Agent Based Temporal Difference Learning for Risk

CS907 MSc Computer Science

Edward Pasfield

Supervisor: Dr. Matthew Leeke

2018-19

Abstract

Reinforcement learning is becoming more and more important in everyday lives, from self driving cars to health-care. This use of widespread of reinforcement learning (RL) technology has caused massive improvements in the field due to the frequency of use. There are many variations and developments within RL based on the specific intended use of the training model. This paper explores the use of it within an unsolved strategy turn based game environment. It should highlight the improvements possible for the other agents within the game. The results of this will provide an in-depth analysis on the practical implementation and its positives/negatives within such an environment.

Keywords: Reinforcement Learning, Temporal Difference, Deep Learning, Machine Learning, Risk, Agent, Multi-Agent System

Acknowledgements

I would like to thank particular individuals for their support and contribution to the project's success.

Dr. Matthew Leeke, has been vital to the projects progress his support and advice throughout has been invaluable. At every stage he was willing to meet and provide important and necessary feedback.

Jamie and Sarah Pasfield and Jade Wanden for support throughout the last year, without which there would not be a project to complete.

Ptolemy Jenkins, Harry Plucknett and Annie Eldhose.

Abbreviations

A3C	Asynchronous Advantageous Actor Critic
AI	Artificial Intelligence
CNN	Convolutional Neural Network
CTMS	Centralised Trust Management System
DL	Deep Learning
DRL	Deep Reinforcement Learning
DQN	Deep-Q Network
IDE	Integrated Development Environment
OMG	Object Management Group
PTL	Parallel Transfer Learning
RL	Reinforcement Learning
TD(L)	Temporal Difference (Learning)
UML	Unified Modelling Language

Contents

1 Introduction

A reinforcement learning AI trainer which has improved the way that current agents strategies play Risk will help highlight how agents learn and can perform in a complex game environment. The specific area of the game that has been successfully trained, is the initial troop allocation stage. This is because of how important it is in the long term and how it increases the probability of winning (which will be shown). These agents and their corresponding strategies will be tested against each other, with this new trained agent to see if it improves the way they play. Graphs generated by copious amounts of data collection will lead to a careful analysis of the trained strategy and how it improves agents existing strategies. This will reflect the competitiveness between the agents and how the trained strategy improves upon gameplay. The data that will be outputted by the trainer will likely be game states along with the probabilities that they win based on a set of rules at that point in time. The obtained source code for Risk is from GitHub and has 8 agents that implement a set of strategies based on heuristics already implemented [2]. This, in turn, will allow a bigger focus on the training and how it can affect the performance of these agents.

The game environment being used is a multi-agent system (MAS). Before describing a MAS the concept of an agent needs some explanation. There is an assumption that the agents are intelligent. This means that they can choose actions based on game states and heuristics to achieve a goal or perform a set of actions [3]. A MAS is where two or more of these agents communicate and act with each other within an environment. They evaluate their relationships whilst deciding what actions to carry out to achieve the said goal [4]. There are many intricacies when dealing with a MAS and this has proved challenging for multiple practical AI learning techniques, self-driving cars for example. The main problems come from a poor selection of which technique to carry out. The selection of an appropriate technique is a vital step in allowing the agents to cooperate or compete in different environments.

The structure of this project consists of Research, (Legal, Social, Ethical and Professional Issues), Requirements, Design, Implementation, Testing and Analysis, Project management followed by Results an Evaluation and a Conclusion. Further information will be available to help to understand of other more 'real-world' applications of multi-agent systems (MAS's) in general; such as transportation, coordinated defence systems and manufacturing. Along with a greater understanding of Risk and reinforcement learning.

1.1 Motivation

The motivation for this is the research and practical application for reinforcement learning with it becoming such a key component of companies and since 2006 has grown rapidly, redefining state-of-the-art performances in a range of fields such as image segmentation, speech/object recognition and health monitoring [1].

1.2 Project Aims

The primary aim of this project is to see if an improvement can be made to the existing agents after training some of the game to an enhanced level. This will be achieved by studying the game environment and carefully selecting which aspect of the game to train. One with a small set of game states that can be correctly identified and can be allocated probabilities to show the likelihood of achieving a goal.

The secondary aim of this project is to understand the game complexities of Risk and be able to explain why the trained strategy helps improve the existing agents/strategies and to debate the possibility of improving the agents in other ways.

1.3 Stakeholders

The main stakeholders for this project are the supervisor Dr. Matthew Leeke and the researcher, Edward Pasfield. There is a keen interest in this project and its success based on the interest in the field and the amount of time that has been put into it. There is a hope that the research will help aid in the tactful application of allocating of resources for more than just the game environment it was made for.

2 Research

The research section is split into three categories, the game environment, the competitive/cooperative strategies for the agents and the learning methods. The game environment is the less academic of the three but gives an important insight into the inner workings of Risk. The other two are the primary research areas and are split between game theory being competitive/cooperative strategies, and machine learning being the learning methods.

2.1 Game Environment

This section will provide some background into the game risk and the way it will relate to the learning method along with providing the rules of the game to a good level as to give proper understanding for the references of the game throughout this project.

Risk was invented in 1957 by Albert Lamorisse (A French filmmaker). Hasbro commercialised this into the world domination strategy game we know today and it has flourished and become one of the most popular games of all time. Over the past 10 years, it has been made into many digital versions and explored more academically in terms of its game theory and strategies. The premise is to control all 6 continents and their corresponding 42 territories by taking turns to attack and defend through dice rolls. Players usually have to make alliances to help them defeat their common enemies and grant them a stronger hold over their territories, this must be done tactfully to be effective [6]. Trust as a social construct is one of the harder concepts to formalise and replicate if it is to be taken into account in the learning process and implementation of the learning method. The agents will have to perceive 'the most powerful' other agent (probably based on army size/territories/continents owned) and cooperate with the other agents to even the odds. There will be 2-6 agents tested as this is the number of players the game allows, fortunately, the open-source version from GitHub is suitable for this project and the application of a learning method due to the command line execution reducing playtime dramatically from up to 8 hours to less than 30 seconds. Even more, fortunately, this version already has some strategies already built-in for different stages of the game for individual agents which implement them in slightly alternating ways. The agents built-in already are as follows; Angry, Greedy, Continental, Furious, Focused and Steady. These will be tested thoroughly as to give a basis for the learning method to improve upon. The current best agents will be identified and used to help the new AIAgent train to perform the best it can. The game can also be run visually which will give insight and understanding of gameplay along with giving context to anyone who does not know the game/code well.

2.1.1 Rules of Risk

The start of the game begins with the "setting up" of the board; the players take turns allocating a single troop onto empty territories claiming them for themselves until all 42 territories are owned. After this, the players still in their same turn order distribute their remaining troops one at a time onto the territories they own. After this, the board is set up for a game to be played and each player performs the four actions on their turn in order; Trading Stage, Placing Stage, Attack Stage and Fortifying Stage [30].

1. Trading: If a player wins at least one attack and takes a territory from another player they earn a card for that turn. There are three types of cards; infantry, cavalry and artillery. If a player has a set of three they can trade in during this phase and collect a bonus number of troops to allocate in the placing stage. The possible sets and their corresponding bonus troops are as follows: 3 - infantry (6 Troops); 3 - cavalry (8 Troops); 3 - artillery (10 Troops), 1 - infantry, cavalry and artillery (12 Troops). If a player has 5 or more cards they must trade-in.

2. Placing: During this stage, the player will decide how to distribute their troops onto the territories they own. The number of troops they have to allocate will depend on how many territories/continents they own. They get a minimum of 3 troops per turn with a bonus troop per every three territories they own. They also receive a continent bonus if they own all the territories within a continent (Australia - 2, Africa - 3, South America - 3, North America - 5, Europe - 5, Asia - 7). This all in addition to the troops they received in the trading stage. The player can choose to allocate any number of troops to whichever territory they wish.

3. Attack: After all the troops have been allocated to a player's territories they may attack adjacent territories owned by other players. The level of success of these attacks is based on dice rolls. The attacking player and defending player roll dice simultaneously and the number of the attacker's dice that are strictly greater than the defender's dice is the number of troops the defender loses. However, the number of the attacker's dice that are equal or less than the defender's dice is how many troops the attacker loses. Furthermore, the amount of dice the attacker/defender can use differs. The defender can use a maximum of 2 dice to defend and can only do this if they have two or more troops, if they have one troop they must use a single die. The attacker has a maximum of 3 dice and can only use this amount if they have more than 3 troops, if they have exactly 3 troops they may use 2 dice and if they have 2 troops they must use 1 dice. The only other thing to take into account is that the attack and defender order their dice rolls from high to low before comparing them, highest number against high-

est number and so on. An attack may make as many attacks as they wish/are able within this stage. Once the defender has 0 troops in a territory the attack has won and can move as many troops as they want to the territory.

4. Fortifying: Once a player has chosen to end their attacking stage they may move a single set of troops they own onto another connected territory through other territories if they are owned. They may move a subset of the troops they have in the original location.

Risk is a complicated game and the technical gameplay needs a large amount of insight to be provided. The Figures below (1,2) are the interconnections for the map which have been formalised into a clear network and the probabilities of winning or losing fights based on the dice roll statistics. This is vital for providing strategies and deciding which territories/continents are important and give greater control of the map by being owned.

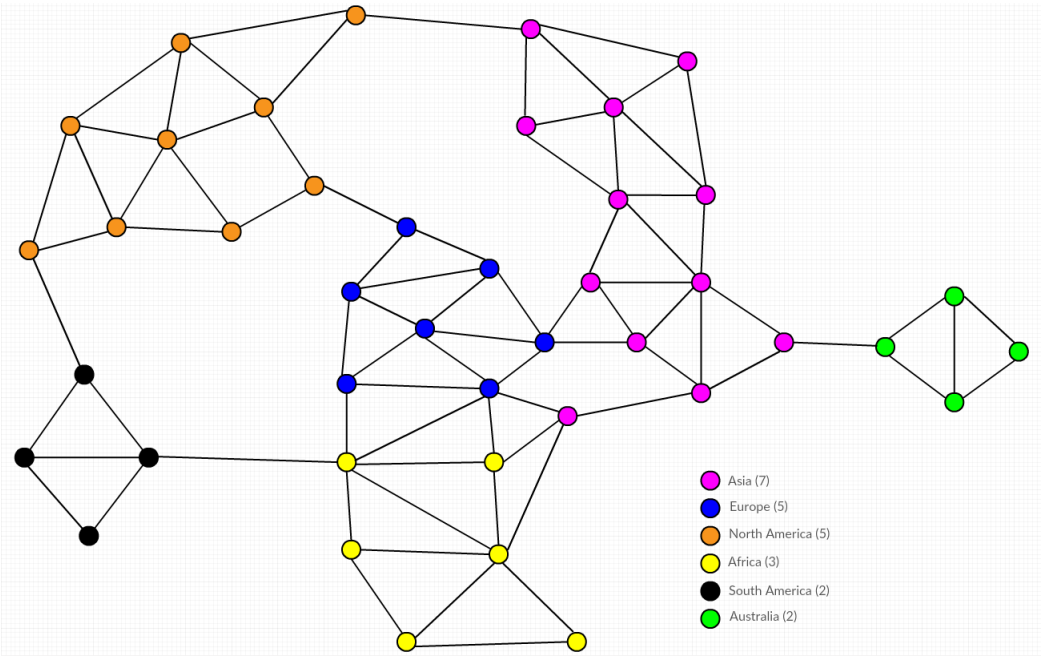

Figure 1: Risk Map Network [7]

Secondly, Figure 2 gives the likelihoods of losing troops when attacking, this data allows for more precise combat strategies which may help in stacking fights in a specific agents way. However, this will never be a certainty due to the nature of the dice rolls randomness.

Defender Rolls One Die			
Number of dice rolled by attacker	1	2	3
Defender Loses One Troop	41.7%	57.9%	66.0%
Attacker Loses One Troop	58.3%	42.1%	34.0%
Defender Rolls Two Die			
Number of dice rolled by attacker	1	2	3
Defender Loses One Troop	25.5%*	22.8%	37.2%
Attacker Loses Two Troop	74.5%*	44.8%	29.2%
Both Lose One Troop	NA*	32.4%	33.6%

* Only one troop Lost

Figure 2: Probabilities of Dice Rolls [7]

Now that there has been a decent level of background given to understanding the game environment the project can now move on into the research of competitive/cooperative strategies for these agents that will be playing the game. There will be some issues which prove very hard to formalise and make decisions on most likely because of the large number of possible game states at every turn of the game.

2.2 Competitive/Cooperative Strategies

When researching strategies there were two main areas that were highlighted, competitive and cooperative. When debating what in particular to train, the game environment will be broken down and related to game theory strategies and techniques. These will be analysed to see whether they will be incorporated into the learning method. The cooperative and competitive nature of the agents will need to be balanced very carefully because even though an agent may be helping you when it gets to late game you will still be enemies. This section will rely heavily on game theory. Strategies will require a large proportion of unpredictability so others cannot play a dominating best response strategy.

This section relies quite heavily on game theory and has directed the research into two main categories of strategies competitive/cooperative. These will have to be implemented together as even though another agent may be helping you to achieve a bigger goal it is still competition. Strategies that would try to incorporate both competitiveness and cooperativeness will require a certain amount of unpredictability to ensure other players cannot incorporate counter strategies [8]. The concept of looking at strategies and making sure they do as well or as badly as possible leads nicely into two main game theory strategies; maximin and minimax. Maximin assumes the best worst-case payoff will be the agent's decision [9]. Minimax, in contrast, minimises the maximum possible loss. Both strategies restrict the number of troops an agent will lose from a defensive perspective [9]; the more defensive of the two being minimax. An example of a minimax strategy

within Risk would be if an agent realised another was about to receive a continent bonus and instead took one of the territories within the said continent to stunt there maximum payoff. If this was implemented into Risk and the existing built-in agents the learning method will train to stunt the bonuses of other agents. If this is implemented and tested with other agents it may cause significant issues; for instance, if it works and trains correctly and all agents enforce its strategy there is a likelihood that an equilibrium will form and no agent will win. This will be further discussed if it is incorporated into the learning process.

When dealing with a MAS one of the biggest issues especially in the context of Risk will be the decision-making process that focuses on the trust level between agents. In normal Risk players base their actions on their trust of players. It will be very interesting to see at certain points when agents decide to betray their allies for their own gain. Trust does lie at the centre of all interactions between agents that are operating within changing environments [10]. The previous research on trust led to a few methods which provided possible ways of dealing with the formalisation. The first was fuzzy filters which were an approach for social modelling in this context. It analyses the data being shared between agents, however, after more research it became clear that the method would have had to of been evolutionary [11]. This meant that the agents would hold grudges between games. This was the primary reason why fuzzy filters were disregarded as a possibility due to the nature of agents needing to adapt their alliances between games and even more specifically turns depending on their individual states.

Secondly, some research into how to formalise this has led to a comprehensive trust management system (CTMS). This would allow for the development of a level of trust for each of the other agents each turn. This is ideal because alliances may change dramatically between turns. To implement this into a learning method first the CTMS will have to be implemented into the existing agents which will deal with the social side of gameplay and will be vital in performance. The implementation acts using three main sub-components; evaluate, establish and engage [12]. Firstly, evaluate, based on the interaction history evaluates the trustworthiness of other agents. Secondly, establish, which identifies the decisions and resources needed to seem trustworthy to other agents. Finally, engage, which assumes rationality for agents to decide and carry out actions with a desired outcome. Overall this would allow decision paths to be carried out based on the trust models they have learned from other agents [12].

Figure 3 below depicts how this CTMS would implement three sub-components which fit in with a generalised MAS. This diagram explicitly defines how a CTMS would be optimally incorporated.

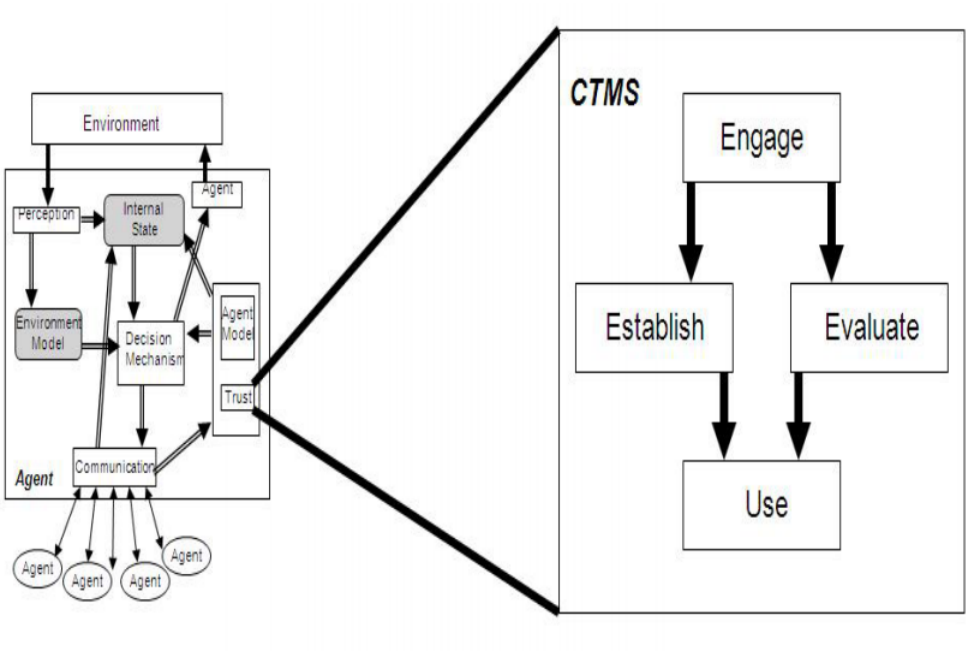

Figure 3: Principal components of the embedded trust module in an agent architecture [12]

As shown in Figure 3 the CTMS is connected and will rely on a decision mechanism and a communication module. This will control the data between agents and will determine the accuracy to which the strategies are executed.

When delving into relating levels of trust between agents, there are five main types that will be implemented into the CTMS if the decision is made to train the new agent on trust rather than a more offensive/defensive strategy. These five types are as follows [13]:

- CopyCat: Start with cooperate then copy whatever opponent did in last round.
- Always Cheat: Never trust anyone.
- Always Cooperate: Trust everyone.
- Grudger: Start cooperative with everyone and if betrayed once then never trust that agent again.
- Detective: Analyze what the other agents are doing, and act accordingly. (may be hard to implement in Risk due to the longevity of the game and alliances that might not be broken until near the end.)

If a trust learning agent is implemented the strategies that are executed will be analysed during learning to attempt to find which ones work most effectively even more so alongside the minimax and maximin strategies.

The research carried out so far along with how the agents will behave has been thorough and a great understanding of how agents work has been looked into. There are many agent strategies that are implemented in the code already. Whilst debating what to feed the trainer research is imperative however what is just as important is the way the game works and these pre-implemented strategies. The next step will look at these individually and explain decisions based on the more specific aspects of the code.

2.2.1 Built in Strategies

This subsection will briefly describe all the different strategy types and which agents use which strategies. It is important to understand the differences between the agents and their corresponding strategies so that there is more context behind the results section and which agents do better than others.

- AggressiveStrategy : This strategy is controls the agent during the combat phase and has certain 'aggressive' behaviours which include; always attacking, always choosing the maximum number of troops, always move the maximum number of troops forward and fortify a significant amount of troops to the front line.
- AIContinentsStrategy : This strategy is used to aid the AIAgent in picking a continent to aim for in the "setting up" phase. It incorporates the choice attribute which selects which continent to aim for based on the probabilities provided by the trainer.
- ContinentsStrategy : This strategy is used to proved all agents with picking a continent based on which one has the least amount of free territories in the "setting up" phase.
- CaptureContinentsStrategy : This class makes the agent pick where to place their troops when it is not the initial allocation.
- CardsStrategy : This trades in cards during the trading phase as soon as a set becomes available, it then tries to pick a promising attack and carry it out. Especially if there has not been a successful attack this turn yet, to ensure the agent receives a card.
- ControlledStrategy : This deals with the placement of troops and allocates them equally along the territories with an enemy owned territory adjacent to them.

- PassiveStrategy : This strategy only trades in cards when there is more than 5. It also chooses never to attack and to defend only using 1 Dice.
- RandomStrategy : This executes random moves based on the decisions that are possible. Attacking random territories and placing random troops along with choosing random amount of defending dice and where to fortify. It is as random as possible.
- SmartAggressiveStrategy : This is similar to AggressiveStrategy with the only difference being it always attacks when there is an advantage.

Each of these strategies is used in specific agents with different combinations leading to different agents with different aims. This leads well onto the next subsection being the agents themselves and which strategies they use.

2.2.2 Agents and their Corresponding Strategies

The multiple different agents have been created through the use of the strategies previously discussed. The research into these is what has given the information to make some high level decisions on which ones to consider training. They will be listed below in a table (Figure 4) with the strategies listed that each agent employs in green and each strategy it does not employ in red.

Agents/Strategies	Aggressive	AIContinents	CaptureContinents	Cards	Continents	Controlled	Passive	Random	SmartAggressive
AIAgent									
AngryAgent									
ContinentalAent									
FocusedAgent									
FuriousAgent									
GreedyAgent									
PassiveAgent									
RandomAgent									
SteadyAgent									

Figure 4: Agents and their Strategies]

This table does clearly show that certain strategies are only used by certain agents. For instance, AIAgent is the only agent to make use of AIContinents which is the strategy which incorporates the trainer. On the other hand, SteadyAgent is the only agent to use the controlled strategy which is interesting because when tested it turns out to be one of the best-performing agents.

2.3 Learning Methods

The next best thing to look into is the possible machine learning methods that would be able to teach the agents in this specific system. Due to the unknown nature of the outcome of moves unsupervised learning is the most appropriate. Unsupervised learning is when an algorithm presents a possible decision to be executed. Whereas supervised learning is not appropriate as the agent must fully know the best decision and predict whether it is a legitimate action or not based on what has previously been categorised as a move [14].

When researching unsupervised learning methods for the agent to improve their current strategies the logical starting point was Deep Reinforcement Learning (DRL). It is the most well-known breakthrough in machine learning for teaching agents and has been remodelled in many ways for many different environments. One of the original uses of DRL was when it was utilised by OpenAI's Atari playing process to a better than human-level performance. It uses DRL in concordance with convolutional neural networks (CNN's) to learn control policies for successfully playing multiple Atari games [15]. DRL is a nice balance of two separate learning methods reinforcement learning(RL) and deep learning(DL). RL is a machine learning method which enables an agent to perform a set of actions by trial and error. The payoffs of these action sets (territories) are then treated as experience and are gathered and to calculate a possible reward based on future possible game states which in turn aids in the decision making process [16]. DL, on the other hand, is a model which is made of multiple hidden layers and learns through extracting representations of data using varying levels of abstraction [17]. An example of this is image recognition would have the power to conceptualise simple facts such as 'a dog has four legs' and use that with other facts to determine what type of animal an image is. AI = DL + RL is a common notation to explain the mix of techniques and how balanced it is leading the way to the future of AI and is regarded by many as the most promising at this current point in time [18]. This research however promising has been successful for single agents within many different game environments and only successful for multiple agents to a certain extent. Both of these single agents and multiple agents do have their own issues [19].

RL itself has been such had such an attraction in the field of machine learning because of the learner being able to learn by itself; without the aid of an intelligent "teacher" [36]. One downside of RL is that sometimes the reward is fed back after x number of actions and the wrong actions can be rewarded by mistake in this situation; this is called the temporal credit assignment problem [36]. One of the ways around this problem is temporal difference(TD) learning. TD predicts the probability of an action being either positive or negative towards the agent itself and then allows the agent to choose an action based on these probabilities

[37]. TD occurs between a state transition and the following update is applied:

$$v^{\pi}(s) = v^{\pi}(s) + \alpha(r(s) + \gamma v^{\pi}(s') - v^{\pi}(s)) \text{ [45]}$$

- where $\alpha \in [0, 1]$ is a confidence parameter: the value of new data.
- $v^{\pi}(s)$ and $v^{\pi}(s')$ are the values of the previous state and the next state.
- γ is the expected sum of rewards (in this case $= 1$).

TD would be useful if the set of game states were comprehendible and the probabilities of each of the outcomes could be calculated and fed back to aid the next decisions the agent would make. This would be an applicable learning method if the agent were to learn which territories/continents to take rather than how to attack effectively. This is a logical assumption based on TD being a learning procedure specialized for prediction problems [38].

The primary issues for DRL and most learning methods is getting stuck in local optima and overfitting to a pattern that is not prefered [20]. DRL managed to beat one of the best DOTA 2 players in which OpenAI by overcoming these issue and using this process to learn specific game nuances all within two weeks of training time [21]. This is an example of just how successful this learning method can be. There are variations of DRL which has improved upon the original, two of these include A3C and DQN. Firstly, Advantage Actor-Critic (A3C) was a possible alternative because of its ability to train multiple agents. It updates the value function and policy through a feed-forward view only after a specific state is reached or a set number of decisions have been carried out. A3C stabilises by executing multiple agents in their own environment instance [23]. This also steers away from CNN's which is positive because of the environments text nature. Secondly, Deep-Q Networks (DQN) is a method that learns from high-dimensional sensory data to enforce successful policies [22]. This method managed to learn to play 49 Atari games to beyond average human capable levels and overcome the issues highlighted previously stabilising the nature of DRL. DQN can only do this because of the way it incorporates experience replay and it iteratively updating an action value (Q value) [21]. DQN is most likely more appropriate due to its single-agent nature. The decision over whether to incorporate a single agent trainer or a multi-agent trainer is important and will be discussed after the research into both have been explored. DQN and A3C would be much more applicable if the agent was learning from visual representations of the Risk map however this strays from the sort of learning method to get a multi-agent system working.

When exploring multi-agent learning methods in a few separate articles there is a suggestion to move on from DRL and suggests transfer learning as the most effective. The first case that became well known was PathNet; there was an easy comparison with this and DRL due to the exploitation of the same game environments. The comparisons show that PathNet was more successful in terms of training times and performance. Google DeepMind developed PathNet and trained agents to complete a task using the knowledge it learnt from a prior task. This transfer of knowledge is the main principle of this learning method. There is a claim that this is the first step to achieving artificial general intelligence [24]. There are a few positives of transfer learning, its main one being that it allows parameter reuse, with no catastrophic forgetting. PathNet was originally used to train a single agent but after a while the possibility to use it to train a MAS became apparent. It does, however, come with its complications. Fortunately, these can be overcome by enforcing restrictions [25]. The multi-agent transfer learning method put forward was called Parallel Transfer Learning (PTL). PTL improved learning by allowing the simultaneous learning between target and source tasks. This, in turn, means that an agent would learn from their peers throughout their own learning process, and vice versa [26]. PTL would be an effective method but would need to be edited for the game hugely. There would have to be very specific classes devoted to sharing information between agents at many stages throughout each turn, let alone each game.

The most promising of these techniques discussed so far was temporal difference learning because of its simplicity with updating probabilities based on previous decisions made. It also was very logical based on the types of data the agents were already dealing with such as which continents to aim for and other simple tasks. Out of the other techniques the next most applicable were PTL followed by A3C as a close third. Theses unlike temporal difference learning have DRL a strong underlying technique but handle attributes such as (outputs, actions and inputs) differently. A step back will be taken from teaching multiple agents instead one will be taught and will just be competed against by others. TDL will be the learning method incorporated due to the ease of use in such a complex environment and the room it has to be developed into more. The suggestion made in the paragraph describing TD was to teach the agents which continent to aim for in the "setting up" phase. This seems like a logical place to start; it has a simple set of possible game states and has the theoretical ability to be very effective. The key to multi-agent learning is the way the methods run multiple tasks. If the system was trying to teach multiple agents PTL would have been the most appropriate to the game environment because of its shorter learning times and its peer to peer learning [5] along with making use of the knowledge it will learn from taking territories to taking continents. A negative to this is the large amount of computing power it would require.

3 Ethical, Social, Legal and Professional Issues

This section highlights the possible issues that may occur throughout the project in other areas than just technical issues. It explores and discusses the ethical, social, legal and professional issues instead.

There is a high level of standards that must be met when ensuring that a software project meets the satisfaction of stakeholders. The possible issues are outlined below in concordance with the steps to mitigate them. These issues will adhere to the British Computing Society Code of Conduct [27].

3.1 Ethical Issues

Ethical issues that might arise due to this project are minimal due to its scope. However, one that does occur is the need to have regard for the legitimate rights of third parties [50]. Where the term third parties include any group or individual that might be affected by the developer's activities. Another important issue to keep track of is to ensure to conduct professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, etc. This means that because the code will be kept open source it must be kept open source to all groups of people.

3.2 Social Issues

There are many possible social issues with machine learning in general. The trustworthiness of the implementation in a day to day environment such as a hospital is poor. Not only the trustworthiness of the system but the fact that jobs will be reduced in the short term once the system is up and running. However, on this specific project, there is the potential to exploit the game of Risk by performing better than usual whilst using this agent. An example of this would be placing money on the agent under the pretence of fair play.

3.3 Legal Issues

The possible legal issues will be copyright infringement and plagiarism. Both of which will be avoided by adhering to the Data Protection Act 2018. However, the Risk game downloaded used has been downloaded from GitHub and is open source and has been made available under the GNU General Public License. The project report has been referenced when necessary to ensure the rights of the words used are accredited to their source.

3.4 Professional Issues

There is room for a few professional issues within this project. The one most likely to occur is submitting the project as open source. This needs to be done as the code originally was open source and should, therefore, be submitted as such. This keeps with the developer's professional values.

4 Project Requirements

The purpose of this section is to identify the risks, assumptions and needs of the project in a clear measurable way. The success of lots of software development can be down to how many of the requirements it meets. The requirements will be split into two sections functional and non-functional. These requirements will be discussed in the evaluation and a clear yes/no will be given as to explain whether they have been met or not as a sign of how successful the project was, and what work is still left to do.

4.1 Functional Requirements

Functional requirements help capture the theoretical behaviour of the system. These will describe the functions that the system will carry out. Usually consists of inputs, behaviour and outputs.

1. The system **must** be able to play through a game without any issues/bugs.
2. A Temporal Difference Learning algorithm **must** allow an agent to be taught how to perform these strategies.
3. Once training is complete the method it learnt **must** be able to be applied to any agent.
4. The trainer **must** output a list of all possible continents that the agent can aim for and the matching probabilities of winning based on which continent it chooses.
5. The trainer **must** improve on the existing agent it is applied to.
6. A working CTMS **must** be incorporated to allow the evaluation of trust levels between agents.
7. An output of agents and their trust levels **must** be output so analysis can take place.
8. Data **must** be output after every 5-10 games to show progress in the learning process.
9. The visual representation of the game **must** work with the agent's strategy and perform without bugs.

4.2 Non-Functional Requirements

Non-Functional requirements are essential to allow the correct use and performance of a system. They usually set standards that the system must meet or need to perform to an adequate level; these are usually more hardware based.

1. Hardware **must** be good enough to host the temporal difference learning/ all possible states that can be chosen.
2. The system **should** make certain security issues are not compromised when editing the source code.
3. The system **should** run in a near identical amount of time with before/after the algorithm is applied.
4. The CTMS **should** be supported by the system running Risk.

4.3 Hardware and Software Constraints

There are no hardware or software constraints. This is because of the decisions made on the learning method. There would be a constraint if the development of the trainer got past the point of just teaching the agent to play the initial placement phase. The hardware constraint would be largely limited by the processing a large number of possible game states. There is however a supercomputer that the University of Warwick allows students to use for research purposes. This would help process the number of game states hugely. Repeated algorithms would also be run much faster as it could be split among cores, this would mean that training times would become much more efficient.

5 Design

This section is divided into two sub-sections; conceptualisation and processes. Conceptualisation discusses the high-level steps that have been taken to achieve the implementation of the program and the training of an agent through temporal difference learning.Then the processes section will cover the design methodology and give examples of how this was used practically whilst developing the system. Along with a class diagram to help lead the design process to a more practical understanding. A few tests and results were collected to help take the design even further to a point where the implementation could then begin.

5.1 Conceptualisation

Firstly, when making some design decisions about how to improve these agents through temporal difference learning the decision to focus on troop placement had already been made. The only decision left to make was at what points throughout the game to implement this. A lot of time was spent formalising how many game states would need to be stored. For instance, if the agent was to be trained to focus on which territories to focus on every turn the number of game states to be stored would have been:

$$n = \#\text{ of agents}, T = \#\text{ of territories} \Rightarrow \quad n^T$$

This assuming n=3 and T=42 for a normal board, the number of game states that would need to be held would be $3^{42} = 1.094 \times 10^{20}$ which is too large for the hardware available to process. After realising this a step back was taken to formalise this in a simpler way. What was decided is that to begin with the trainer would learn which continent to aim for during initial troop allocation; which could be further developed into which continent to aim for at every turn. The data on which continent to aim for has to be edited to allow a normal probability distribution based on prior runs to help aids its decision on which to aim for in the next run through.

However, the amount of game states that would need to be stored for which continent to aim for at initial troop allocation is very simple its:

$$a = \#\text{ of agents being trained}, C = \#\text{ of continents} \Rightarrow \quad C^a$$

Under the assumption that only one agent is being trained the possible amount of game states being stored is 6^1. This unlike the previous is number is a good starting point for the training process as it can be developed further. This development leads into which continent should the agent aim for at every turn. Where

this formula becomes dramatically larger per turn:

$$t = \#\text{ of turns},\ a = \#\text{ of agents being trained},\ C = \#\text{ of continents} \Rightarrow\ \ (C^a)^t$$

This number will depend on the turn count, as there is a possibility of a game lasting days this could also be very complicated. Unless a game boundary was put in place which said if the turn count reached over a certain limit, for example, 60. Then the game would terminate. Even then the number of possible states would be 6^60 also a number too large to deal with. If however, t was turned to the number of turns divided by 10 and the agent was trained to aim for a continent every 10 turns this would be a much more realistic number and is likely to be able to be dealt with. This will be an extension if the first training of initially allocating troops is successful with enough time to spare.

5.2 Processes

Figure 5 below is the software development methodology which is vital to the way the program will be tested and produced. It is an agile methodology which fitted best because of its iterations between deliverables and feedback which will reduce the number of bugs and changing of requirements. It would also reduce the cost of overruns if it applied to this project, which in industry is the most common reason for software development failure. The iterations are useful in this context due to the number of small decisions which need to be made throughout development; most of which had feedback given from the supervisor. Testing functionality like this at different stages and receiving feedback allows for a very flexible development process. Unfortunately, the main negative of Agile is the fact its very labour intensive but as this was already a given for this project it did not matter. A couple of other methodologies were considered, waterfall being one. However, because of its rigid nature led to agile being the final decision. There is a likelihood that the requirements will change because of the small scale of the project which solidified the decision between agile and waterfall [28].

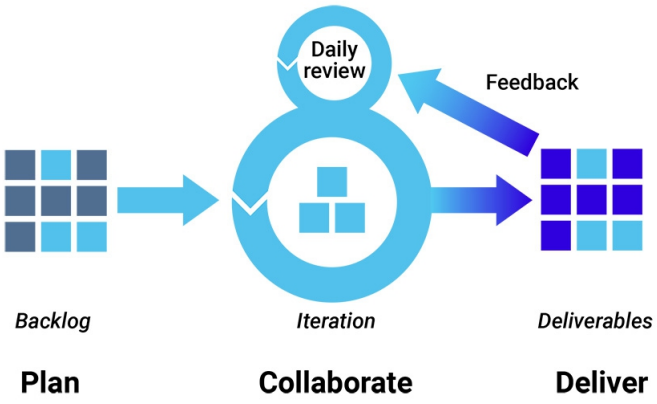

Figure 5: Design Methodology [29]

Now that a design methodology has been chosen and a good starting point for the learning has been decided a more thorough explanation of what decisions were made and why whilst programming will be discussed. To understand the structure of the code from a high level it is important to understand how all of the classes relate. The main way of dealing with understanding the complexity of software systems is to use models expressed in unified modelling language(UML) [31]. UML describes a high-level abstraction of a system [32]. A perfect UML tool for understanding the static system structure/architecture is a class diagram [33]. There are two main variations in notations for class diagrams, ours shall use that put forward by the Object Management Group (OMG) in 1997. This is the notation that would provide a better performance [34]. Figure 6 is the class diagram for the Multi-playerRisk file after the trainer has been added. This makes the relationships between classes and how the trainer fits in very clear and provides the majority of the attributes that are used in each class. This was made using an online diagram tool called visual paradigm [35].

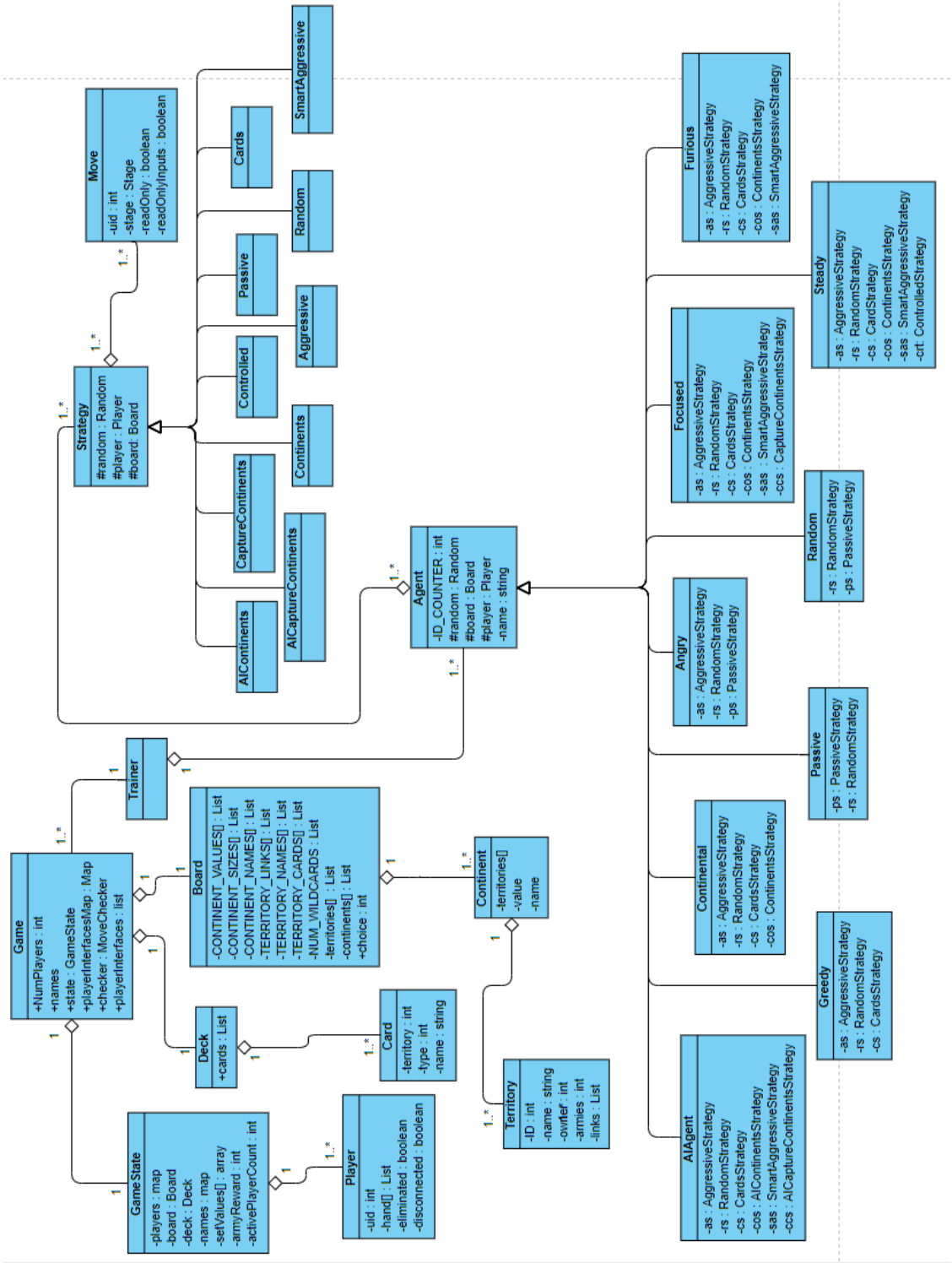

Figure 6: Class Diagram

This class diagram is quite complicated with lots of data but hopefully will make understanding references to the code and/or specific classes a lot easier. Once this was drawn up the next step was to figure out which agents to use in the learning process. It was important to pick the best one to develop upon as this one is likely to perform best. The way to test which agent and corresponding strategies it employs was the best performer the game was run 1000 times with the agents that execute the most strategies all competing against one another. A tally was kept for how many times each agent won. Fortunately, there was a clear winner and runner up. Figure 7 below shows this data in the form of a bar graph. Focused won with 494 wins and Steady came in second place with 303. It would make sense that the fewer the agents on average the faster the game and as these two agents were the best by a significant amount. It was then logical to train the agent on the Focused agent, in a game environment with an untrained Focused agent and a Steady agent for a difference in strategy and some decent competition.

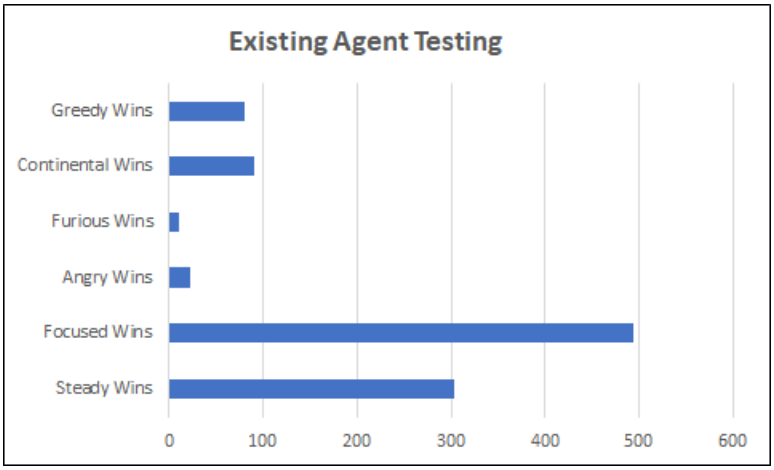

Figure 7: Existing Agents and their Win Rates

Sufficient high-level decisions have been made on the design of the program/trainer. The way the trainer will be implemented has been conceptualised in the form of a class diagram. These decisions/conceptualisations lead onto starting software development. The progress made will be shown to Matthew Leeke at small intervals to provide feedback and new implementation goals following the agile software development methodology. The program will train an agent to at first make a decision on which continent to aim for at the initial troop allocation; if possible will be developed into which continents to aim for at each turn. This leads onto the implementation section.

6 Implementation

This section discusses the steps taken at a more practical level within the program and in the context of Java; it may help to familiarise oneself with some Java syntax if it is not well known. There was a large proportion of time spent understanding the code and the way classes passed variables between one another.

The First step taken was recreating the agent class to its separate AIAgent, and corresponding strategy; AIContinentsStrategy, this was necessary to have a separate version of the Focused Agent to train. This was done by re-using the code existing classes and tailoring their class names and adding a variable to the Board class called choice and having the strategy use this as the territory to initially aim for instead of just aiming for the one with the least amount of empty territories in it (the normal ContinentsStrategy Focused uses). This choice variable was implemented and could be chosen at random through the use of the Java function ThreadLocalRandom to find assign the variable a random integer correlating to one of the continents. Once this randomly assigned continent was working and the agent would aim for it during the "setting up" of the board a meeting was had with the supervisor to get feedback and discuss other possible options.

The second step was to create the class trainer which would run the game multiple times with just the AIAgent, Focused Agent and Steady Agent playing. It would iterate through games and output how many times each Agent wins. This was achieved by recreating the WatchCLI class but manually assigning the userAgent to be the AIAgent and commenting out all the other possible agents from within the classes AgentFactory and AgentTypes, which is where the agents are built (Figure 8). When the variable numAI was set to 3 within trainer this meant the first one was AIAgent every game and the other two were a mix between Focused and Steady. Once this step had been achieved a second meeting for feedback was had with the supervisor who suggested was pleased with the progress made. There was a separate AIAgent which could pick which continent to aim for and had competition with the two other best agents which would hopefully benefit the training process a significant amount. The fact the game could be run multiple times was also a great step as this meant data collection could be started as well as being very promising for allowing the trainer to run through as many epochs as needed.

```
//NUMBER OF AI's
int numAI = 3;
writer.format("Loading game with %d AIs\n", numAI);

int nextPlayerID = 0;
List<IPlayer> players = new ArrayList<~>();
Agent userAgent = AgentFactory.buildAgent(AgentTypes.Type.AI);
CommandLinePlayer user = new CommandLinePlayer(userAgent, reader, writer, userAgent.getName(), nextPlayerID++);
players.add(user);
//unit test 4.0 test whether the agent factory has correctly made the user based on the AIAgent
//System.out.println(" Unit Test 4: " + user.getPlayerName());
for (int i = 0; i != numAI - 1; ++i) {
    Agent agent = AgentFactory.buildAgent(AgentTypes.Type.valueOf(agentList.get(i)));
    AgentPlayer ai = new AgentPlayer(agent, nextPlayerID++);
    players.add(ai);
}
```

Figure 8: Loading the correct Agents into the Game

The third step was to make the trainer pick a continent based on probabilities defined in a file. The probabilities represent the likelihood of winning based on which Continent was aimed for by the agent. As there are 6 continents it made sense that each one was given an equal probability of 0.167. The file is shown in the format below in Figure 8. The Java built-in Scanner within a try-catch statement allowed for line by line reading of this file and the Scanner function substring() made extracting the probabilities from the 12th character position onwards. Each of these probabilities was then added to a list called probabilities. This list was used to individually sum the probabilities giving bounds for each continent choice between 0 and 1 at 0.167 intervals (0.167,0.334,...,1) (Figure 9). Once these bounds were defined, simple if/else if statements were incorporated to allow a random double between 0 and 1 to randomly select one of these continents for the agent to aim for based on their evenly spread probabilities.

```
List<Double> probabilities = new ArrayList<>();
try {
    File file = new File( pathname: "RiskRewards.txt");
    Scanner scanner = new Scanner(file);
    while (scanner.hasNextLine()) {
        probabilities.add(Double.valueOf((scanner.nextLine().substring(12))));
    }
    scanner.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
}

double A = probabilities.get(0);
double B = probabilities.get(1)+probabilities.get(0);
double C = probabilities.get(2)+probabilities.get(1)+probabilities.get(0);
double D = probabilities.get(3)+probabilities.get(2)+probabilities.get(1)+probabilities.get(0);
double E = probabilities.get(4)+probabilities.get(3)+ probabilities.get(2)+probabilities.get(1)+probabilities.get(0);

//Integration test 3 Test to see if bounds match to the probabilities and finish on 1
System.out.println("Integration Test 3: " + A + " : " + B + " : "+ C + " : "+ D + " : "+ E);
double r = ThreadLocalRandom.current().nextDouble( origin: 0,  bound: 1);
```

Figure 9: Read Probabilities from a file and Incorporate their bounds [0-1]

The fourth and final step was two parts, the first was to update the probabilities. This was done by updating the probabilities list within the Trainer class only when the AIAgent won using the List function set() adding 0.05 to the continent's probabilities that were correlated to the agents choice variable and subtracting 0.01 from all the other continents probabilities. The reason 0.05 and 0.01 was used was that this kept the total probabilities summing to 1 (Figure 10).

```
    // To Ensure no Overfitting in the first 125 epochs
    //Update the probability of the winning choice by 0.005 and the losing choices by -0.001 (This keeps the total at 1)
if (probabilities.get(game.state.getBoard().choice) <= 1 && probabilities.get(game.state.getBoard().choice) >= 0) {
    if (winnerid == 0 && totalWins <= 125) {

        probabilities.set(game.state.getBoard().choice, probabilities.get(game.state.getBoard().choice) + 0.005);

        for (int p = 0; p <= probabilities.size() - 1; p++) {
            if (p != game.state.getBoard().choice) {
                probabilities.set(p, probabilities.get(p) - 0.001);
            }

        }
```

Figure 10: TD for before 125 wins for the Agent (Exploratory)

A Java built-in writer was then utilised to update the file with these new probabilities which were then in turn read by the scanner in the third step and used to make new probability bounds in the next game iteration. These probabilities were then in a situation where they were learning from their past selves and updating with small increments based on the success of the agent's actions. It works out that:

α (being the confidence parameter) for each choice
$= (numberOfContinentWins)/(numberOfAIwins) \pm 1/(numberOfAIwins)$

Which meant the TD was now working. This was the first part and was successful, there was however an issue. If the trainer ran for more than a set number of times the probabilities would eventually become negative for certain continents. The second part was a solution to this, which was to keep a total number of AIAgent wins and the total number of wins for each continent choice and start updating the probabilities based on the continent choice wins divided by the total AIAgent wins (Figure 11).

```
for (int i = 0; i < probabilities.size(); i ++) {
    double c = ChoiceCount.get(i);
    double tot = totalWins;

    probabilities.set(i, (c / tot));
```

Figure 11: TD for after 125 wins for the Agent (Exploitative)

This resulted in another probability distribution over the continent choices. This second way was only implemented after the first 125 games as to give the Trainer class some time to make sure it explores a sufficient amount before starting to use this method which is more exploitation an important balance to keep to ensure that overfitting does not occur [20]. Figure 12 is the format of the possible continents that can be aimed for followed by the probabilities of the agent winning before training occurs. This is the format the trainer reads from and outputs to.

```
1:0:0:0:0:0 0.167
0:1:0:0:0:0 0.167
0:0:1:0:0:0 0.166
0:0:0:1:0:0 0.166
0:0:0:0:1:0 0.167
0:0:0:0:0:1 0.167
```

Figure 12: RiskRewards Format

The implementation of the temporal difference learning and weaving it into the code was complicated. This section chronologically went through the steps

taken in concordance with the agile development methodology. It followed the testing and feedback iterations. Based on the design section this was where the implementation would stop for now, until enough concrete data was collected proving that this updated probability distribution would improve the decisions made by the agents at during the "setting up" stage of the game. Once this data has been collected and analysed then more implementation may occur into allowing the agent to update probabilities based on continents throughout the game not just at the initial stage. The testing for each of these 4 steps is key to ensuring the agile methodology was carried out to a sufficient standard. There will be multiple testing sections based on the different parts of the code.

7 Testing and Success Measurement

This section will be split into two due to its importance in producing the data that will lead to the evaluation of the agent's performance.

7.1 Testing

Testing will be carried out on the program at different stages throughout the project to give as much insight as possible into well components work. Testing is a vital component of any project. It is responsible for the trustworthiness of the system and the data that is collected. Without testing the system may likely break unexpectedly and output false data [39]. Because of this, results that have been produced by an untested software cannot be trusted. Thorough testing will make the results as trustworthy as possible. testing will aim to prove the proper functionality of the system. Tests have been carried out in three different sub-categories unit, integration and system to make certain that the whole program has been covered [40]. These tests will all be run 10 times to make certain that the output is always the same, ensuring they are not only thorough but reliable. Testing will only be carried out on the code that has been added to this file, an assumption that the open-source code provided works has been made. The tests data will be in the form of a test table. Each test will have the attributes; test number, test description, input, expected output, output and result. The program will work 100% as intended if all of the outputs and expected outputs are the same which in turn will result in the test being a pass rather than a fail. The below tests are just a selection of varying tests which are believed to cover a large proportion of the functionality of the code implemented.

7.1.1 Unit Testing

Unit testing is a level of software testing where each 'unit' of software is tested individually. Its purpose is to ensure that the software is reliable and performs as intended [40]. A unit is a subsection of code which performs a single action, these will be the small chunks of code that are expected to be tested in unit testing. For example, reading probabilities from the RiskRewards.txt. Tests will be performed immediately after their implementation, this helps discover bugs and also gives useful feedback to help fix the issue. Figure 13 below is the test table for unit testing.

Unit Test Table					
Test Number	Test	Input	Expected output	Output	Results
1	The continent the AIContinentsStrategy selected should be the same as choice.	Integer variable choice	Two of the same integers in range [1,6]	4 4	**Pass**
2	The writer will write to UnitTest2.csv with the winning strategy AI 1.	Variable winner and its corresponding strategy	AI 1	AI 1	**Pass**
3	The array list agentList should be cut down to only STEADY and FOCUSED.	STEADY, FOCUSED	[STEADY, FOCUSED]	[STEADY, FOCUSED]	**Pass**
4	To check that the AIAgent has been made and implemented correctly by the agent factory.	AgentTypes.Type.AI	AI 1	AI 1	**Pass**
5	This tests whether the probabilities have been read and added from RiskRewards.txt to the probabilities list.	1:0:0:0:0:0 0.167 0:1:0:0:0:0 0.167 0:0:1:0:0:0 0.167 0:0:0:1:0:0 0.167 0:0:0:0:1:0 0.166 0:0:0:0:0:1 0.166	[0.167, 0.167, 0.167, 0.167, 0.166, 0.166]	[0.167, 0.167, 0.167, 0.167, 0.166, 0.166]	**Pass**
6	Test to check if the total win count totalwins adds up for each win.	A full game where AIAgent is the winner	1	1	**Pass**
7	This checks whether the conversion of the ChoiceCount to its double version c works.	ChoiceCount	A double value of ChoiceCount followed by an integer value of the equivalent number e.g. 55.0 = 55	31.0=31	**Pass**
8	This Checks whether the conversion of the totalwins to its double counter part is correct.	totalWins	A double value of totalWins followed by an integer value of the equivalent number e.g. 215.0 = 215	317.0 = 317	**Pass**
9	This checks to see if the probabilities were updated correctly.	A full game where the AIAgent is the winner and a +0.005 for whichever continent it chose and -0.001 for all the other continents probabilities.	[0.166, 0.166, 0.166, 0.166, 0.165, 0.171]	[0.166, 0.166, 0.166, 0.166, 0.165, 0.171]	**Pass**

Figure 13: Unit Test Table

As shown in Figure 9 the 9 tests of individual components and key functionalities all passed with no errors. This was useful information, it meant that the step towards integration testing could be taken with the certainty of the units involved working well.

7.1.2 Integration Testing

There was confidence in the individual components of the program after unit testing had been completed. There was an importance now to test how these interacted with one another. Integration testing is vital to provide certainty to the components connection functioning as intended [41]. This testing state will analyse the outputs and the inputs of these individual components to make sure there is a match between what is output and input for consecutive units. Without integration testing there could be inaccuracies in the outputs; this, therefore, ensures the accuracy of these outputs. When tested the more inner workings of TD there will be a level of uncertainty within the expected outputs. This is an existing problem with many machine learning algorithms. Figure 14 below is the integration test table, as done previously each test will also be run 10 times to provide even more reliability.

As shown in Figure 14 the 3 integration tests also proved the reliability of the cohesion between units. This provides a platform to support the System testing with the certainty of the Integrated units working.

Integration Test Table					
Test Number	Test	Input	Expected output	Output	Results
1	Test that the probabilities once they have been updated have a correct distribution (sum up to 1) before 125 AI wins.	List of probabilities	1 or equivalent	1.00	**Pass**
2	Test that the probabilities once they have been updated have a correct distribution (sum up to 1) after 125 AI wins.	List of probabilities	1 or equivalent	1	**Pass**
3	This tests checks the integration of the board boundaries set up for the choice variable and the probabilities.	List of probabilities	0.167 : 0.334 : 0.501 : 0.668 : 0.834	0.167 : 0.334 : 0.501 : 0.668 : 0.834	**Pass**

Figure 14: Integration Test Table

System Test Table					
Test Number	Test	Input	Expected output	Output	Results
1	Test that the system works properly and after going through all the steps it should update the trainers probabilites, along with following the strict algorithm for less than 125 AIAgent wins.	x number of games where agent wins < 125.	The probabilities of the penultimate game, the probabilities of the final game. The number of games and the nunmber of times the AIAgent won.	[0.162, 0.156, 0.192, 0.168, 0.155, 0.167] [0.161, 0.155, 0.197, 0.167, 0.154, 0.166] 10 8	**Pass**
2	Test that the system works properly and after going through all the steps it should update the trainers probabilites, along with following the strict algorithm for more than 125 AIAgent wins.	x number of games where agent wins >= 125.	The probabilities of the penultimate game, the probabilities of the final game. The number of games and the nunmber of times the AIAgent won.	[0.22666666666666666, 0.06222222222222222, 0.17333333333333334, 0.09333333333333334, 0.28888888888888886, 0.15555555555555556] [0.23008849557522124, 0.06194690265486725 6, 0.17256637168141592, 0.09292035398230089, 0.28761061946902655, 0.15486725663716813] 350 226	**Pass**

Figure 15: System Test Table

7.1.3 System Testing

System testing executes and inspects the output of the whole system. This test table is smaller but gives reliability to the data that is being collected and analysed in the Results section. It can only give this reliability due to the unit and integration tests that have come before it giving it a supporting platform of evidence that that system not only works but is cohesive and produces accurate results. Again the tests will be run 10 times to ensure that the results are true values.

As shown in Figure 15 the two tests have been successful and the trainer trains in two different ways, one for the first 125 AIAgent wins; and a second for more than 125 AIAgent wins. These two different training methods, therefore, work as two systems with a reliable nature and outcomes. This data means that a step forward onto data collection and the results based on these systems now can be taken with all soundly tested components.

7.2 Success Measurement

The success of this project cannot be formalised with anything to do with the probabilities. There are many different strategies that agents play and these can counter other strategies for several different reasons. There is too much of a social side to the game for the probabilities to be taken as fact that the agent does well. The success of this project will, therefore, come down to how well the agent improves upon the existing agents and how many of the other agents it does better than. The wider the range the better. There will also be a few different intensities of training to be tested against the other agents from 10,000 games to 30,000. This will be the level of exploration Vs exploitation wherewith more training (30,000 games) comes more exploitation whereas with a lower intensity of training (10,000) games brings more proportional exploration.

The developer will be testing the AIAgent with probabilities learnt after 10,000, 20,000 and 30,000s games which will provide these different levels of exploitation and exploration and how effective they are in the game environment. The different AIAgent's probabilities will be used for 5,000 games and the proportion of wins for different levels of training will be the true measurement of success. If the AIAgent wins more than the Focused Agent in the same situations then the training could be classed as successful due to this improvement.

8 Project Management

This section discusses the different methods which kept this project on track over the last 6 months. Firstly, a timeline in the form of a Gantt chart (Figure 16) which was updated at intervals depending on the progress of certain sections. It is an effective visual portrayal of when the project is due (Blue) and other four other sections which split up the project (Research, Software Development, Testing, Evaluation). These tasks have differed throughout the completion of the project. A Google calendar has been used to enforce this Gannt chart by setting alarms with a week and 24 hours notice before tasks should be started and finished

Secondly, meetings with Matthew Leeke have been weekly for an hour on average. This is on average because at different points throughout the project there has been less need for meetings than at other times. There has been near-constant communication with Matthew which has helped massively on keeping track of smaller tasks and the overall progress of the project. The meeting minutes have been taken, and emails saved to keep a note of meeting times and frequency.

Documentation of testing will occur throughout to make certain that the tests are providing proof of units working alone and together. A specific dissertation planner has allowed the track of notes and thoughts. As tests are performed the data will be saved in .csv files to make for easy data analysis. Constant online documentation will also be kept on google drive storing multiple different versions as a form of version control for security reasons and ensuring there is no loss of a significant amount of work at a critical time.

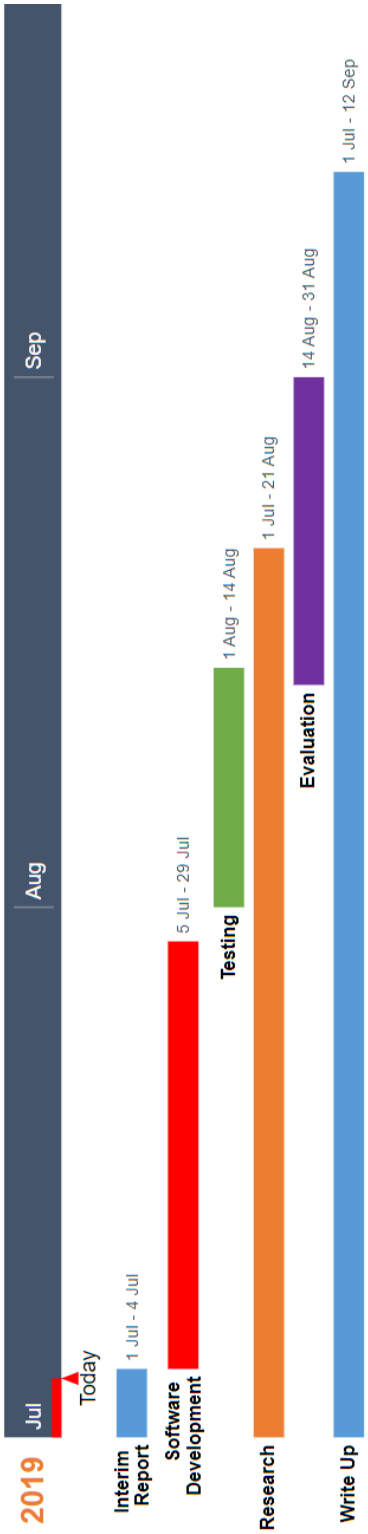

Figure 16: Gannt Chart

8.1 Tools

3rd party tools have been utilised throughout the project in a number of different ways. Each one has helped in their own ways increasing performance in terms of efficiency/ security. This section describes the use of each 3rd party tool in the context of the project.

8.1.1 Overleaf/Latex

Overleaf is an online Latex editor, it is free and has real-time collaboration which allows the Latex template that is used to help write and format the project. Latex is a typesetting system which is designed for producing scientific and technical documents[42]. It has kept the project very well organised and has enabled editing on many different systems without a physical backup.

8.1.2 IntelliJ

Intellij is an integrated development environment (IDE) which has allowed the Java program MultiplayerRisk-Master to be turned into a maven project and has given great support in the software development. It uses the information on the code to assist with looking for connections between classes and enables clever error analysis, fast navigation and refactoring [43]. This is also one of the developers preferred IDE's.

8.1.3 Google Drive

Google drive is free cloud storage, it has been used to keep versions of MultiplayerRisk-Master as it is very reliable and has 15GB of space. This large personal storage has allowed the developer access to the code on any system and has been useful throughout the development process [44].

8.1.4 GitHub

GitHub is a used globally for its ability to share and manage code. The developer used this to get the open source version of Risk which was used.

8.2 Risk Management

During the project it has been important to be wary of possible risks that could hinder the progress/success. Figure 17 suggests that bugs in the program are the most likely risk but are quite managable. Overall a low risk project.

Risk	Severity	Likelihood
Personal Illness	Varying	Medium
Supervisor Illness	Varying	Medium
Bugs	Medium	High

Figure 17: Risk Management Table

The lack of interaction with the public is what gives this project such a small table of risks. Fortunately, none of these occurred. The most likely to have any effect on the project was the possible varying levels of illness. These would have caused the schedule of the project to overrun. The length of which it would overrun is directly correlated to the severity of the illness. The risks of bugs were mitigated by having backups of the code for which if there was an issue a step back was available and another route may be taken.

9 Results

The agents that are built into MultiplayerRisk-Master have a possibility to be improved. The TD algorithm has had an impact on the performance of the agents. This impact is explored in this section. There is a range in training times and the agents used. It is important to understand the training process of each of the different lengths of training and to see how they slowly converge on the different probabilities of different continents. These convergences on continents show the best/most likely option. Some data analytical techniques have been used to show this clearly and concisely. The results when accumulated should provide a final conclusive answer to whether TD learning has improved the existing agents which will satisfy the main project aim.

The only metric that can be used to show improvement is the number of games won. This is a decisive continuous number which if greater proportionally to its predecessor will show improvement and the success of the algorithm. A brief explanation will be given for each Figure into the reasons behind the nature of the data.

9.1 AIAgent Results

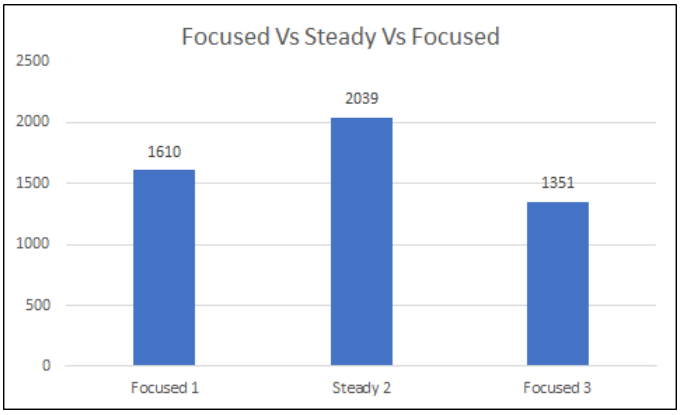

Figure 18: Base Agent Strategies : Wins out of 5000 games

Figure 18 shows the amount of wins each strategy wins over 5000 games. This gives a strong base for the AIAgent to improve from. This does show that the Focused agents roughly split their winnings and the Steady Agent therefore wins more. The reason why Focused 1 wins more suggests that the first player has a slightly better chance of winning because they get to pick a territory first.

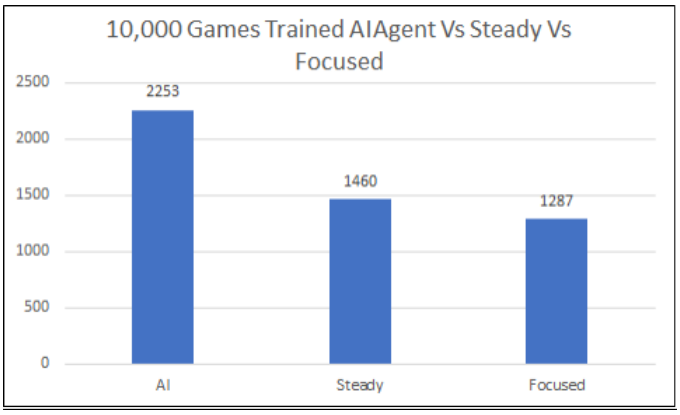

Figure 19: AIAgent Trained on 10,000 Games Vs Base Agents : Wins out of 5000 games

Figure 19 shows the amount of AIAgent wins after 10,000 games worth of training. This clearly shows a dramatic improvement on the base set where the Focused Agent won 1610 games; and this improved upon strategy has now won 2253; suggesting the training has been successful. The 643 wins that the AIAgent has gained seems to have come from the Steady agents pool of wins.

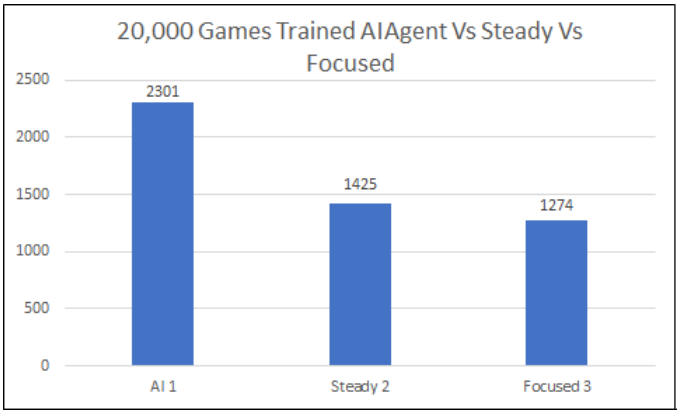

Figure 20: AIAgent Trained on 20,000 Games Vs Base Agents : Wins out of 5000 games

Figure 20 shows an improvement for an Agent trained on 20,000 games. This is an improvement on the original strategy a total of 691 extra games won. Interestingly these are similarly taken away from the Steady Agent rather than the Focused Agent. An even strong suggestion of success for the AIAgent's training.

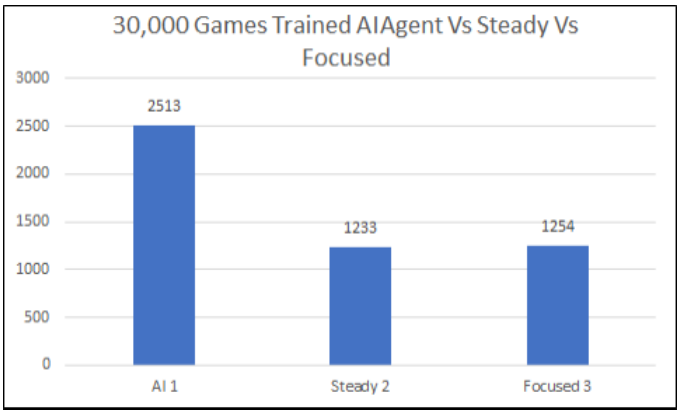

Figure 21: AIAgent Trained on 30,000 Games Vs Base Agents : Wins out of 5000 games

Figure 21 shows an even bigger improvement on the previous levels of trained AIAgent with 2513 wins. This has had an even bigger effect on the Steady Agent. The Focused Agent has lost 97 games before and after the AIAgent has been implemented and trained for 30,000 games. This suggests that the AIAgent is stunting the other Focused agents performance as well.

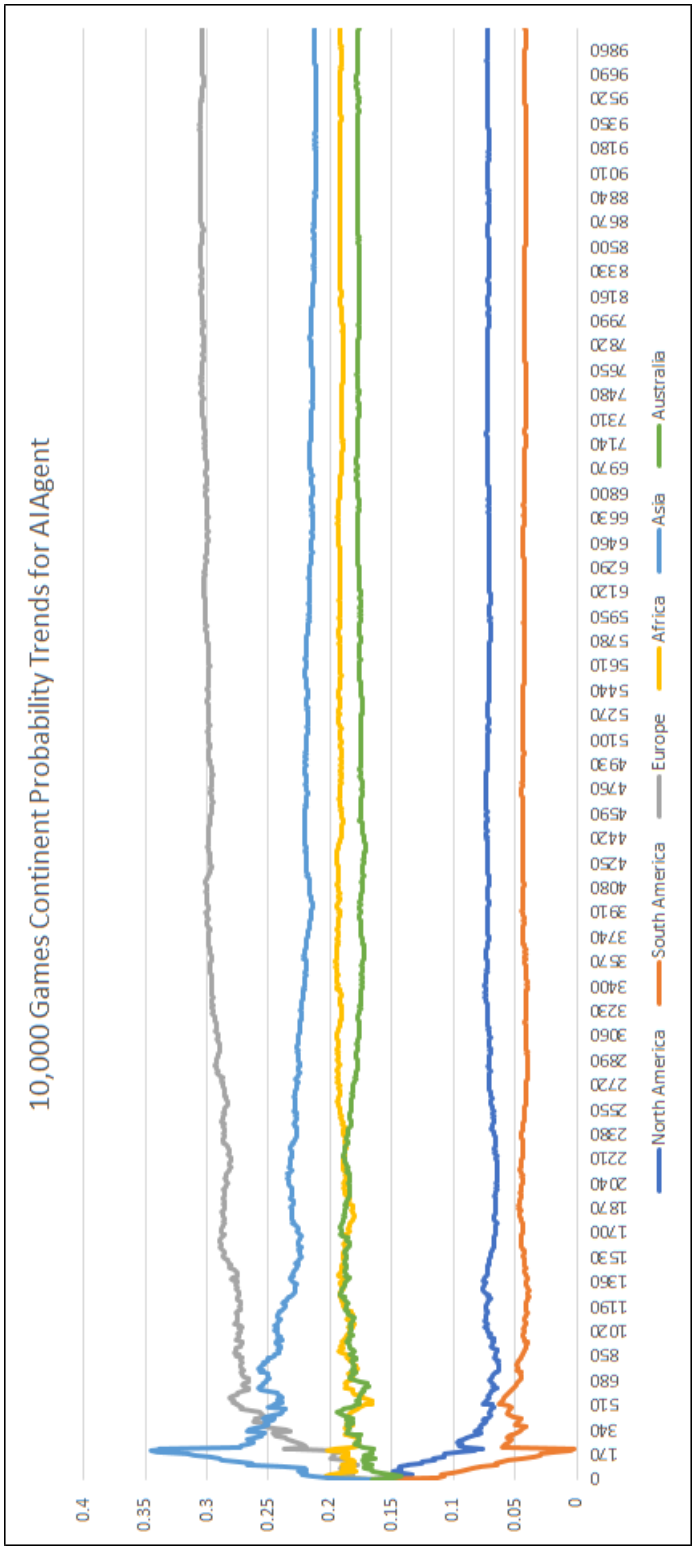

Figure 22: AIAgent Probability trends whilst training for 10,000 Games

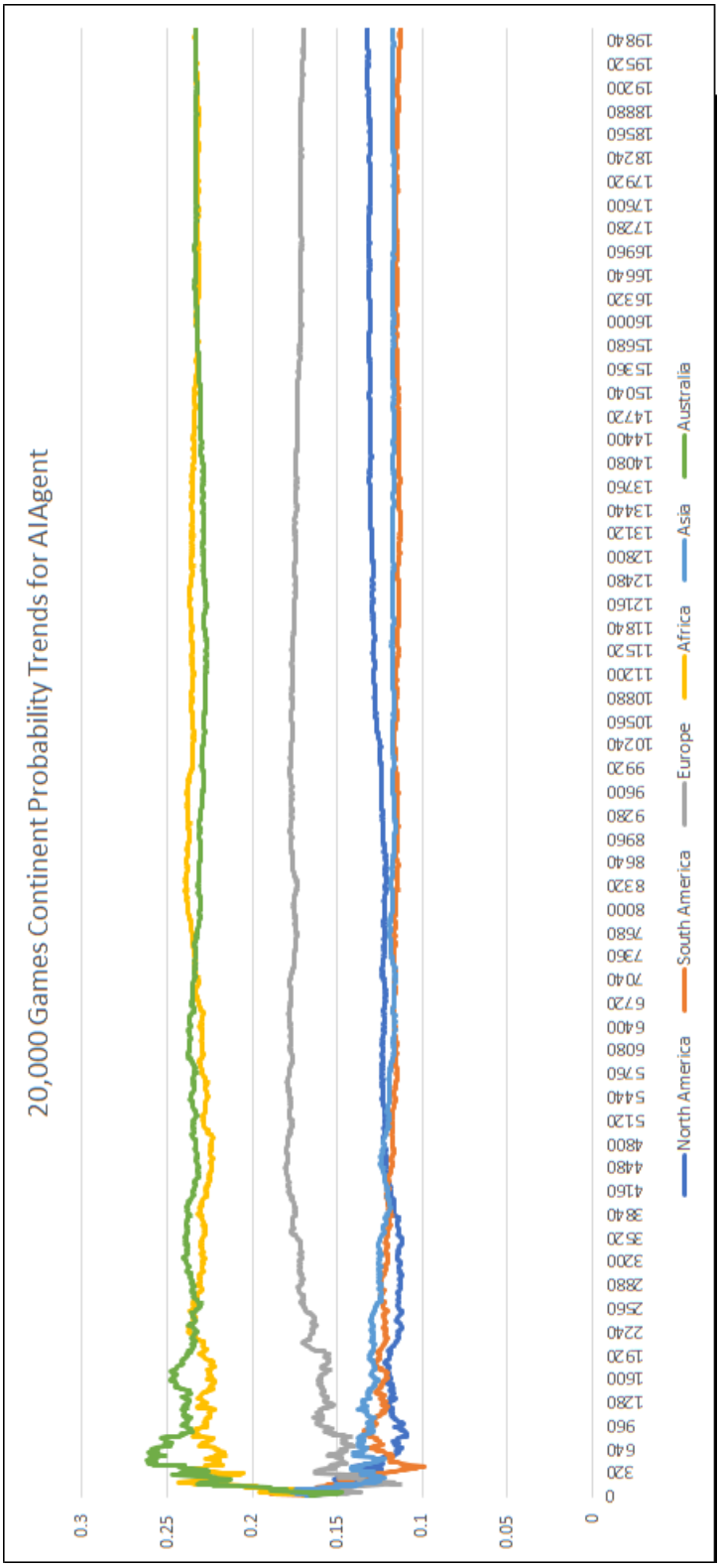

Figure 23: AIAgent Probability trends whilst training for 20,000 Games

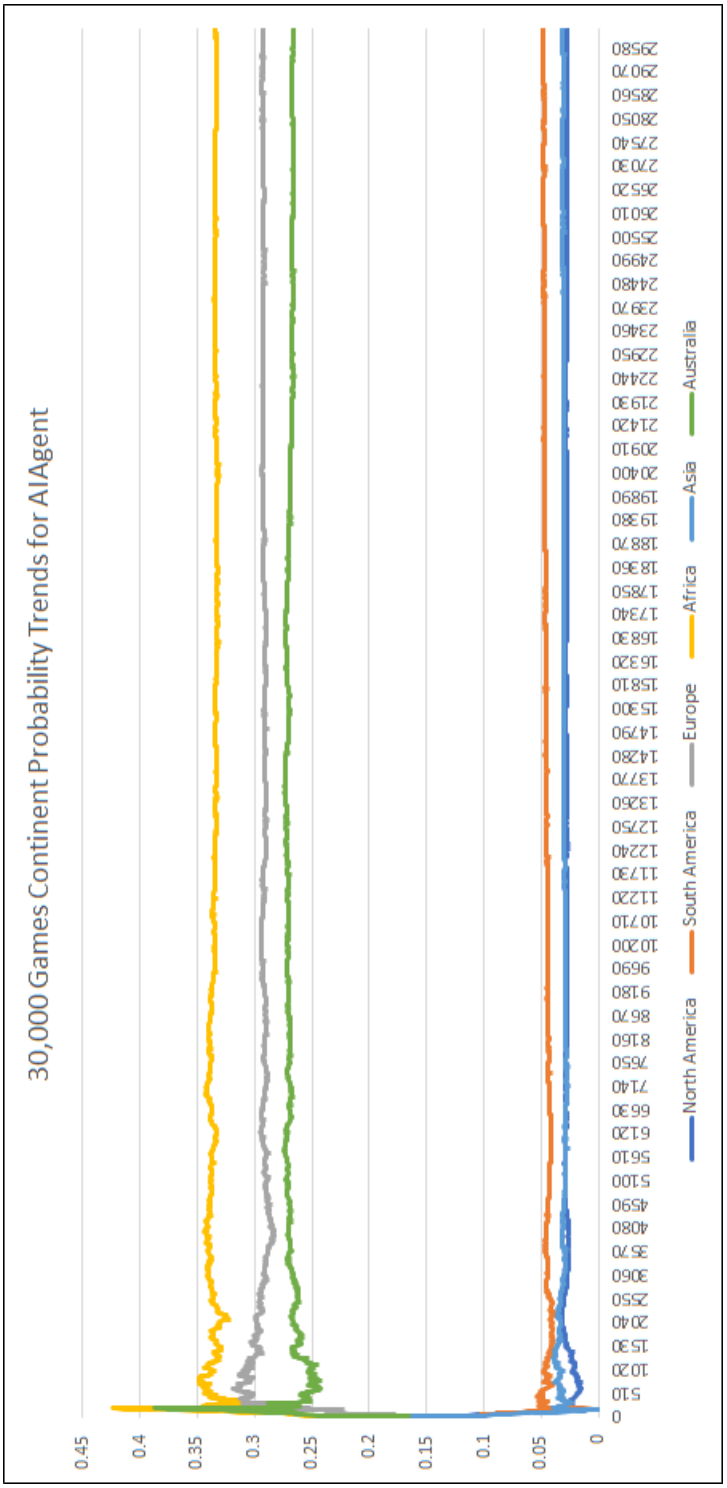

Figure 24: AIAgent Probability trends whilst training for 30,000 Games

Figures 22, 23 and 24 show the probabilities of the continents winning for the AIAgent throughout the training process allowing a visual representation of how the agents begin by exploring which ones work well and then plateau into their fitted best probabilities. In all three of these line graphs, it clearly shows that Africa is in the top three no matter how much training occurs. Whereas South America is near the bottom of all three graphs. The beginning proportion of roughly 3000 games shows the exploration the Agents take before finally starting to use their exploitation tactics to make the best of their situations. This suggests the TD and the corresponding α are working as intended as the algorithm explores and then exploits its possible options.

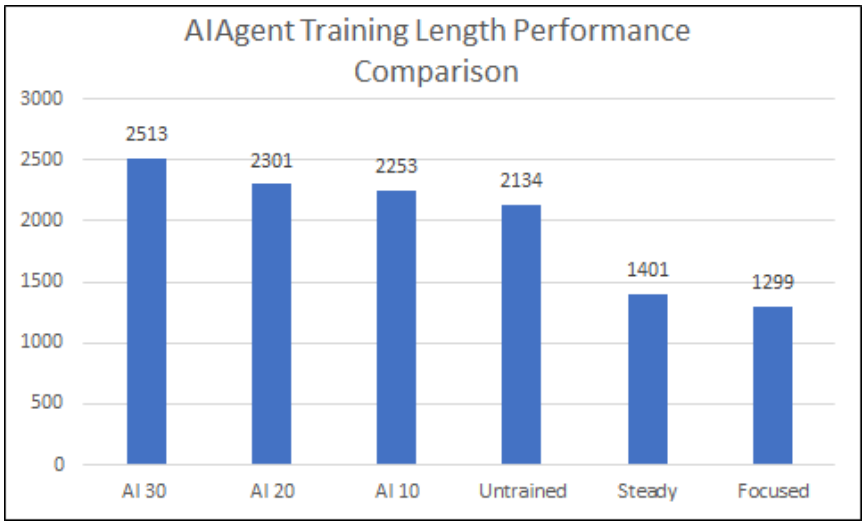

Figure 25: AIAgent Training Length Comparisons

Figure 25 shows a concise comparison between the AIAgent's performances and their Training lengths from not trained, to 30,000 games worth of training. As shown, the less training the worse the performance is the assumption from this data. It also shows the base Agents as a proportional comparison. The fact that the untrained Agent also improves on the previous Focused Agent suggests that some of the improvements the trainer made also came from an improvement of the way the Agent plays.

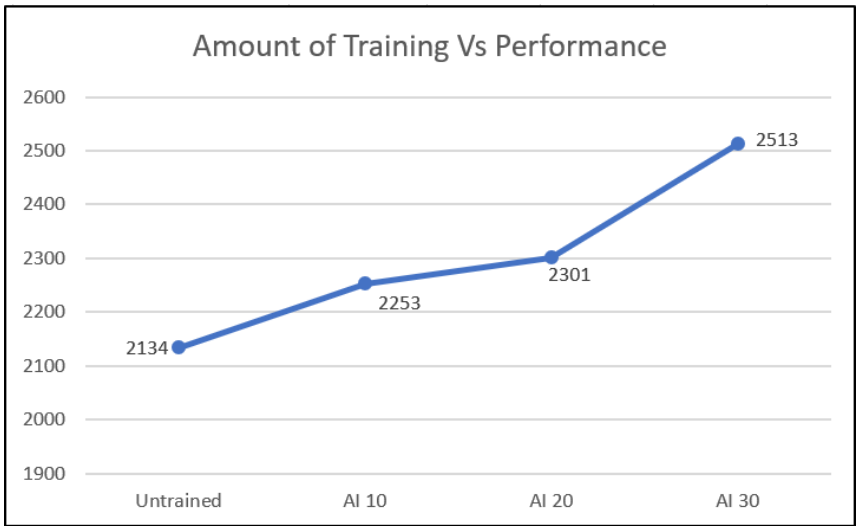

Figure 26: AIAgent Training Lengths vs Performance

Figure 26 is a line graph depicting the positive correlation that training length has to performance when training the AIAgent. This shows the amount of games increasing as the amount of training increases.

9.2 AI Steady

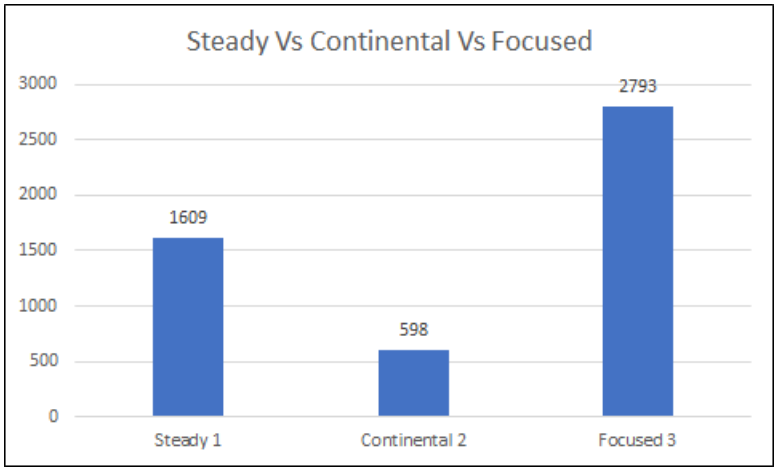

Figure 27: Steady Vs Continental Vs Focused : Wins out of 5000 games

Figure 27 is the base results when including the Continental Agent. Focused 3 has the biggest proportion of winners by a large amount showing it is the strongest pre-implemented strategy. Steady performed second best as expected with Continental coming third with only 598 wins.

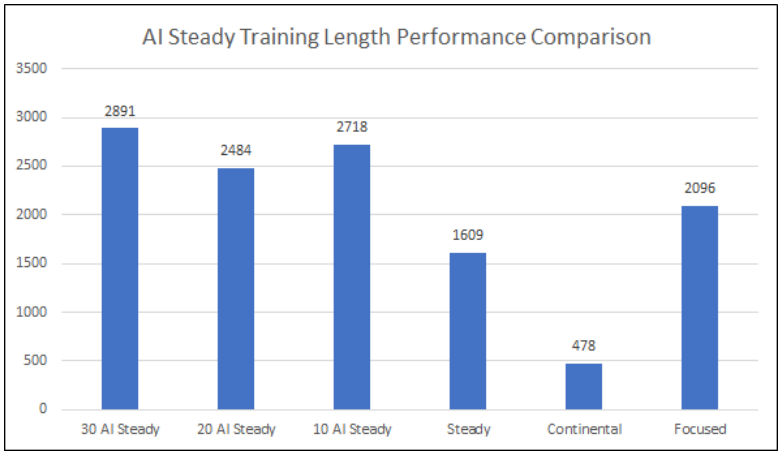

Figure 28: AI Steady Training Length Comparisons

Figure 28 Shows the comparison of the performance levels of the different lengths of training when the TD was applied to the Steady Agent. It shows a similar pattern to that of AIAgent; the difference being that the learning was a bit more unstable, this is suggested by the correlation being slightly less of a positive correlation.

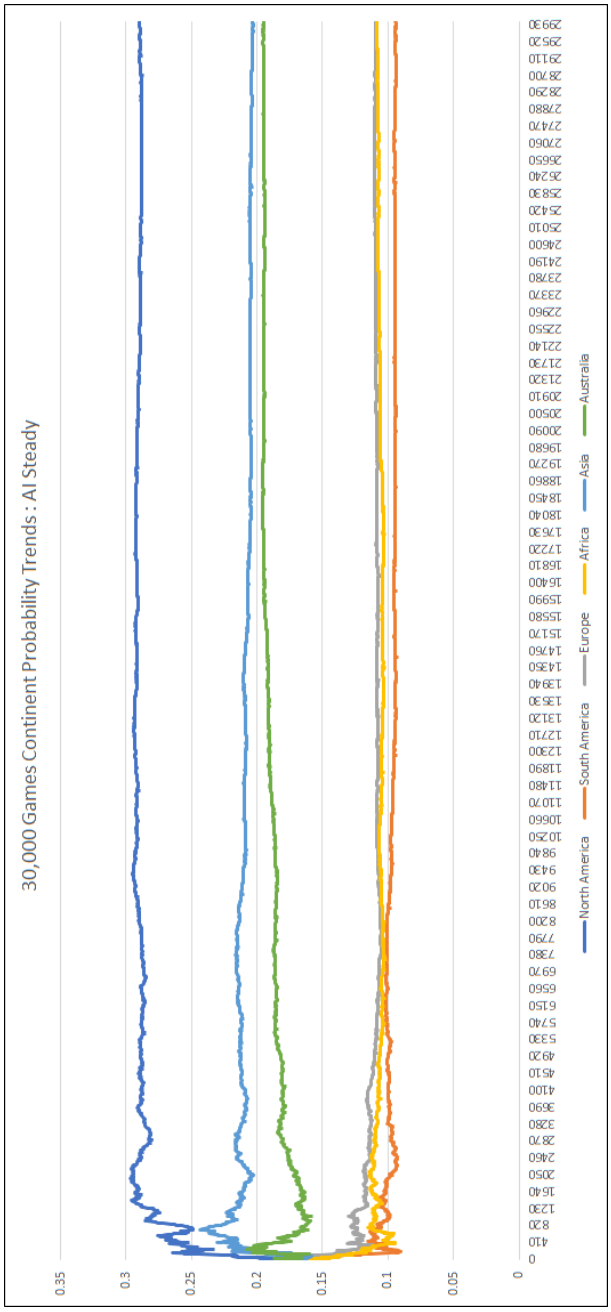

Figure 29: AI Steady Probability trends whilst training for 30,000 Games

Figure 29 is a line graph showing how the probabilities of the different continent choices change throughout the 30,000 games used to train the AI Steady Agent. As clearly visualised North America was the best continent to aim for.

9.3 AI Continental

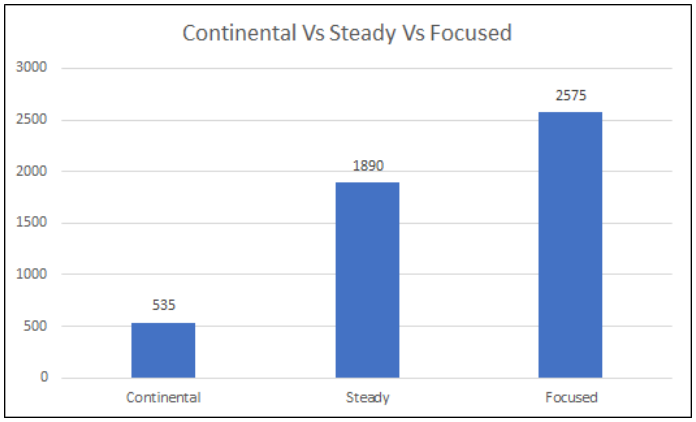

Figure 30: Continental Vs Steady Vs Focused : Wins out of 5000 games

Figure 30 shows the wins out of 5000 games for the strategies Continental, Steady and Focused. The Continental Agent performs very badly whilst there is no interaction from the AI and only wins 535 games. This was as expected.

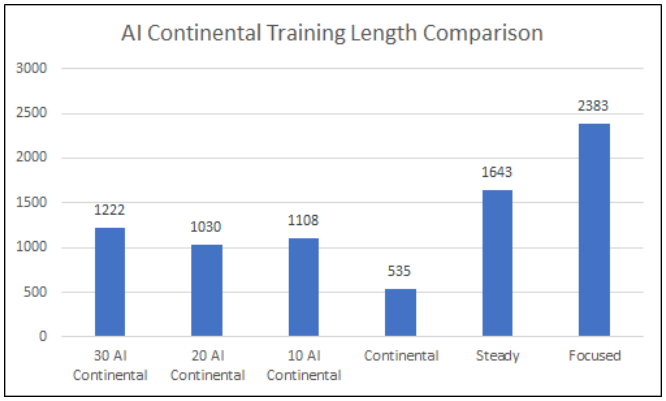

Figure 31: AI Continental Training Length Comparisons

The training length comparisons made in Figure 31 shows how the AI strategy improves the Continental Agent. However, it still performs worse than the other strategies. The correlation between training length and performance is not as strong as in the other agents.

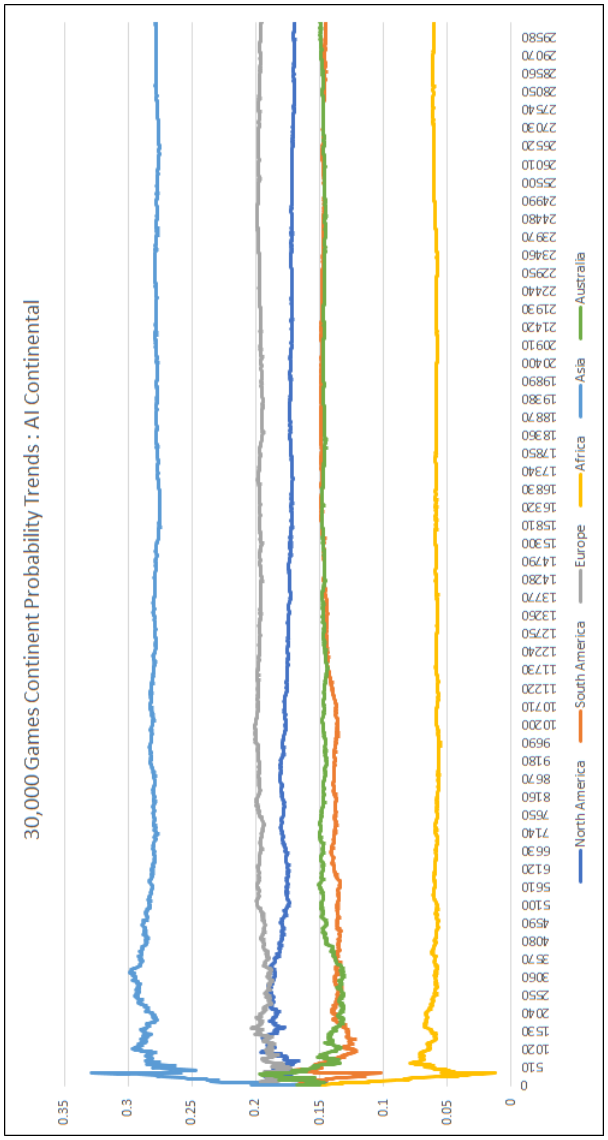

Figure 32: AI Continental Probability trends whilst training for 30,000 Games

Figure 32 shows that the continent Asia was the optimal choice and that Africa was the worst possible choice. The other 4 continents are relatively similar and level off with a probability of between [0.2-0.15]. This suggests that the Agent was indifferent in the choices between these and mainly wanted to avoid Africa and aim for Asia.

9.4 Discussion

The results are split into three main areas; AIAgent, AI Steady and AI Continent. AIAgent's results are based on the whole new Agent that was created for developing this TD learning algorithm. It was created using the Focused Agent as its base and went through the most thorough testing. The AIAgent was trained to 4 different levels; Untrained, 10,000, 20,000 and 30,000 games. Figure 18 was the independent variable for this testing as these are the amounts each of these three strategies initially won out of 5000 games without any interaction. Figure 19, 20 and 21 show that as the amount of training increases as does the performance of the AIAgent. This proves that the TD has made a difference and improves the existing Agent Focused. Further analysis occurred by studying the probabilities of selecting a continent during training (Figures 22, 23 and 24). This showed the favoured continent and also the behaviour of the trainer. It shows North/South America in all three cases losing the majority of the time and therefore being the least likely to be picked. On the other hand, these Figures show that Africa is in the top three in all three levels of training. This means that for the Focused Agent in particular Africa was a good choice and the Americas resulted in losing more often than not. Finally, Figures 25/26 show the correlation in two ways between the level of training and the performance with a more direct comparison. More specifically in Figure 26 is a clear positive correlation showing the AIAgent improves upon its existing counterpart, Focused. This confirms the research that the TD slowly updates and has a positive effect on the Agent. The two different confidence intervals that are used show the exploitation and exploration differences in Figure 22, 23 and 24. Roughly the first 2000 games are when the most exploration occurs in all three trainers. There are two reasons for this, Firstly, the initial confidence interval (which covers the first 125 wins) gives much more of a reward for a win. Secondly, the more the trainer has trained the more fitted each of the continents probabilities become, making it harder for them to deviate out of their local optima. This happens frequently with RL which is supported by research in section 2.3.

AI Steady, these results were primarily to show that when the AI was applied to the second best Agent it would improve its performance. Figure 23 shows the data used to measure the Agents before any alterations. Figure 24 shows a clear (rather large) improvement on the Steady Agent. This new AI Steady Agent improved by up to 1200 more wins than its predecessor. The probability trends in Figure 25 were from more of an interest perspective; it does give insight into how the AI Steady Agent learns and which continent it assumes best. The probability trends were only shown for the best performing level of training (30,000 Games). Even though the best continent was North America and this did conflict with the AIAgent's training; this was likely to be because of more game theory reasoning behind the strategies. It would be logical as North America is a strong continent

to hold due to it high troop bonus and low number of territories linking to other continents. Overall the AI strategy when applied to Steady did improve it. Dramatically so. The training on this Agent fits to its local optima at 4000 games. This shows that Steady took more time to train effectively than the AIAgent. Which explains why the training levels and AI Steady's performance level had less of a positive correlation than AIAgent.

AI Continental, the AI trained Continental Agent performs better than the original. Even though it does not beat the other agents it plays against the improvement is around a 200% improvement; the Agent goes from 535 wins to 1222 (Figure 27). The Agent performs best at 30,000 games which leads to Figure 28 showing that Asia is the optima Continent to aim for. This contradicts the other two strategies Focused and Steady and their AI training counterparts. This difference is almost certainly because of their difference in the way the Agent plays rather than the training technique. Once again the training only plateaus after 4000 games, this means it has fit to another local optima. This just like AI Steady, proves that AIAgent trains in a faster more progressive manner. Which is the reason for the lesser positive correlation in the AI Continental's performance levels.

These three agents and their corresponding AI opposites have provided sufficient evidence to conclude that there is an improvement on the strategies due to the TD method implemented. The results on their corresponding continent probabilities during training also provide data on which agents prefer which continents. The reason for this personal preference will come from the difference in tactics their underlying strategies will implement. There is a possibility that when the agents fit to their local optima's that is fitting is premature and there is another better optima to fit too they algorithm just can't escape this. This fitting also occurs at a different amount of training for each Agent. These differences in when the local optima are found show the differences in the effectiveness of the training for each Agent this effectiveness will again come down to the way they behave and the decisions they have been programmed to make.

10 Evaluation

An objective evaluation of the project will be provided in this section. It will compare the final results with that of the initial expectations. This will be done by testing if the requirements were met (functional and non-functional). Whether any of the ethical, social, legal and professional issues occurred; and a thorough analysis of the project management along with suggestions as to how to improve these topics. The author's evaluation will help to discuss whether a satisfactory level has been met with the project as a whole and anything that would be done differently.

10.1 Requirements Evaluation

There has been a close eye kept on the requirements throughout development in an attempt to keep the aim of the project on target. Examining these requirements near the end of the project allows a more objective way of seeing if the project was a success or a failure. This examination occurred formally through requirement testing. The original requirements were formalised into a table which had a Pass/ Fail/ N/a allocated to each. The amount of these which were Pass's will give a strong indication.

10.1.1 Functional

Functional requirements are the tasks that the system aims to be able to execute at the end of the project. These usually consist of data input/ output and objectives of the code. 6/9 of these functional requirements were successful with an unfortunate 3/9 failing. The reason why these 3 (6,7 and 9) failed was that the CTMS was not implemented in the end. It would have been really interesting to analyse the level of trust between agents at different stages of the game. Unfortunately, because of the step back in the TD learning to learn in between games, this was no longer applicable so could there was no reason in achieving (6,7). 9, on the other hand, failed because of the visual game working in an entirely different manner than the command line interface that the majority of the project had been working on; an oversight which occurred. However, the rest of the functional requirements the applicable and more important set were all passes (Figure 33). These were assumed to be true after the results section has explained them all working in the correct contexts.

Functional Requirements Testing		
#	Requirement	Result
1	The system **must** be able to play through a game without any	Pass
2	A TD learning algorithm **must** allow an agent to be taught how to perform these strategies.	Pass
3	Once training is complete the method it learnt **must** be able to be applied to any agent.	Pass
4	The trainer **must** output a list of all possible continents that the agent can aim for and the matching probabilities of winning based on which continent it chooses.	Pass
5	The trainer **must** improve on the exisiting agent it is applied to.	Pass
6	A working CTMS **must** be incorporated to allow the evaluation of trust levels between agents.	Fail
7	An output of agents and their trust levels **must** be output so analysis can take place.	Fail
8	Data **must** be output after every 5-10 games to show progress in the learning process.	Pass
9	The visual representations of the game **must** work with the AIAgent's strategey and perform without bugs.	Fail

Figure 33: Functional Requirements Tests

10.1.2 Non-Functional

The non-functional requirements have a similar level of importance to their functional counterpart. Fortunately, the non-functional requirements had a much higher lack of fails. All of the applicable requirements passed. The first two could be assumed to be true because of the nature of the results section and the code being open source (Figure 34). However, the third needed proof. This proof is shown in Figure 35. As there is a difference in the length of games of risk these numbers show a near-identical amount of time taken to carry out a run of the game, with and without the AIAgent. This is an important requirement because it is a way of analysing the complexity in a way which would affect the training rate hugely. Number 4 was not applicable due to the decision to teach the trainer in between games instead of during.

Non-Functional Requirements Testing		
#	Requirement	Result
1	Hardware **must** be good enough to host the Temporal Difference Learning/ all possible states that can be chosen.	Pass
2	The system **should** make certain security issues are not compromised when editing the source code.	Pass
3	The system **should** run in a near identical amount of time before/after the algorithm is applied.	Pass
4	The CTMS **should** be supported by the system running Risk.	N/A

Figure 34: Non-Functional Requirements Tests

Timings			
Turns	WatchCLI	Trainer	Turns
119	1898	1726	112
84	1642	1621	85
62	1531	1478	58
54	1393	1323	55
42	1103	1139	41

Figure 35: Timings (before and after AIAgent is implemented)

10.2 Ethical, Social, Legal and Professional Evaluation

There are 4 types of possible issues within a project of this scale that must be considered through the entire duration. Issues that required attention were split into these 4 types, ethical, social, legal and professional. As mentioned in section 3 when deliberating these issues the ethical issue was made certain it would not occur due to the nature of GitHub and project being open for those who wish to use it, just as the developer did. The one legal issue that may occur has been mitigated the entire way through this project with referencing where necessary and checking the licensing of the code used. The possible social issue of using this to exploit the game of Risk through improved performance will be near impossible because of the large amount of integration that would need to occur into a widely used version of Risk. The likelihood is the more widely used version would have security to stop this sort of integration anyway. The way around the professional issue is similar to that of the ethical but the reasoning behind it is different. The use of open source code in this project leads to the developer's professional values causing the project also to become open-source. All of these possible issues were therefore handled with appropriate measures and if this project was to be repeated nothing would be changed in the way these were handled.

10.3 Project Management Evaluation

The project management has been hugely successful at every step. The constant changing and updating of the Gannt chart is probably the most useful management tool. It has allowed flexible and realistic goals to achieve in order to keep organised. The weekly meetings with Matthew have been very successful, specific questions prepared for each meeting meant that progress never slowed. Better minutes could have been taken in these meetings, however, these evolved into note-taking which ended up being useful; just less formal. One issue which occurred was the supervisor missed a meeting. The negative effects of this were easily overcome by organising another meeting time the following day. There was a good reason for this occurring and therefore the professional relationship between supervisor and developer was not hindered. This strong level of communication and understanding is what led to this strong professional relationship and low negative effect of this small issue.

The documentation of testing has allowed for test tables to be created and a reliable platform of code to be used for the results to be collected from.

The use of third-party tools has meant that there has been constant online documentation with multiple backups. Fortunately, these were not needed as no crashes occurred. There this was as well successful. To conclude overall the project management was sufficient, with the project being completed with plenty of time to spare.

10.4 Author's Evaluation

This is a more subjective evaluation over the project and how it has been completed. The author's opinions on the different sections will provide insight into what worked well or did not. A lot of this section focuses on the project and how it contributes to the academic world. The structure of this will be based on the questions which will query the usefulness/successfulness of the project in several different ways.

What is the contribution of this project to the specific field of study?
There is a primary area that this project delves into. The practical use of TD. TD is a field within machine learning which is explored and is developing massively at this present time. There is a heavy focus on the research around other machine learning techniques and how to theoretically apply these to a specific MAS environment. The project was carried out in accordance with a full software development plan and has added to base software engineering practices. Data analytical techniques have been carried out on data to help show/ prove the con-

tribution in a more visual way. These techniques have also added to the machine learning statistical analysis that others may wish to understand. The last main contribution was applying an RL technique in java on an open-source platform so anybody can view/ replicate.

Is this project a success?
To tell if this project has been a success it is vital to look at every section. If all the sections seem to reflect a success; then the assumption will be safe. The management of this project has been carried out beyond satisfaction, no problems have occurred and everything has been done with time to spare. The majority of the requirements have been fulfilled all those which are still applicable to the system that is. Not only have the requirements been met, but the results prove that the agent does meet the main project aim which was to improve existing agents performance through the use of Reinforcement Learning. Because of these sections all being carried out correctly and helping to provide the means to achieve the main aim of the project; it is safe to assume this project was a success.

Are there any limitations that apply to the project? There are a few limitations. First, is the hardware available could not process the number of possible game states for every turn for every agent. This is why the decision against teaching the agent to play the game at a lower level was made. Secondly, a limitation was the unsolvable nature of risk having a social side which is almost more important than the gameplay itself. This means that even though the agents were improved if two teamed up on one the agent would always lose. This is more of a limitation of the game which will affect the performance of the agents. Thirdly and finally, the time limit meant that the application of TD method to teach agents to aim for a continent every 10 turns or so could not be attempted which had the following effect of there not being a CTMS to implement. If there was another month provided then this might have been possible.

11 Conclusion

In conclusion, this project has been executed with the underlying aim to help assist current Agent strategies to perform at the popular board game Risk, through the help of machine learning. After a significant amount of research, the decision to use TD was made. This was based on the aim to teach the agent which continents to aim for. A small step back was taken which kept this aim true but taught the agent in-between games at a higher level. This led to most of the requirements being achieved except for a few non-applicable in-game ones. Furthermore, there has been a great structure designed and enforced by a lot of well thought out project management techniques which reduced the likelihood of

issues occurring. All of the points above lead to the assumption that the project was a success. To run/view the source code and the raw data there is a .zip file which contains this information along with some extra figures.

11.1 Summary

The findings of this project show that the TD method implemented does improve the existing agents in the game environment, Risk. The better the agents perform to start with, leads to them having a better learning curve during training. This means the more they train the better they become. However this being said, it is not a certainty due to the nature of machine learning, in general, being unstable and the game Risk being very complicated. What has been proved, is the AI strategy works on the existing agents.

11.2 Future Work

Whilst completing this project there were a few areas which left open doors to different areas of research and implementation. The majority of these were not in the scope of the project. The more interesting of the few that were in the scope will be discussed here.

11.2.1 Future Research

Machine learning is split into three primary research topics. These topics are (1) task-oriented studies; which is the development and evaluation of learning methods to improve the performance in an environment. The engineering approach. (2) Cognitive Simulation; which is the investigation into a computer simulation of human learning. (3) Theoretical analysis; which explores the possible learning methods independent of a domain [46]. So to be more specific this section entails future theoretical analysis.

As the project implemented a form of reinforcement learning, it is logical to do further research into this area. RL is where the algorithm learns a policy of how to behave given an observation of the environment it is in [47]. Within RL there are two simple tasks; learning prediction and learning control. Furthermore, learning control can have function approximation applied to it; this devises a formula to judge the decision made by the process and to see if it could have done better. This is described as a loss function [48]. Research into the area of RL with function approximation would be very interesting and is likely to apply

to this project in one way or another.

The second area of future theoretical analysis is the eligibility trace of TD which evolves it into TD(λ) where the λ is this trace. It enables a theoretical forward view and backward view to look at whether its future or past predictions have done well or badly and updates the possible predictions accordingly. This helps bridge the divide between training data and events [49].

These two areas cover further developing the existing learning method by introducing either a loss function to give the agent a negative reward when it does something bad. Or, to have it look forward and backwards to predict its options to a higher level. Two interesting topics which would help build upon this project.

11.2.2 Future Implementation

Two further implementations could be developed. Firstly, having the agent learn every 10 turns or so on which continent to aim for. This would have needed a much large set of possible states but would have given a higher level of learning for the agents. This would have been much more time consuming than the current method, but fortunately, it would only add to the depth of the intelligence of the agent. Both of the learning methods could be incorporated to help aid one another cohesively. Secondly, once this learning method had been incorporate in-game rather than in-between games the CTMS would have been applicable and an agents level of trust would help determine which continent to aim for. This would also have benefited the agents level of understanding on a wider scale by incorporating a social aspect.

These two implementations are possible future developments and if linked correctly with the future research could lead to some very effective learning methods/implementations in this specific game environment.

References

[1] Zhao, Rui, et al. Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing 115 (2019)*: pp213-237. 2019.

[2] A. Casey. (2015). *adamncasey/MultiplayerRisk, GitHub*. [Online]. Available: https://github.com/adamncasey/MultiplayerRisk.

[3] K. S. Løland, Intelligent agents in computer games. Masters Dissertation. Computer Technology and Informatics (IDI). The Norwegian University of Science and Technology. Trondheim, Norway. 2008, Jun.

[4] Ferber, Jacques, and Gerhard Weiss. Multi-agent systems: an introduction to distributed artificial intelligence. Reading: Addison-Wesley. Vol 1. 1999.

[5] A. Taylor, E. Galvan-Lopez, et al. Transfer Learning in Multi-Agent Systems Through Parallel Transfer. in International Conference on Machine Learning (ICML), *Theoretically Grounded Transfer Learning Workshop*. Atlanta. Vol 1. 2013.

[6] Hasbro.*Risk, Hasbro. (1993)*. [Online]. Available: https://www.hasbro.com/common/instruct/risk.pdf. [Accessed: 03- Jul- 2019].

[7] G. Robinson. *The Strategy of Risk*. [Online]. Available: http://web.mit.edu/sp.268/www/risk.pdf. [Accessed: 04- Jul- 2019].

[8] I. Palacios-Huerta. Professionals Play Minimax, *The Review of Economic Studies*. Vol 70. 2. pp395-415. 2003, April.

[9] J. P. McInerney. Maximin Programming — An Approach to Farm Planning Under Uncertainty. *Journal of Agricultural Economics*. Vol 18(2). pp279-289. 2008, Nov.

[10] Sarvapali D. Ramchurn, Dong Huynh, and Nicholas R. Jennings. Trust in multi-agent systems. *The Knowledge Engineering Review*. Vol 19. no. 1. pp 1–25, 2004.

[11] Esteve del Acebo, Josep Lluis de la Rosa, A fuzzy system based approach to social modeling in multi-agent systems. *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*. 2002, July.

[12] Sandip Sen. A comprehensive approach to trust management. *AAMAS '13: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. pp 797-800. 2013, May.

[13] N. Case. *The Evolution of Trust*. (2017). Ncase.me. [Online]. Available: https://ncase.me/trust/. [Accessed: 01- Jul- 2019].

[14] J. Brownlee. (2016. March). *Supervised and Unsupervised Machine Learning Algorithms*, Machine Learning Mastery. [Online]. Available: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

[15] V. Mnih. (2013. Jan). "Playing Atari with Deep Reinforcement Learning — DeepMind". DeepMind. [Online]. Available: https://deepmind.com/research/publications/playing-atari-deep-reinforcement-learning/.

[16] Y. Li, Deep Reinforcement Learning: An Overview. arXiv:1701.07274. 6. Vol 1. 2017, Jan.

[17] Y. LeCun, Y. Bengio, G. Hinton. Deep Learning. *The multidisciplinary nature of machine intelligence*. Nature. 521. Vol 1. pp436-444. 2015, May.

[18] D. Silver. (2016, Jun). "Tutorial: Deep Reinforcement Learning. Presentation" [Online]. Available: http://hunch.net/ beygel/deep_rl_tutorial.pdf.

[19] P. Hernandez-Leal, B. Kartal, M. E. Taylor. Is multiagent deep reinforcement learning the answer or the question? A brief survey. University of Alberta. Edmonton. CCIS 3-232. Vol 3. pp 232. 2018, Oct.

[20] A. Irpan. (2018, February). *Deep Reinforcement Learning Doesn't Work Yet*, Sorta Insightful. [Online]. Available: https://www.alexirpan.com/2018/02/14/rl-hard.html.

[21] J. Fingas. (2017, Dec).*AI beats top 'Dota 2' players in one-on-one matches*, engadget. [Online]. Available: https://www.engadget.com/2017/08/12/ai-beats-top-dota-2-players/.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning. *Nature. 518*. pp529–533. 2015, Feb.

[23] V. Mnih, A. P. Badia, M. Mirza, et al. Asynchronous Methods for Deep Reinforcement Learning. *ICML 2016*. Vol 2. 22. 2016, Feb.

[24] C. Fernando, D. Banarse, C. Blundell, et al. PathNet: Evolution Channels Gradient Descent in Super Neural Networks. Google DeepMind, London, UK. Vol 1. 23. 2017, Jan.

[25] G. Boutsioukis, I. Partalas, I. Vlahavas. Transfer Learning in Multi-agent Reinforcement Learning Domains. *EWRL9 (2011)*. Aristotle University, Thessaloniki. (LNCS, Vol 7188). pp249-260. 2012.

[26] A. Taylor, E. Galvan-Lopez, et al. Transfer Learning in Multi-Agent Systems Through Parallel Transfer. *International Conference on Machine Learning (ICML), Theoretically Grounded Transfer Learning Workshop*. Atlanta. Vol 1. 2013.

[27] BSC. (2019) "BCS, The Chartered Institue For IT Trustee Board Regulations - Schedule 3 code of conduct for BSC Members". Bcs.org. [Online]. Available: https://www.bcs.org/upload/pdf/conduct.pdf. [Accessed: 14- Aug- 2019].

[28] S. Balaji. Waterfall Vs V-Model Vs Agile: A Comparative Study On SDLC. in *International Journal of Information Technology and Business Management*. vol 2. 1. pp26-29. 2012, Jun.

[29] synopsys. (2017). *Top 4 Software Development Methodologies — Synopsys*. Software Integrity Blog. [Online]. Available: https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/. [Accessed: 14- Aug- 2019].

[30] M. Lutol, A Learning AI for the game Risk using the TD(λ)-Algorithm. Undergraduate. University of Basel Department of Mathematics and Computer Science. 2019.

[31] Szlenk M. Formal semantics and reasoning about uml class diagram. *2006 International Conference on Dependability of Computer Systems*. pp 51-59. IEEE. 2006, May, 25.

[32] Booch, Grady, Jim Rumbaugh, and I. Jackobson. UML: Unified Modeling Language. 1997.

[33] Ali NH, Shukur Z, Idris S. A design of an assessment system for UML class diagram. *2007 International Conference on Computational Science and its Applications (ICCSA 2007)*. pp 539-546. IEEE. 2007, Aug, 26.

[34] Purchase HC, Colpoys L, McGill M, Carrington D, Britton C. UML class diagram syntax: an empirical study of comprehension. *Proceedings of the 2001 Asia-Pacific symposium on Information visualisation*. Australian Computer Society, Inc. Vol 9. pp 113-120. 2001 Dec 1.

[35] Visual-paradigm. (2019). *Ideal Modeling & Diagramming Tool for Agile Team Collaboration*. Visual-paradigm.com. [Online]. Available: https://www.visual-paradigm.com/. [Accessed: 05- Sep- 2019].

[36] Tesauro G. Temporal difference learning and TD-Gammon. *Communications of the ACM. 38*. Vol 3. pp 58-68. 1995, Mar, 2.

[37] Schraudolph NN, Dayan P, Sejnowski TJ. Temporal difference learning of position evaluation in the game of Go. *Advances in Neural Information Processing Systems 1994*. pp 817-824. 1994.

[38] Sutton RS. Learning to predict by the methods of temporal differences. *Machine learning*. Vol 3. Issue 1. pp 9-44. 1988, Aug, 1.

[39] J. Wedel. (2018). *Everything that's not tested will break*. [Online]. Available: https://dev.to/stealthmusic/everything-thats-not-tested-will-break-1adg. 5. [Accessed: 04- Sep- 2019].

[40] Software Testing Fundamentals. (2019). *Software Testing Levels - Software Testing Fundamentals*. [Online]. Available: http://softwaretestingfundamentals.com/software-testing-levels/. [Accessed: 06- Sep- 2019].

[41] Software Testing Fundamentals. (2019). *Integration Testing - Software Testing Fundamentals*. [Online]. Available: http://softwaretestingfundamentals.com/integration-testing/. [Accessed: 06- Sep- 2019].

[42] Latex. (2019). *LaTeX - A document preparation system*. Latex-project.org. [Online]. Available: https://www.latex-project.org/. [Accessed: 06- Sep- 2019].

[43] JetBrains. (2019) *Features - IntelliJ IDEA*. [Online]. Available: https://www.jetbrains.com/idea/features/. [Accessed: 06- Sep- 2019].

[44] Google. (2019) *Google Drive: Free Cloud Storage for Personal Use*. Google.com. [Online]. Available: https://www.google.com/drive/. [Accessed: 06- Sep- 2019].

[45] P. Turrini. Agent-based Systems: Reinforcement Learning. Warwick University, 2019.

[46] Carbonell JG, Michalski RS, Mitchell TM. An overview of machine learning. *Machine learning*. pp 3-23. 1983, Jan, 1.

[47] Ayodele TO. Types of machine learning algorithms. *New advances in machine learning*. IntechOpen. 2010, Feb, 1.

[48] Xin Xu, Lei Zuo, Zhenhua Huang. Reinforcement learning algorithms with function approximation: Recent advances and applications. *Information Sciences*. Vol 261. pp 1-31. 2014.

[49] Richard S. Sutton, Andrew G. Barto.Reinforcement Learning: An Introduction. The MIT Press. Second Edition (in progress). pp 167-181. 2014.

[50] Cdn.bcs.org. (2019). [Online]. Available: https://cdn.bcs.org/bcs-org-media/2211/bcs-code-of-conduct.pdf. [Accessed: 09- Sep- 2019].