clipping bounds of the clipping loss function of Equation 4.4.	0.2	0.1 to 0.3
beta_schedule	Determines how the beta hyperparameter should be annealed.	learning _rate _schedule	linear or constant
epsilon_schedule	Determines how the epsilon hyperparameter should be annealed.	learning _rate _schedule	linear or constant
lambd	Determines the decay rate (λ) of the advantage function in Equation 4.5. Smaller lambd values causes the agent to rely more on the value estimate than the reward received.	0.95	0.9 to 0.95
num_epoch	Number of times that must be made through the experience replay buffer when performing gradient descent.	3	3 to 10
shared_critic	Declares if the policy and value networks should share a backbone.	False	True or False

Table D.3: SAC hyperparameters in Unity ML-Agents framework [58]

Hyperparameter	Description	Default value	Typical range
buffer_init_steps	Number of replay experiences that must first be collected before updating the policy's model.	0	1000 to 10 000
init_entcoef	Initial value to use for the temperature parameter before automatically adjusting it. Higher value encourages more initial exploration.	1.0	0.05 to 0.5
tau	Corresponds to τ that is used to obtain the approximation of $\bar{\theta}$ in Equation 4.12	0.005	0.005 to 0.01
steps_per_update	Average number of steps that should elapse before making an update to the agent's model. A higher steps_per_update value will increase the amount of CPU computations in order to improve the agent's sample efficiency.	1	1 to 20
reward_signal _num_update	Sets the update frequency of reward signals based on the number of steps per mini-batch sampled. Usually kept in sync with steps_per_update, but can be beneficial to adjust when performing imitation learning	steps _per _update	

Table D.4: Self-play hyperparameters in Unity ML-Agents framework [58]

Hyperparameter	Description	Default value	Typical range
play_against_latest _model_ratio	Probability of playing against the policy of the latest opponent. Probability of 1 - play_against_latest_model_ratio to play against an old opponent policy in their window.	0.5	0.0 to 1.1
window	Number of opponent policies that the agent can play against. The window contains only the latest opponent policies captured by the snapshot.	10	5 to 30
save_steps	Number of training steps elapsed before a snapshot is taken of the agent's policy and added to its window of past agents.	20 000	10 000 to 100 000
team_change	Number of training steps that an agent remains the learning entity before it becomes the opponent and the opposing agent becomes the learning agent. A larger value allows the agent to accumulate stronger policies in its window to be used when it becomes the opponent. Setting this value too high can cause over-fitting to the opponent's policies.	5 × save_steps	(4 to 10) × save_steps
swap_steps	The number of ghost-steps that must elapse before swapping to another policy in the window during training. The swap_steps size needed for x swaps in a team's training period is calculated with $swap_steps = \frac{num_agents_in_team}{num_opponent_agents} \times \frac{team_change}{x}$	10 000	10 000 to 100 000

Appendix E

Game played by the control group

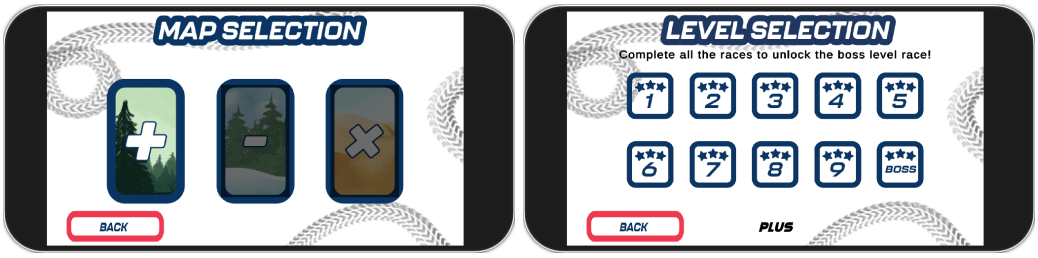
Section 6.1 mentions that a control group is needed in order to improve the experiment's validity against external threats. The game given to the control group is known as *Number Nitro* and is presented in this section. *Number Nitro* was developed by Leon Freese at the NWU as part of his final year project in computer and electronic engineering, under the supervision of Prof. Pieter van Vuuren. The objective of Number Nitro is to improve the arithmetic skills of learners who are in grade 4.

Number Nitro was chosen for the control game due to it looking like a mathematics game (makes it less obvious which game is the experimental game) without influencing the results of the experimental game (*Number Nitro* aims to improve arithmetic skills opposed to algebra skills). Furthermore, *Number Nitro* made it possible to incorporate the same type of passive gameplay data capturing methods of Section 6.3.3 into the control game since the source code of *Number Nitro* was available for modification.

The principal gameplay of *Number Nitro* can be seen in Figure E.1. The game has a stark resemblance to the *Hill Climb Racing* game, where a player races in his car against AI opponents to see who can reach the finish line the quickest. The player is able to provide a speed boost to his car by correctly answering the multiple-choice question shown at the bottom of Figure E.1. Incorrect answers prohibit a speed boost and eventually lead to the opponents passing the player's car.

Figure E.1: Number Nitro gameplay

The game contains a variety of levels, with different levels changing the difficulty/type of multiple choice questions that are generated during the race. Figure E.2a shows that the game contains addition, subtraction, and multiplication types of questions that rely on integer arithmetic. Figure E.2b continues to illustrate that a level category (such as addition) consists of 9 levels and a boss level. The levels increase in difficulty by adding more significant digits to the numbers that are used in the generated questions. The boss fight uses a mixture of the questions that were generated in the previous levels (starting at level 1 and increasing to level 9). Players must obtain at least one star in a level (place third or higher in the race) before they are allowed to play subsequent levels.



(a) Level categories (b) Specific level selection

Figure E.2: Levels in control game

Number Nitro contained personalization options (see Figure E.3) where players can change the appearance of their car, car wheels, driver, and car colour. All customization options are unlocked by default and are not earned during gameplay.

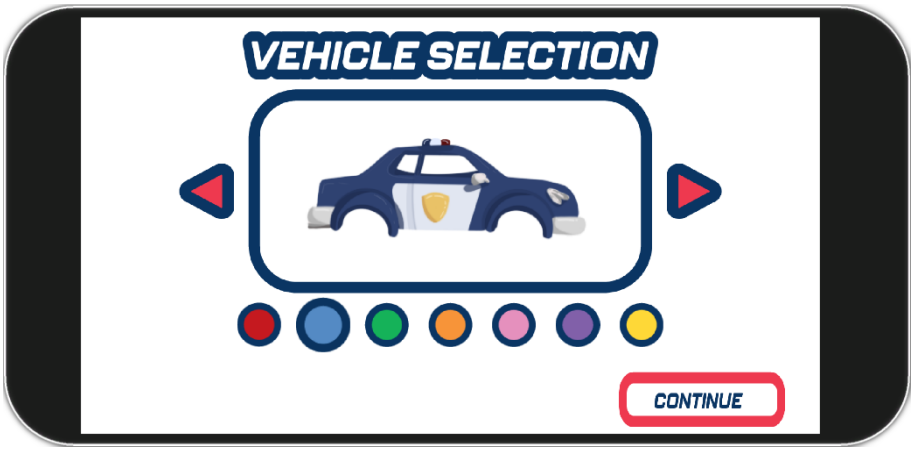


Figure E.3: Number Nitro customization

Finally, the main menu for *Number Nitro* is given in Figure E.4. Players can navigate the main menu to select a level to play, do a quick race (randomizes the level to play for a selected level category) or perform player personalization (as shown in Figure E.3).

Figure E.4: Number Nitro main menu

Appendix F

Adapted TIMSS test

The questions of the adapted TIMSS test, used in the experiment of Chapter 6, are provided in Table F.1. Only references to the questions are given due to the permission agreement with the authors of the TIMSS test (permission reference IEA-24-027) [84]. Learners had 32 minutes to complete the test. The test was available in Afrikaans and English.

Table F.1: Adapted 2015 TIMSS test used in the study [84]

Order	TIMSS 2015 Block	TIMSS Question Number	Question Domain	Topic Area	Cognitive Domain
1	M01	M01_09	Algebra	Expressions and Operations	Applying
2	M01	M01_10	Algebra	Expressions and Operations	Applying
3	M07	M07_05	Algebra	Relationships and Functions	Applying
4	M05	M05_03	Number	Whole Numbers	Knowing
5	M07	M07_03	Number	Ratio, Proportion and Percent	Knowing
6	M05	M05_09	Algebra	Equations and Inequalities	Knowing
7	M05	M05_07	Algebra	Relationships and Functions	Reasoning
8	M07	M07_14	Data and Chance	Chance	Knowing
9	M07	M07_01	Number	Whole Numbers	Knowing
10	M07	M07_12	Data and Chance	Data Interpretation	Knowing
11	M05	M05_08	Algebra	Expressions and Operations	Knowing
12	M05	M05_11 A	Algebra	Expressions and Operations	Knowing
13	M05	M05_11 B	Algebra	Expressions and Operations	Knowing
14	M01	M01_07	Algebra	Expressions and Operations	Knowing
15	M07	M07_13	Data and Chance	Data Interpretation	Applying
16	M01	M01_08	Algebra	Equations and Inequalities	Applying
17	M07	M07_06	Algebra	Equations and Inequalities	Applying
18	M05	M05_01	Number	Fractions, Decimals, and Integers	Knowing
19	M01	M01_05	Algebra	Expressions and Operations	Knowing
20	M05	M05_06	Algebra	Equations and Inequalities	Knowing
21	M07	M07_07	Algebra	Expressions and Operations	Knowing
22	M01	M01_12	Geometry	Geometric Shapes	Reasoning
23	M01	M01_03	Algebra	Expressions and Operations	Knowing
24	M01	M01_04	Number	Ratio, Proportion and Percent	Knowing
25	M01	M01_11	Geometry	Geometric Shapes	Applying

Reference: IEA-24-027

PERMISSION REQUEST FORM

To be completed by anyone seeking permission to reuse or reproduce IEA materials.
Please read and adhere to IEA's Permission Request Guidelines.

*If you have any questions about your request, contact IEA at permission.requests@iea.nl, indicating your **full name** (as submitted) and the **date** of your request. **Thank you!***

1. IEA Materials

1. **Which study are you requesting IEA material from?**
   TIMSS

2. **You have requested materials from these cycles of TIMSS:**
   2015

3. **Which type of IEA material would you like to request?**

☑ Released student assessment item(s) and/or passage(s) from selected study (restricted-use items)

☐ Background questionnaire(s) from selected study — home, school, student, or teacher (type of background questionnaire(s) available is dependent on which study is selected)

☐ Text excerpt(s) — i.e. chapter number, figure(s), and/or tables in IEA's publications

☐ Other (not included in the categories above)

4. **Please list the specific student assessment/test item(s) from TIMSS you would like to request.**
   2015: G8, Block M01, M05 and M07

5. **Which grade(s) of TIMSS are you requesting IEA material from?**
   Only eighth grade

6. **Which TIMSS subject(s) are you requesting IEA material from?**
   Only math

7. **Please indicate the language(s) of the requested material.**
   Both English (International Version) and other language(s)

8. **Please list your requested national language(s).**
   Afrikaans and English

2. Your Intended Use

Reference: IEA-24-027

1. Please describe your intended use of the material.

☑ Thesis (Bachelor's/Master's/Doctoral)

☑ Academic purposes (i.e. projects/presentations)

☑ Publication (i.e. article/book)

☑ Data collection

☐ Other (not included in the categories above)

2. Please indicate the details about your data collection.

Sample Size: *50-200* ***Data Collection Window:*** *around July 2024*

3. Will you use the requested materials for a commercial purpose?

☐ Yes

☑ No

4. Please provide a summary about the reasons for accessing and using the requested materials.

I am developing a serious game for grade 9 learners that aims to improve algebra skills and motivation to do mathematics. This project is part of my Masters study in Engineering at the North-West University of South Africa.

I want to test the game by using a pre-tests and post test, together with a control and experimental group. A comparison between the difference in scores of the experimental and control group can then show if the game had any effect. The pre-test and post-test will be the same test.

I would like to use the TIMMS test as the pre-test and post-test since it does not require textbook specific knowledge. This aids in measuring possible mathematical insight/aptitude gained from the serious game.

NOTE: *By filling out and signing this permission request, you are agreeing that no comparisons to the international study results will be made and that no conclusions about the study as a whole will be drawn.*

5. Please provide a summary about how the information will be reported.

The difference between the pre-test and post-test scores of the experimental and control group will be compared to establish if the serious game had any effect. Results will be analyzed on a question by question basis to see if only algebra scores increased. Hypothesis test and graphs will be used to indicate any differences.

No TIMMS question will be published. Only references to the questions' ID and block will be made.

6. Please provide sufficient information about where the requested materials or subsequent research will appear.

☑ Print publication

☑ Online publication (i.e. online journal)

☐ Website

☐ Other

7. Information about the planned publication, if applicable.

Author: *Andrew le Roux; Prof. Pieter van Vuuren; Prof. Roelof Coetzer*

Title: *A novel serious game utilising reinforcement learning to enrich high school mathematics*

Publisher/Sponsor:

Language: *English*

Intended Audience: *Engineers and educators*

Number of Distribution Copies:

Retail Price:

Release Date: *2024/2025*

8. Please share the URL where the requested materials or research will appear.

3. Consent and Requester Information

First Name: Johan Andrew Ian

Last Name: le Roux

Email: andrewleroux777@gmail.com

Affiliation:

Name of Institution or Organization: *North-West University*

City: *Potchefstroom*

Country: *South Africa*

☑ I consent to the processing of my personal data for the purpose of processing this permission request.

☑ I agree that my personal data will be kept on file by IEA for documentation purposes for a period of five years and then deleted in accordance with IEA's retention policy.

Reference: IEA-24-027

☑ I have read and agree to the Permission Request Disclaimer.

☑ I confirm that I have filled out this form truthfully and to the best of my knowledge. Providing IEA with incorrect or incomplete information will not only invalidate permission if granted but can also hold me liable for any damage arising from my failure to comply with these requirements. I agree that the requested material will be used only as part of, or alongside the original work, where the primary value does not lie with the requested material itself.

Reference: IEA-24-027

Signature: Johan Andrew Iau le Roux **Date of Request:** 07 / 02 / 2024

4. Permission

☑ **Granted** ☐ **Denied**

Terms of agreement: Permission is granted for non-exclusive rights to reproduce the material requested above upon the terms and solely for the purpose indicated.

☑ **Free** ☐ **Agreed Fee** (payable to NL63ABNA0481961968)**:**

Signature: **Date:** 27 / 03 / 2024

Name: Dirk Hastedt

Title: Executive Director

Signature Certificate

Reference number: QELXV-NZG3Q-C9JHJ-SPMPB

Signer	Timestamp	Signature
Johan Andrew Ian le Roux Email: andrewleroux777@gmail.com		
Sent: Viewed: Signed:	26 Mar 2024 11:45:43 UTC 26 Mar 2024 17:06:44 UTC 26 Mar 2024 17:14:27 UTC	
Recipient Verification: ✔Email verified	26 Mar 2024 17:06:44 UTC	IP address: 105.184.216.66 Location: Potchefstroom, South Africa
Dirk Hastedt Email: d.hastedt@iea.nl		
Sent: Viewed: Signed:	26 Mar 2024 11:45:43 UTC 27 Mar 2024 13:17:12 UTC 27 Mar 2024 13:17:46 UTC	
Recipient Verification: ✔Email verified	27 Mar 2024 13:17:12 UTC	IP address: 212.123.229.166 Location: Amsterdam, The Netherlands

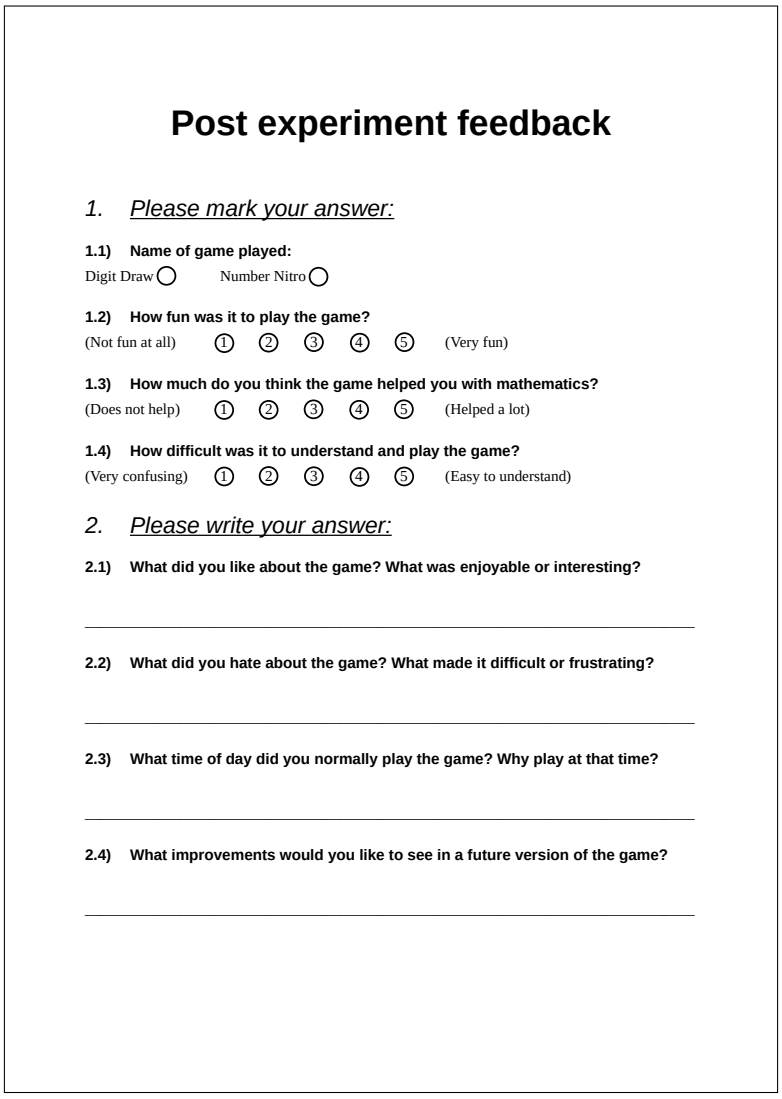
Document completed by all parties on:
27 Mar 2024 13:17:46 UTC

Signed with PandaDoc

PandaDoc is a document workflow and certified eSignature solution trusted by 50,000+ companies worldwide.

Appendix G

Post test survey



Post experiment feedback

1. Please mark your answer:

1.1) Name of game played:
Digit Draw ◯ Number Nitro ◯

1.2) How fun was it to play the game?
(Not fun at all) ① ② ③ ④ ⑤ (Very fun)

1.3) How much do you think the game helped you with mathematics?
(Does not help) ① ② ③ ④ ⑤ (Helped a lot)

1.4) How difficult was it to understand and play the game?
(Very confusing) ① ② ③ ④ ⑤ (Easy to understand)

2. Please write your answer:

2.1) What did you like about the game? What was enjoyable or interesting?

2.2) What did you hate about the game? What made it difficult or frustrating?

2.3) What time of day did you normally play the game? Why play at that time?

2.4) What improvements would you like to see in a future version of the game?

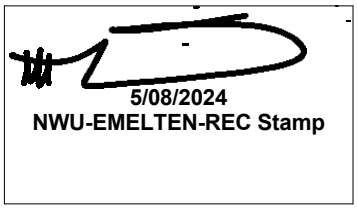
Figure G.1: Survey given to participants after the post test

Appendix H

Informed consent documentation

Private Bag X1290, Potchefstroom
South Africa 2520
Tel: +2718 299-1111/2222
Fax: +2718 299-4910
Web: http://www.nwu.ac.za

The Faculty of Health Sciences Ethics Office of the North-West University is acknowledged for the use of their document with minor adjustments made by the North-West University Education, Management and Economic Sciences, Law, Theology, Engineering and Natural Sciences Research Ethics Committee (NWU-EMELTEN-REC).



INFORMED CONSENT FOR PARENTS OF GRADE 9 CHILDREN PARTICIPATING IN THE MATHEMATICS GAME RESEARCH STUDY

TITLE OF THE RESEARCH STUDY: Development of a new serious game utilising reinforcement learning to enrich high school mathematics

ETHICS REFERENCE NUMBER: NWU-00407-24-A2

PRINCIPAL INVESTIGATOR: Prof. Pieter van Vuuren

POST GRADUATE STUDENT: Mr. Andrew le Roux

ADDRESS: Office 244, Building N1A, Faculty of Engineering, NWU, Potchefstroom

CONTACT NUMBER: pieter.vanvuuren@nwu.ac.za

Your child is being invited to take part in a research study that forms part of a Masters degree in Computer Engineering. Please take some time to read the information presented here, which will explain the details of this study. Please ask the researcher or person explaining the research to you any questions about any part of this study that you do not fully understand. It is very important that you are fully satisfied that you clearly understand what this research is about and how your child might be involved. Also, your child's participation is **entirely voluntary** and he/she is free to say no to

participate. If you or your child says no, this will not affect you or your child negatively in any way whatsoever. Your child is also free to withdraw from the study at any point, even if he/she does agree to take part now.

This study has been approved by the **North-West University Education, Management and Economic Sciences, Law, Theology, Engineering and Natural Sciences Research Ethics Committee (NWU-00407-24-A2)** and will be conducted according to the ethical guidelines and principles of Ethics in Health Research: Principles, Processes and Structures (DoH, 2015) and other international ethical guidelines applicable to this study. It might be necessary for the research ethics committee members or other relevant people to inspect the research records.

What is this research study all about?

- *We have designed a new digital game that aims to improve children's understanding of grade 9 Algebra in an enjoyable manner. The first objective of this study is to determine whether the game does improve their understanding of grade 9 Algebra.*
- *The non-human opponent in the new game has been trained by means of various machine learning techniques. The second objective of this study is to compare the various machine learning techniques against each other.*
- *The study will kick off with a maths test (which your child doesn't have to study for). This test will be taken after school in a classroom at your child's school in the presence of his/her maths teacher and the postgraduate student who designed the game.*
- *Your child will then be given the new game to play with in his/her free time on his/her phone for a period of two weeks.*
- *At the end of the two week period he/she will be asked to write another maths test in a classroom at your child's school in the presence of your maths teacher and the postgraduate student.*

Why has your child been invited to participate?

- *Your child has been invited to be part of this research because he/she is a grade 9 learner in a school in the North-West province.*
- *In addition, he/she should have access to an Android smart phone. The smart phone should preferably use Android 5.1 or newer and have 250 MB of free storage.*
- *Your child will unfortunately not be able to take part in this research if he/she is not in grade 9 and don't have access to an Android smart phone. (The iOS version of the game might only appear next year.)*

What will be expected of your child?

- *Your child will be expected to write a mathematics test at the start of the study. He/she don't have to study for this test. This test will be taken after school in a classroom at his/her school in the presence of his/her maths teacher and the postgraduate student who developed the game. The test will take between 30 minutes and an hour.*
- *Your child will then be given one of two maths games to play with in his/her free time on his/her phone for a period of two weeks. Your child should please stick with the specific game that we have given him/her, he/she will be given access to the other game after two weeks.*

- *While your child is busy playing the game, the following data will be stored and used for research:*
  - *The dates and times that he/she played the game*
  - *The choices that he/she made during the game*
  - *How many levels he/she started and completed in the game.*
  - *However no personally identifiable information will be collected.*
- *At the end of the two week period your child will be asked to write another maths test after school in a classroom at his/her school in the presence of his/her maths teacher and the postgraduate student. The test will take between 30 minutes and an hour.*

Will your child gain anything from taking part in this research?

- *The most direct benefit for your child if he/she takes part in this study will be that he/she will devote more time to mathematics for a two week period. This is a good thing since "practice makes perfect". Hopefully the game will be an enjoyable way to practice mathematics and lead to a small improvement of his/her math grades.*
- *If the game does improve his/her understanding of algebra we can improve and expand the game so that in the near future grade 9 children of the entire South Africa can improve their math grades in an enjoyable manner.*

Are there risks involved in your child's participation in this research and what will be done to prevent them?

- *There are no foreseeable risks for your child if he/she chooses to participate in this study.*

How will we protect your child's confidentiality and who will see his/her data?

- *Your child's performance in the game as well as his/her test scores will be kept anonymous with no personally identifiable information being made available to the researcher and student.*
- *During the course of the project the data will be kept safely on a secure server. Thereafter the data will be moved to a password-protected hard-drive and locked safely away in the researcher's office.*
- *Only the researcher and his student will have access to the data.*
- *The data will be kept for five years after which it will be permanently deleted.*

What will happen with the data or samples?

- *The data of this study will only be used for this study.*

How will you know about the results of this research?

- As soon as we've analysed the results of the study we'll communicate our findings to your child via the school's newsletter and a general WhatsApp message broadcasted to all grade 9 learners by the mathematics teacher. This will probably be towards the end of 2024.
- You will be informed of any new important information regarding this study by your child's maths teacher.

Will your child be paid to take part in this study and are there any costs for you?

This study is funded the university and there are no direct expenses for you or your child.

Refreshments will be served to the children after both math tests as a token of appreciation for those who took the time to write the tests.

Is there anything else that you should know or do?

- You can contact prof Pieter van Vuuren at pieter.vanvuuren@nwu.ac.za if you have any further questions or have any problems.
- You can also contact the North-West University Education, Management and Economic Sciences, Law, Theology, Engineering and Natural Sciences Research Ethics Committee via mz Paballo Molefe at 018 285 3037 or paballo.molefe@nwu.ac.za if you have any concerns that were not answered about the research or if you have complaints about the research.
- You will receive a copy of this information and consent form for your own purposes.

Declaration by parent

By signing below, I ……..…… agree to let my

child ……..…….. take part in the research study titled: "Development of a new serious game utilising reinforcement learning to enrich high school mathematics"

I declare that:

- I have read this information/it was explained to me by a trusted person in a language with which I am fluent and comfortable.
- The research was clearly explained to me.
- I have had a chance to ask questions to both the person getting the consent from me, as well as the researcher and all my questions have been answered.
- I understand that taking part in this study is **voluntary** and we have not been pressurised to take part.
- My child may choose to leave the study at any time and will not be handled in a negative way if he/she chooses do so.
- My child may be asked to leave the study before it has finished, if the researcher feels it is in the best interest, or if he/she do not follow the study plan, as agreed to.

Signed at (*place*) .....…..…….. on (*date*) ……...……. 20....

.....
Signature of parent

Declaration by person obtaining consent

I *(name)* ……..…… declare that:

- I clearly and in detail explained the information in this document to

  ……

- I did/did not use an interpreter.
- I encouraged him/her to ask questions and took adequate time to answer them.
- I am satisfied that he/she adequately understands all aspects of the research, as discussed above
- I gave him/her time to discuss it with others if he/she wished to do so.

Signed at (*place*) .....…..…….. on (*date*) ……...……. 20....

.....
Signature of person obtaining consent

Declaration by researcher

I *(name)* ……..…… declare that:

- I explained the information in this document to the grade 9 children who showed interest in the study.
- I did not use an interpreter
- I encouraged them to ask questions and took adequate time to answer them. I also was available should any want to ask any further questions.
- The informed consent was obtained by an independent person.
- I am satisfied that all participants adequately understand all aspects of the research, as described above.
- I am satisfied that all participants had time to discuss it with others if any one of them wished to do so.

Signed at (*place*) .....…..…….. on (*date*) ……...……. 20....

.....
Signature of researcher

Appendix I

Ethical clearance

Private Bag X1290, Potchefstroom
South Africa 2520

Tel: 018 299-1111/2222
Fax: 018 299-4910
Web: http://www.nwu.ac.za

Faculty of Humanities Research Ethics Training and Support

North-West University Education, Management and Economic Sciences, Law, Theology, Engineering and Natural Sciences Research Ethics Committee (NWU-EMELTEN-REC)

Tel: +27 18 289 2899
Email: Erhabor.Idemudia@nwu.ac.za

Senate Committee for Research Ethics
Tel: 016 103 4446
Email: Feziwe.Mseleni@nwu.ac.za

5 August 2024

Dear Prof van Vuuren

ETHICS APPROVAL LETTER OF STUDY

Based on approval by the **North-West University Education, Management and Economic Sciences, Law. Theology, Engineering and Natural Sciences Research Ethics Committee (NWU-EMELTEN-REC)** on **5/08/2024**, the NWU-EMELTEN-REC hereby approves your study as indicated below. This implies that the NWU-EMELTEN-REC grants its permission that, provided the general conditions specified below are met and pending any other authorisation that may be necessary, the study may be initiated, using the ethics number below.

Study title: Development of a new serious game utilising reinforcement learning to enrich high school mathematics.

Study Leader/Supervisor (Principal Investigator)/Researcher: Prof Pieter Van Vuuren

Student: Mr Andrew le Roux (31765157)

Ethics number:

N	W	U	-	0	0	4	0	7	-	2	4	-	A	2
Institution				Study Number						Year			Status	

Status: S = Submission; R = Re-Submission; P = Provisional Authorisation; A = Authorisation

Application Type: Single-Study

Commencement date: **5/08/2024**

Expiry date: 5/08/2025

Risk: **Low Risk**

Approval of the study is initially provided for a year, after which continuation of the study is dependent on receipt and review of the annual (or as otherwise stipulated) monitoring report and the concomitant issuing of a letter of continuation.

General conditions:

While this ethics approval is subject to all declarations, undertakings and agreements incorporated and signed in the application form, the following general terms and conditions will apply:

- *The principal investigator/study supervisor/researcher must report in the prescribed format to the NWU-EMELTEN-REC:*
  - *twelve-monthly on the monitoring of the study, whereby a letter of continuation will be provided annually, and upon completion of the study; and*
  - *without any delay in case of any adverse event or incident (or any matter that interrupts sound ethical principles) during the course of the study.*
- *The approval applies strictly to the proposal as stipulated in the application form. Should any amendments to the proposal be deemed necessary during the course of the study, the principal investigator/study supervisor/researcher must apply for approval of these amendments at the NWU-EMELTEN-REC, prior to implementation. Should there be any deviations from the study proposal without the necessary approval of such amendments, the ethics approval is immediately and automatically forfeited.*
- *Annually a number of studies may be randomly selected for active monitoring.*
- *The date of approval indicates the first date that the study may be started.*
- *In the interest of ethical responsibility, the NWU-EMELTEN-REC reserves the right to:*
  - *request access to any information or data at any time during the course or after completion of the study;*
  - *to ask further questions, seek additional information, require further modification or monitor the conduct of your research or the informed consent process;*
  - *withdraw or postpone approval if:*
    - *any unethical principles or practices of the study are revealed or suspected;*
    - *it becomes apparent that any relevant information was withheld from the NWU-EMELTEN-REC or that information has been false or misrepresented;*
    - *submission of the twelve-monthly monitoring report, the required amendments, or reporting of adverse events or incidents was not done in a timely manner and accurately; and/or*
    - *new institutional rules, national legislation or international conventions deem it necessary.*
- *NWU-EMELTEN-REC can be contacted for further information via Ethics-EMELTEN-apply@nwu.ac.za or 018 285 3037*

The NWU-EMELTEN-REC would like to remain at your service and wishes you well with your study.

Please do not hesitate to contact the NWU-EMELTEN-REC for any further enquiries or requests for assistance.

Yours sincerely,

Prof Erhabor Idemudia
Chairperson NWU-EMELTEN-REC

Original details: (27271005) C:\Users\27271005\OneDrive - North-West University\EMELTEN DATABASE\EMELTEN-REC\EMELTEN\EMELTEN-REC letters\3. Approval Ethics Letter\ETHICS APPROVAL LETTER.docm
2 September 2023

File reference: 1

Appendix J

Software package versions

As mentioned in Chapter 4, the designed agent is implemented with the ML-agents framework of the Unity game engine. However, the stability of the ML-agents framework is dependent on the software versions that are integrated with it. This study used ML-Agents release 20 (Unity Package 2.3.0-exp.3) together with Unity 2022.3.34f1. The Python (3.8) packages employed by the framework are provided in Table J.1.

Table J.1: Python packages and versions used

(a)

Package	Version
absl-py	2.1.0
alembic	1.13.1
attrs	23.2.0
bottle	0.12.25
cachetools	5.3.3
cattrs	1.5.0
certifi	2024.2.2
charset-normalizer	3.3.2
cloudpickle	3.0.0
colorlog	6.8.2
filelock	3.13.1
google-auth	2.28.1
google-auth-oauthlib	1.0.0
greenlet	3.0.3
grpcio	1.62.0
gym	0.26.2
gym-notices	0.0.8
h5py	3.10.0
idna	3.6
importlib-metadata	7.0.1
importlib_resources	6.1.2
joblib	1.3.2
Mako	1.3.2
Markdown	3.5.2
MarkupSafe	2.1.5
mlagents	0.30.0
mlagents-envs	0.30.0
numpy	1.21.2
oauthlib	3.2.2
optuna	3.5.0

(b)

Package	Version
optuna-dashboard	0.14.0
packaging	23.2
PettingZoo	1.15.0
pillow	10.2.0
pip	22.3.1
protobuf	3.20.0
pyasn1	0.5.1
pyasn1-modules	0.3.0
PyMySQL	1.1.0
PyYAML	6.0.1
requests	2.31.0
requests-oauthlib	1.3.1
rsa	4.9
scikit-learn	1.3.2
scipy	1.10.1
setuptools	65.5.1
six	1.16.0
SQLAlchemy	2.0.28
tensorboard	2.14.0
tensorboard-data-server	0.7.2
threadpoolctl	3.3.0
torch	1.8.2+cpu
torchaudio	0.8.2
torchvision	0.9.2+cpu
tqdm	4.66.2
typing_extensions	4.10.0
urllib3	2.2.1
Werkzeug	3.0.1
wheel	0.42.0
zipp	3.17.0

Appendix K

System requirements of the game

The game used in the experiment can only be played on an Android smartphone. The smartphone should preferably use Android 5.1 (or newer), contain 1 GB+ RAM, and have at least 250 MB of free storage available.

Appendix L

Specifications of computer used to train RL agents

The specifications of the computer system that was used for training the RL agents are given below. It should be noted that training was done only on the CPU and was not GPU accelerated.

CPU: AMD Ryzen 5600G

RAM: 2x16GB DDR4 Geil Orion 3600MHz CL 18-22-22-42

Motherboard: MSI A520M-A PRO

Storage: Crucial P3 1TB PCIe Gen3 M.2 NVMe SSD

OS: Fedora Linux 38