ARTIFICIALLY INTELLIGENT PLAYERS FOR PVP IN MMORPGS

MMORPG'LERDE PVP İÇİN YAPAY AKILLI OYUNCULAR

TARIK KARŞI

Assist. Prof. Dr. ZÜMRA KAVAFOĞLU
Supervisor

Submitted to Institute of Informatics of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
in Computer Graphics

2020

ABSTRACT

ARTIFICIALLY INTELLIGENT PLAYERS FOR PVP IN MMORPGS

Tarık Karşı
Master of Science, Department of Computer Graphics
Supervisor: Asst. Prof. Dr. Zümra Kavafoğlu
October 2020, 76 pages

Massively Multiplayer Online Role Playing Games (MMORPG) are very similar to real life problems since they are complex and partially observable. These games, which have many features such as trade, character improvement, item crafting, questing and player versus player (PvP) battle, have hosted many studies. It is also very suitable for developing and testing artificial intelligence algorithms in terms of the difficulty of the domain, its similarity to real life and abundance of data. In PvPs, the players need to take hundreds of real time actions and develop complex strategies in a partially observable environment. Therefore building an AI for PvPs in MMORPGs constitutes a difficult problem.

With reinforcement learning, a general AI can be developed that can make PvP in MMORPGs. But, designing a reward function manually for reinforcement learning problems in partially observable environments is quite a daunting task. Most of the existing work in the literature aim to accomplish reward shaping by learning from demonstrations or with some kind of human assistance like preferences or rankings. However, for most of the real world problems, it's nearly impossible to attain such facilities, and even if they are available, they restrict the success of the agent to some human level. In this work, we propose a reward shaping method for PvP in MMORPGs without any demonstrations or human assistance, and compare it with several manually designed reward functions. We compared manual reward shaping method, which use predefined rewards, with the MCTS reward shaping method, which only exploits the prediction characteristic of a simply crafted Monte Carlo Tree Search planner. PvP results for a single character type showed that MCTS method can be used for reward shaping.

Keywords: Artificial Intelligence, AI, Reinforcement Learning, Massively Multiplayer Online Role Playing Game, MMORPG, PvP, Monte Carlo Tree Search, MCTS, Reward Shaping.

ÖZET

MMORPG'LERDE PVP İÇİN YAPAY AKILLI OYUNCULAR

Tarık Karşı
Yüksek Lisans, Bilgisayar Grafiği Bölümü
Tez Danışmanı: Dr. Öğr. Üyesi Zümra Kavafoğlu
Kasım 2020, 76 sayfa

Çok oyunculu çevrimiçi rol yapma oyunları (MMORPG) karmaşık ve kısmen gözlemlenebilir olmaları açısından gerçek hayattaki problemlere çok benzerlik gösterirler. Ticaret, karakter geliştirme, eşya zanaati, görev yapma ve oyuncuya karşı oyuncu savaşı (PvP) gibi pek çok özelliğe sahip olan bu oyunlar birçok çalışmaya ev sahipliği yapmıştır. Alanın zorluğu, gerçek hayata benzerliği ve veri bolluğu yönünden yapay zeka algoritmaları geliştirmek ve test etmek için de çok uygundur. PvP'lerde, oyuncuların kısmen gözlemlenebilir bir ortamda yüzlerce gerçek zamanlı eylem gerçekleştirmesi ve karmaşık stratejiler geliştirmesi gerekir. Bu nedenle MMORPG'lerde PvP'ler için yapay zeka oluşturmak zor bir problemdir.

Pekiştirmeli öğrenme yöntemi ile MMORPG'lerde PvP yapabilen genel bir yapay zeka geliştirilebilir. Ancak, kısmen gözlemlenebilir ortamlarda pekiştirmeli öğrenme problemleri için manuel olarak bir ödül fonksiyonu tasarlamak oldukça zor bir iştir. Literatürdeki mevcut çalışmaların çoğu, örnek verilerden öğrenme veya insan yönlendirmesiyle ödül biçimlendirmeyi başarmayı amaçlamaktadır. Bununla birlikte, gerçek dünyadaki sorunların çoğu için, bu tür imkanlara ulaşmak neredeyse imkansızdır ve mevcut olsalar bile, yapay zekanın başarısını insan seviyesiyle sınırlarlar. Bu çalışmada, herhangi bir gösteri veya insan yardımı olmaksızın PvP için ödül şekillendirme yöntemi öneriyor ve bunu birkaç manuel tasarlanmış ödül yöntemi karşılaştırıyoruz. Önceden tanımlanmış ödülleri kullanan manuel ödül şekillendirme yöntemini, yalnızca basit bir şekilde hazırlanmış Monte Carlo Ağaç Arama planlayıcısının tahmin özelliğini kullanan MCTS ödül şekillendirme yöntemiyle karşılaştırdık. Tek bir karakter türü için yapılan PvP sonuçları, MCTS yönteminin ödül şekillendirme için kullanılabileceğini gösterdi.

Anahtar kelimeler:Yapay Zeka, Pekiştirmeli Öğrenme, Çok Oyunculu Çevrimiçi Rol Yapma Oyunu, Oyuncuya karşı Oyuncu, Monte Carlo Ağaç Araması, Ödül Şekillendirme.

ACKNOWLEDGEMENT

I wish to express my sincere appreciation to my supervisor, Asst. Prof. Dr. Zümra Kavafoğlu, she convincingly guided and encouraged me to be professional and do the right thing even when the road got tough. Without her persistent help, the goal of this project would not have been realized. I wish to show my gratitude to Prof. Dr. Haşmet Gürçay for his continuous support of leading me to better academic opportunities.

Finally, I must express my very profound gratitude to my precious wife Elçin Karşı for providing me with unfailing support and continuous encouragement throughout my study and through the process of researching and writing this thesis. She gave me a priceless gift, our newborn son Aras Tarık, and I present this thesis to him. Also I would like to thank to my family Fatma Karşı, Satılmış Karşı and my sister Gülcan Mehmedikoğlu who made me come to these days. I would like to acknowledge to my lovely cat Papi, who always stands next to me while writing this thesis. This accomplishment would not have been possible without them. Thank you.

TABLE OF CONTENTS

Page

ABSTRACT i
ÖZET iii
ACKNOWLEDGEMENT v
TABLE OF CONTENTS vi
LIST OF FIGURES viii
LIST OF TABLES ix
ABBREVIATIONS x
1 INTRODUCTION 1
1.1 Motivation and Scope of the Work 1
1.2 Thesis Outline 2
2 BACKGROUND 3
2.1 Artificial Intelligence 3
2.2 Reinforcement Learning 4
2.2.1 Reward Design in RL 5
2.3 Supervised Learning 6
2.4 Artificial Neural Networks 7
2.5 Monte Carlo Tree Search 9
2.6 Artificial Intelligence in Games 11
2.6.1 AI for Games 11
2.6.2 Games for AI 11
2.7 Massively Multiplayer Online Role-Playing Games 12
2.7.1 Common MMORPG Features 13
2.7.2 Common MMORPG Strategies 14
2.7.3 Common MMORPG Character Actions 15
2.8 Artificial Intelligence in MMORPGs 16
2.8.1 AI for NPCs in MMORPGs 17
2.8.2 AI for PCs (Agents) in MMORPGs 17
2.8.3 AI for Cheat Detection in MMORPGs 18
3 ARTIFICIALLY INTELLIGENT PLAYERS FOR PVP IN MMORPGS . . . 19
3.1 Game Design 19

3.2 Reinforcement Learning Design . . . 22
3.2.1 Observation Design . . . 22
3.2.2 Manual Reward Design . . . 27
3.2.3 MCTS Reward Design . . . 29
3.3 Implementation Details . . . 32
3.3.1 Environment . . . 32
3.3.2 Unity . . . 33
3.3.3 ML-Agents . . . 33
3.3.4 Training Parameters . . . 37
3.3.5 Warrior Character . . . 38
3.3.6 MCTS . . . 40
3.4 Results . . . 40
4 CONCLUSION . . . 48
REFERENCES . . . 50
Appendices . . . 53
A Other MMORPG Information . . . 53
A.1 Other MMORPG Features . . . 53
A.2 Other MMORPG Strategies . . . 57
B PAPI Online Implementation Details . . . 59
B.1 Interfaces . . . 59
B.1.1 Player Properties . . . 59
B.1.2 Player Interface . . . 60
B.1.3 Skill Interface . . . 65
B.1.4 Attack Skill Interface . . . 67
B.1.5 Buff/Debuff Skill Interface . . . 67
B.2 Creating Warrior Character . . . 69
C Manual Reward Adjustment . . . 73
C.1 Manual Reward Test Scenarios . . . 73
C.1.1 Passive Player vs Passive Player . . . 73
C.1.2 Active Player vs Passive Player . . . 74
C.1.3 Active Player vs Active Player . . . 75
CURRICULUM VITAE . . . 76

LIST OF FIGURES

2.1 Reinforcement Learning 4

2.2 Sparse and Shaped Reward 6

2.3 Artificial Neural Network Architecture 8

2.4 Monte Carlo Tree Search 10

3.1 Ray Perception Sensor 3D 26

3.2 Reward Shaping with MCTS 31

3.3 Learning Environment 35

3.4 Behavior Parameters 36

3.5 Tensorboard Charts of Training with Manual Reward Design 42

3.6 Tensorboard Charts of Training with MCTS Reward Design 43

3.7 Manual (Red Agent) vs MCTS (Blue Agent) PvP Result 45

3.8 Manual (Blue Agent) vs MCTS (Red Agent) PvP Result 2 45

3.9 Manual (Blue Agent) vs MCTS (Red Agent) PvP Result 3 46

3.10 Manual (Blue Agent) vs MCTS (Red Agent) PvP Result 4 46

3.11 Manual (Blue Agent) vs MCTS (Red Agent) PvP Result 5 47

3.12 Test Results 47

LIST OF TABLES

3.1 Observations 22
3.2 Player Info Observations 23
3.3 Enemy Info Observations 24
3.4 Skill Info Observations 25
3.5 Attack Skill Info Observations 25
3.6 Buff/Debuff Skill Info Observations 26
3.7 Reward Categories 27
3.8 Action Free Rewards 29
3.9 MCTS Rewards 31
3.10 Hyperparameters 38
3.11 Warrior Properties 39
3.12 Warrior Attack Skills 39
3.13 Warrior Buff/Debuff Skills 39
3.14 Agent Behaviors 44

ABBREVIATIONS

MMORPG	Massively Multiplayer Online Role Playing Game
PC	Player Character
NPC	Non Player Character
PvP	Player versus Player
ML	Machine Learning
AI	Artificial Intelligence
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
SL	Supervised Learning
ANN	Artificial Neural Network
PPO	Proximal Policy Optimization
MCTS	Monte Carlo Tree Search
UCB	Upper Confidence Bound
UCT	Upper Confidence Bound 1 applied to Trees

1 INTRODUCTION

1.1 Motivation and Scope of the Work

Until now, mankind has found solutions to many complex problems by using their mind. Partially observable environments are among these problems. In time, they learn what they have to do in an environment which they cannot fully control. They act by making predictions with their experiences. Developing artificial intelligence techniques take this behavior as an example. Reinforcement learning method tries to reach the best solution with trial and error. It associates the actions with the observations from the environment and tries to maximize cumulative reward. Since the number of observations that can be obtained in a partially observable environment is limited, it is very difficult to establish this relationship. Therefore, the reward mechanism becomes extremely important. The agent, which has to fulfill many tasks in order to reach the target, must be guided by reward shaping. Reward functions are getting more and more complicated because the importance of actions and tasks change dynamically. It is very difficult to maintain and reuse such calculations.

In this thesis, we compare different reward shaping techniques that can be used for Reinforcement Learning methods developed for PvP in MMORPGs. First, we used a reward shaping method that gives the agent predefined rewards. In this method, which we call manual reward shaping, we gave different weighted positive rewards to the agent according to the importance of the achievements. We designed several manual reward shaping functions to reach the most successful result. Then, we have created a system that can feed reinforcement learning with rewards from Monte Carlo Tree Search, which is a heuristic search algorithm. When the agent does the best action in a situation, it should get the highest reward. So, how do we interpret that action as the best action? Since the same action may produce different results in different situations, a reward should be given according to the result. It is a daunting task to evaluate and prioritize the results. The MCTS algorithm finds the best action of a given state with its forward simulations. While doing this, it simulates the game many times and evaluates these results. So the result-oriented approach we just mentioned is also valid for MCTS. We created a system for evaluating actions which are taken by RL with MCTS. After each action, we ran MCTS to find the best action for the current state. Then we gave a positive reward to RL if the selected action is same as the MCTS's best action, otherwise, we gave a negative reward.

In order to compare these methods, we chose the player versus player (PvP) battle of MMORPGs, which constitutes a partially observable environment. We trained only warrior character agents with self-play deep reinforcement learning with different reward strategies on a sample MMORPG implementation we wrote. One agent is trained with the manual reward shaping strategy and the other is trained with the MCTS reward shaping strategy. Then, we had them battle each other and compared the results and agent behaviors.

With this work, we contributed the literature with a novel technique for reward shaping for PvP in MMORPGs. As far as we know, this is the only work in the artificial intelligence literature that includes MCTS for reward shaping. The algorithm we have implemented in our own testbed can be applied to PvPs in all MMORPGs. We made our testbed game and algorithm implementation open source [1] so that artificial intelligence developers can benefit. Finally we contributed to Unity Community by using ML-Agents in a different way.

1.2 Thesis Outline

The rest of the thesis consists of the Background chapter followed by the the main chapter that cover the details of the proposed method. Lastly, the Conclusion chapter is presented. A more detailed overview of the chapters are given below.

In Chapter 1, first a detailed information given about Artificial Intelligence, Reinforcement Learning, Supervised Learning and Artificial Neural Networks. Then, detailed information and related works in the literature are given about Monte Carlo Tree Search, Artificial Intelligence in games, Massively Multiplayer Online Role-Playing Games (MMORPG) and Artificial Intelligence in MMORPGs.

In Chapter 2, a detailed explanation about Artificially Intelligent Players for MMORPGs is presented. First details about sample MMORPG game design, Reinforcement Learning design are given. Then, implementation details of writing and applying of Reinforcement Learning and Monte Carlo Tree Search are given. Finally, training results are explained.

In the last chapter, we conclude the thesis by discussing the limitations and future directions of the presented research.

2 BACKGROUND

2.1 Artificial Intelligence

Artificial Intelligence (AI) is a concept that aims to generate computer systems that can perceive and learn their environment in order to perform specific missions like a human. It is inspired from human's natural intelligence that is able to solve problems that cannot be solved by brute force. Thus, AI algorithms are designed to solve the most demanding challenges which constitute complex software engineering problems. AI technology consists of learning, reasoning and self-correction processes.

Artificial Intelligence has drawn considerable attention in recent years because of its achievements in many areas. AI is created so that it can assist people, automate some activities or do some functions by itself. These functions can be listed as; speech recognition, decision making, perception, learning, problem solving and so on. Artificial intelligence can be classified as strong (artificial general intelligence) or weak (narrow AI). Weak AI is designed for a specific task such as Siri that seems very smart because she can talk with people and make explanations, however she actually works in a limited and predefined way. Google Search, IBM Watson and self driving cars can be other examples of Weak AI. On the other hand, Strong AI (artificial general intelligence) has human cognitive abilities and it can generate a solution without human intervention.

Nowadays computers can make billions of computations in one second, but this computational power is not enough for some situations. For example, in a chess game, there are more possible positions than atoms in the world. Although computers are powerful enough, it doesn't make sense to solve this problem by taking into account all the positions. Instead, solving the problem by learning from past experience and interpreting the current situation is preferred. There is a need for this human-like behavior in many areas like automotive, robotics, industry, education and games. The success of artificial intelligence is gaining people's trust in it day by day. Autonomous vehicles are preferred because they drive better than humans. Smart cities and houses make life easier, intelligent energy systems protect nature more. The benefits and possible risks of artificial intelligence will appear more clearly in the future.

Artificial intelligence methods also develop as their usage areas increase. The prominent one is Machine Learning (ML), which enables computers to learn by interpreting the data they obtain. It uses observations and data to find patterns and make decisions in the future. ML methods can be classified as supervised learning, unsupervised learning and reinforcement learning.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is the process of learning which actions to perform to get maximum reward in an environment. It tries to understand the connection between actions taken and environmental situations called states. Actions to be taken in any state create new states. Each state has a value that it can take in terms of its proximity to the solution of the problem. After each action taken, a feedback is given to the algorithm in order to interpret the states. The reason it is "reinforced" is these feedbacks, which are called the rewards. Rewards are calculated by taking into account the values of these situations and the factors related to the problem.

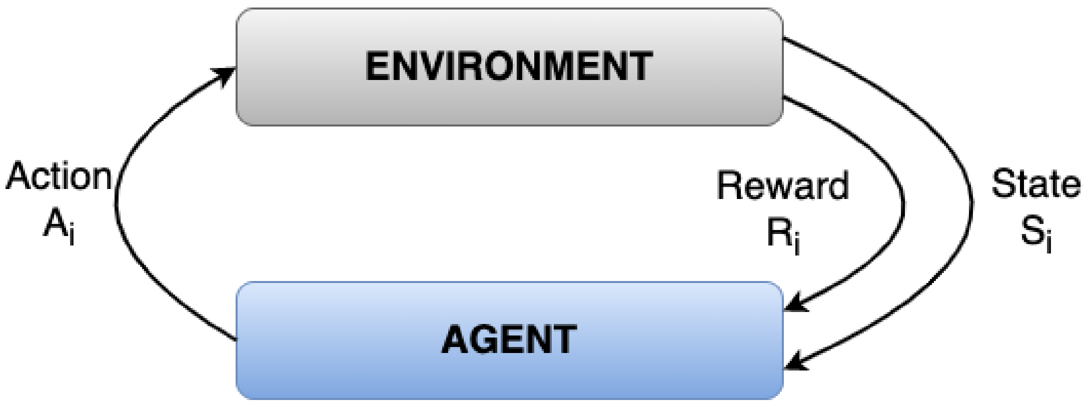


Figure 2.1: Reinforcement Learning

The sum of the rewards given from start to the end called cumulative reward. The RL tries to increase the cumulative reward with trial and error. In some cases, however, the action may also affect the value of subsequent states. In this case, the delayed reward is calculated by taking into account the future rewards. Another issue that needs to be taken into consideration is whether or not there is a bigger reward than the one received by taking an action against a situation. Therefore, RL should also give a chance to actions that are different from the previously learned actions. This is called exploration. Another strategy is to select the actions that were given higher rewards in previous cases and try to get higher rewards again. This is called exploitation. Exploration exploitation dilemma is one of the biggest challenges in RL.

The combination of Reinforcement Learning and Deep Learning principles is called **Deep Reinforcement Learning (DRL)**. The "Deep" implies multi-layered Artificial Neural Networks that resemble the human brain. DRL has been able to solve a wide range of complex decision-making tasks that were previously out of reach for a machine. This is because the DRL can learn from raw sensors or image signals as input. Deep learning requires large amounts of training data and significant computing power. Over the past few years, DRL usage has exploded, with an increase in data volume and a decrease cost of computing. Thus, DRL began to be used in many domains such as healthcare, robotics, games, finance and many more.

2.2.1 Reward Design in RL

One of the most important parts of Reinforcement learning is the reward mechanism. Rewards are sent to the ANN as a feedback of the actions it has taken, which tries to understand how well the action it made from these rewards. Its general purpose is to maximize cumulative reward. Rewards should be given to the results of the actions, not to the desired actions. For instance; while writing AI whose main purpose is to jump over the wall, if the jump action is rewarded, AI will try to increase its reward by constantly jumping and will not try to jump over the wall. So, it should be encouraged to take the desired action. These rewards are calculated with 'Reward Function' that evaluates the state and action. There are two types of reward function strategies: Sparse reward and shaped (dense) reward. In sparse reward strategy, a positive reward is given when agent reaches the desired result, and a negative reward when it fails. For instance; rewarding +1 when the agent wins the battle, and -1 when it loses. In this case, the algorithm spends a lot of time to learn the steps required to win the battle. This makes learning difficult. In the shaped reward strategy, the agent is encouraged by giving rewards to the intermediate steps that will enable it to reach the desired result. Thus, learning is easier.

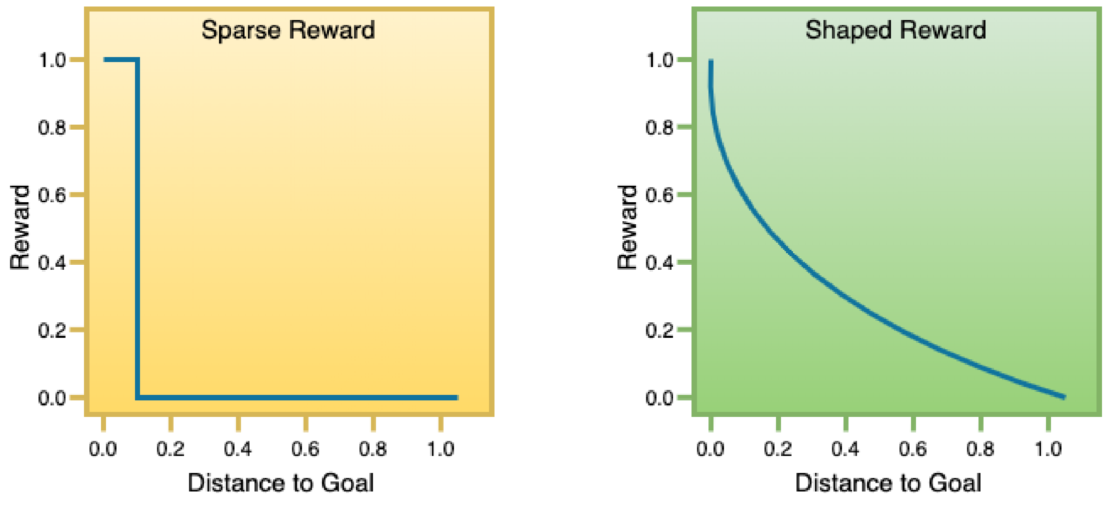


Figure 2.2: Sparse and Shaped Reward

2.3 Supervised Learning

Supervised learning takes a labelled dataset called tarining dataset as input, and aims to infer the relationship between each data and its label within this dataset. With a trained supervised learning model, labels of unlabelled data can be estimated. The larger and more diverse the training data, the more consistent the model's predictions will be.

Supervised learning is divided into classification and regression according to the type of the label. If the label takes discrete values, then the learning problem is a classification problem. If there are only two labels for each data point like 1 and 0, or true and false, then the problem is called binary classification. For example; We want to train a model trying to figure out if there are people in a picture. In the training dataset there will be pictures with and without people. Pictures which contain people are labeled with "yes" and the ones which do not contain people are labeled with "no". The case where the value of the label is more than two is called multi-class classification. As an example, the next step of the above example can be given which groups the people in the pictures by skin colors (brunette, blonde, auburn...)

If the labels take continuous values, the regression model is used. This model tries to find the value closest to the label value using the information provided. The lower the margin of error between the label value and the value found by the model, the higher the accuracy rate. For example; training a model that tries to calculate an estimated price of a car based on the given characteristics, is a regression problem. The training dataset will contain cars with different characteristics and their prices as labels.

2.4 Artificial Neural Networks

Artificial Neural Networks (ANN) are algorithms that are designed inspired by the human brain to recognize patterns. They cluster and label real life data such as audio, image etc. by processing them in raw format. They can divide the unlabeled data into groups according to their similarities, and classify the labelled data. They consist of layers and neurons, just like the human brain. In neural networks, neurons are the nodes in which the calculation process is done separately. These nodes are separated into three layers; input layer keeps input data, hidden layer makes mathematical calculations with entries and output layer returns output for the program. Each node takes in an 'activation function' and a 'weight'. The activation function of a node defines the output of that node given an input or set of inputs. The weights determine the significance of the input value. On the other hand, one of the objectives of the activation function is to standardize the output from the neuron. After a dataset passes through all the layers of the neural network, it returns a result from the output layer.

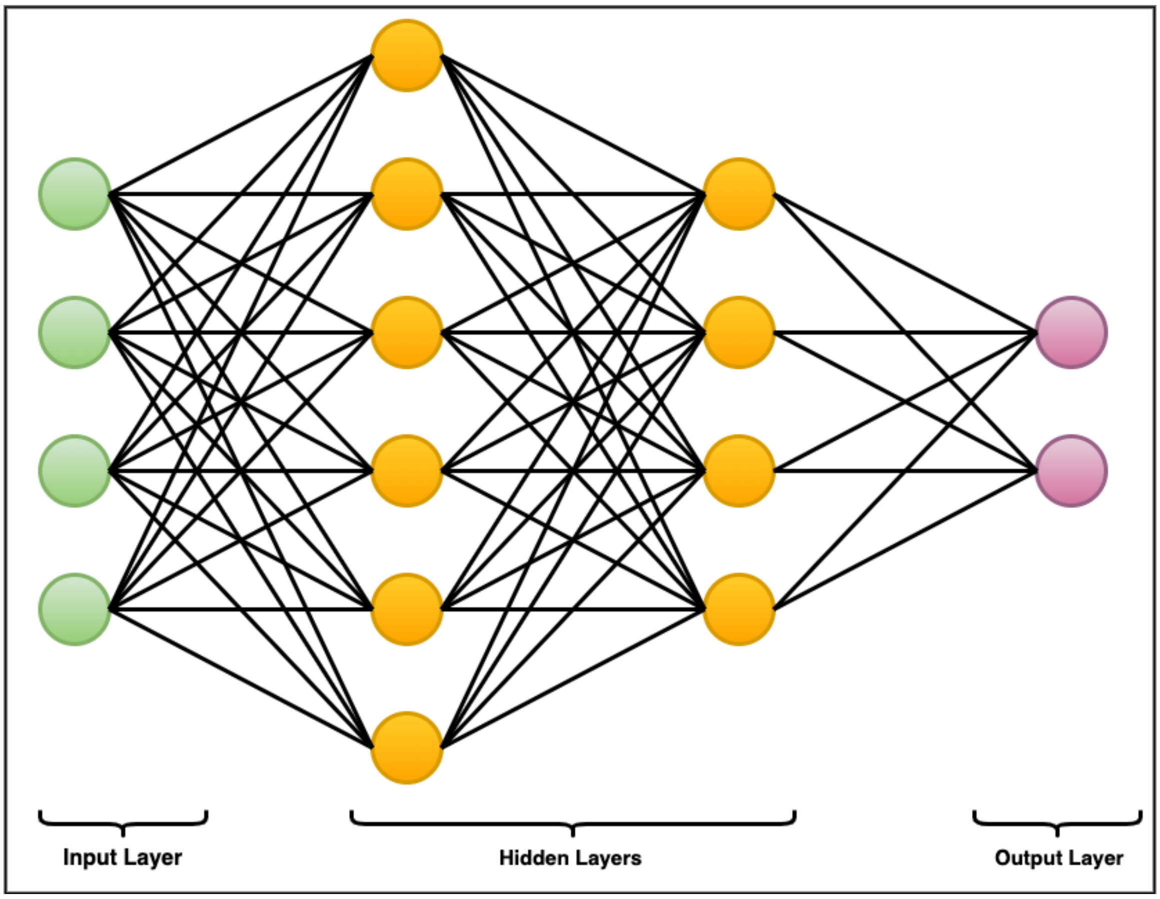


Figure 2.3: Artificial Neural Network Architecture

For instance, let's imagine an ANN for estimating the price of flight tickets according to these entries; departure airport, arrival airport, flight date and airline company. There are four neurons in the input layer that sends these entries to the hidden layer and finally, output layer produces price estimation of the flight. When estimating the price of a flight ticket, one of the most important factors is the departure date. Therefore, the departure date will have a large weight in neuron connections. Once artificial intelligence has been trained for flight ticket price estimation and the error rate is minimized, it can be used to predict future prices.

Large amount of labelled data and computational power are required to train ANNs. In order to train ANNs, the inputs in the dataset must be given and the outputs must be compared to the outputs in the dataset. The difference between the output of the ANN and the actual output is calculated by a specialized function called 'cost function'. It shows how wrong our model works, so when its value gets closer to zero, ANN's predictions get more consistent. The value of the cost function is intended to be as close as possible to zero so that the outputs of the ANN and the actual outputs in the dataset are close to each other. Several optimization techniques like Gradient Descent are used to find weights that will minimize the cost function.

2.5 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm which is introduced in 2006 by Rémi Coulom [2]. This algorithm, which is often used in games, uses randomness for problems that are difficult or impossible to solve with other methods. When the current state of the game is accepted as the root node, the possible actions form a huge tree consisting of billions of nodes. Therefore, it becomes impossible to evaluate this whole tree to decide which action to take in this current state of the game. The main focus of this algorithm is to find the most promising successive node. It tries to do this by going to the end with randomly selected actions in the tree called playout or roll-out. The game result reached at the end of each playout determines the weights of the nodes so that better nodes are more likely to be chosen in future playouts. Although it benefits from actions that have been identified as the best so far, it continues to search for alternative decisions and decides whether they can replace the current best strategy. In this way, unexplored parts of the tree can be discovered, and a more convenient way can be found. MCTS process consists of four phases which are selection, expansion, simulation and backpropagation.

Selection Phase In the selection phase, a search tree is traversed from the root node and followed down to the corresponding leaf nodes to select most promising leaf node. When the algorithm always chooses a node with a high score (exploitation), it misses the high-scored nodes that have not yet been selected. However, when it always tends to different nodes (exploration), it cannot determine how successful a node is in different situations. This is called exploitation and exploration dilemma. The following formula called UCT (Upper Confidence Bound 1 applied to trees) is used to balance exploitation and exploration.

$$S_i = x_i + c.\sqrt{\ln(t) \div n_i} \tag{2.1}$$

S_i = value of a node i

c = a constant

x_i = empirical mean of a node i

t = total number of simulations

n_i = number of simulations of a node i

Expansion Phase In the expansion phase, a new child node is created under the leaf node that was optimally found in the selection process. This child node is created by selecting a random action of this leaf node.

Simulation Phase In the simulation phase, the playout is done with random moves until terminal state is reached. Terminal state is the state where the game ends. (For example in chess, the game is won, lost, or drawn)

Backpropagation Phase In the backpropagation phase, the nodes that are visited to reach the terminal state, are updated with the result of the game. The number of wins is increased for each won game and the number of simulations is increased for each playout made. The ratio of the number of wins to the number of simulations shows the success of the node.

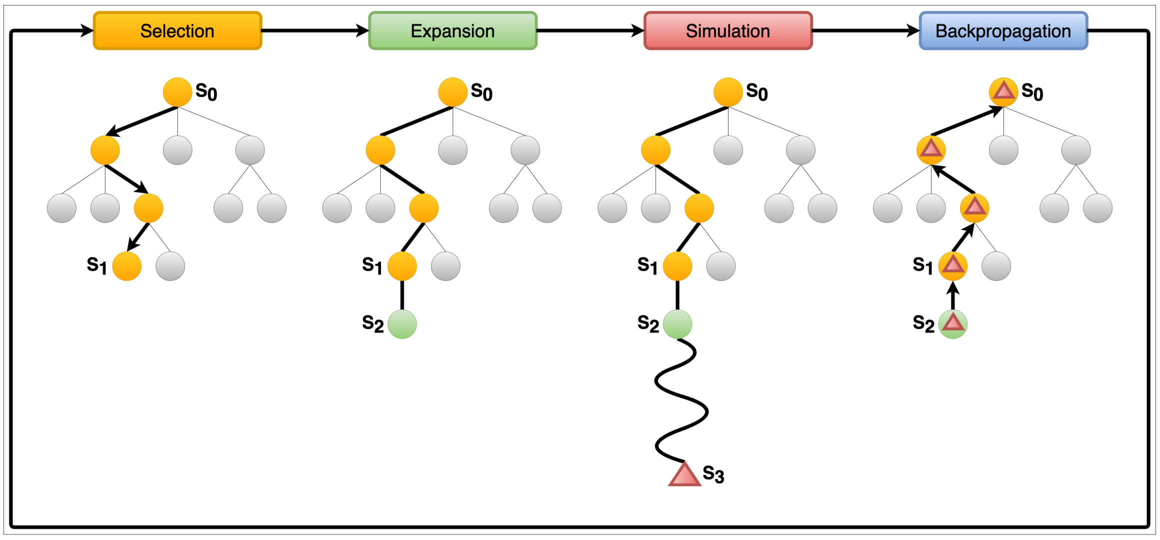


Figure 2.4: Monte Carlo Tree Search

MCTS offers significant advantages that minimize the search space. In addition, this algorithm can be interrupted at any time because it can give the most promising node it has found while it is running. This enables time-limited solutions. Although MCTS is suitable for making optimal decisions in Artificial Intelligence problems, it requires a great number of iterations and memory to determine the most efficient path.

2.6 Artificial Intelligence in Games

Games usually have situations to hang on, problems to solve, and goals to achieve. Players compete to achieve the desired goals. To be successful in a game, they use their abilities such as patience, intelligence, memory and speed. Competitions for some games are organized all over the world and awards are given to the top ranked players. This competitive environment inspires developers to develop AI algorithms that can contribute to these games or play them better than humans. Additionally, in some games there are scenarios very similar to real life. The test environment and data that cannot be obtained while trying to solve real life problems are available in these games. This allows the games to be used to develop and test artificial intelligence. In general, we can divide the relationship between AI and games into two groups as AI for Games and Games for AI as in [3].

2.6.1 AI for Games

Games have become more intelligent in certain aspects since they began to use AI. In video games, the human-like behavior of non-player characters (NPC) increases the user experience. For example, for a player who plays a car race alone, the robotic behavior of opponent racers is not enjoyable. Pathfinding and decision trees are mostly used for making NPCs intelligent and responsive. Using the patfinding and A* algorithms, Sazaki et al. developed NPCs that can find the shortest way by overcoming obstacles [4]. Some learning methods are also used to develop NPC agents. Wang et al. designed an reinforcement-learning-based NPC team which can accomplish tasks for playing Unreal Tournament (UT) Domination games [5]. Fang et al. applied reinforcement learning on a tank battle game to enhance tank NPC behavior [6]. These NPCs can be used to remedy player shortages in games. In board games, the learning techniques are used to develop strong opponents for players.

2.6.2 Games for AI

Games are very good choice to be the simulation and test environment of real life problems. Today, artificial intelligence algorithms are developed for smart cities, autonomous vehicles, medical and many other fields. It is very difficult to find test data or environment in such subjects. At this point, the games come to the rescue of AI developers. For example; An AI developed for autonomous driving can be easily tested in car racing or city simulation games.

The AI model can reach perfect level by playing millions of different game situations. Deepmind team trained an AI using MCTS and self play reinforcement learning which has defeated all AIs developed for Go and Chess so far [7]. Some moves of this AI were interpreted as not human-like and strange by the human experts, but it turned out that they are quite wise.

AI is also used to play games like a human. AIs which play games are often in a race with real players. The developed algorithms are tested with the best players on the world and when they beat the best, they're considered successful. OpenAI team made history by defeating the world champion team in Dota 2 game with the AI system they developed with self-play deep reinforcement learning [8]. OpenAI Five demonstrates that self-play reinforcement learning can achieve superhuman performance on a difficult task. Deepmind's multi-agent reinforcement learning based AI AlphaStar beat the best players in Starcraft II [9].

AI algorithms, just like humans, need to observe objects, actions and their interactions in the game. Players see the game objects, states and results of the actions taken. The data provided to these algorithms should be the same as human can see in terms of equality. It is unfair that AI uses information that human cannot obtain. This information can be gathered from game API or directly from screen. Some games have APIs that are accessible from outside programs. In such situations programmers can gather action results from these APIs and focus only on creating the AI algorithm. As opposed to this, many games don't give such an opportunity. The programmers cannot get any information about the game. So, the programs must understand the outputs like the human brain. To do this, outputs should be obtained by interpreting the game screen with image processing methods. First, programmers must find a way to convert screen information to game information. This can move them away from the main problem.

Games and AI became a beautiful duo that developed each other. In the future, games will move to a different dimension with the use of AI more effectively.

2.7 Massively Multiplayer Online Role-Playing Games

Massively Multiplayer Online Role-Playing Games (MMORPGs) are combinations of Role-Playing Games (RPGs) and Multiplayer Online Games (MOGs). RPGs have characters whose actions are controlled by players. In fantasy RPGs, these characters have some special abilities and items. On the other hand, in realistic ones, characters have equipment and

weapons. Players have to use these to advance in the game. MMORPGs differentiate from RPGs by hosting many players, with a persistent world which exist and develop independent of the activities of the individual players. These worlds include maps and dungeons. Most MMORPGs have different characters from different races. These games are like real life simulation. Players are competing with each other and players of different races. This competition includes experience, strength, economy, time and communication.
This section describes the common features of the MMORPGs for giving an insight about the problem domain of this thesis study, which is a simplification of the PvP battles of MMORPGs.

2.7.1 Common MMORPG Features

Characters First step of a MMORPG is character selection. Characters are grouped by their types and each type has specific properties and combat abilities (skills). Also, physical properties like agility, speed, defense etc. differ between character types. For example; Magician use magic powers and attacks from long range. Warrior uses weapons and physical power to make close range high damage attacks. Shaman uses buff skills to help himself and allies, and makes spell attacks from long range. Archer uses bows to make long distance attacks. Ninja makes super-fast close-range blade attacks.

Choosing right character is very important, because each type has a unique gameplay style. Games provide information about character types and their skills in character selection page or in their website. Players select their character according to their feeling or from this information and try to improve it in the game world. In MMORPG games, players compete with each other to make their characters stronger than others. To do this, they try to collect the most powerful items and gain experience to level up their characters. Selecting characters which do not fit players expectancy may cause waste of time. But some MMORPGs have the feature that enables players to change their character type without losing their current levels.

Skills Skills are very important in MMORPGs. Every Player Character (PC) has its own skills according to its type. Players use them to kill monsters or win a battle, which makes them quite crucial. In some MMORPGs skills have their own level system. Players empower themselves by improving their skills. Skill levels are upgraded by gaining experience or consuming special items. The skills can be grouped under two categories:

- **Buff/Debuff Skills:** Buff skills increase player's properties like health, mana, strength, power, speed and can be used for ally players. In contrast to buff skills, debuff skills decrease enemies' properties. Both buff and debuff skills can be temporary or permanent. Generally variable properties like mana, health is changing permanently and constant properties like strength, speed is changing temporarily.

- **Attack Skills:** Attack skills are used for attacking enemies. Attack skills have damage and range properties. Damage property denotes how much damage will be applied to the enemy when used. Range property denotes the maximum distance this skill can be used. Some attack skills have different effects on enemies for a certain period of time. This is called the debuff effect. For example:

  - **Dodge, Freeze:** Enemy cannot move and cannot use skills when it is applied.
  - **Burn, Poison:** This decreases enemy's health for a while.
  - **Slow:** This decreases enemy's speed.

All skills have timeout property which indicates the elapsed time required to use this skill again. Using any skill consumes mana. In order to use a skill, the player should have more mana than this skill needs. According to its type, skills can be applied on a single player or multiple players in an area.

All character types have different kinds of skills and these skills have perfectly balanced in MMORPGs. Every character type has a special role and importance, which emerges in group battles and dungeons. Mastering a character's abilities is also very effective in player versus player (PvP) battles and killing monsters. Using the right skill at the right time is very important for a player to overcome strength wars. Players create their own strategies for using skills.
Other features of the MMORPGs are given in Appendix A.1 as well.

2.7.2 Common MMORPG Strategies

PvP In MMORPGs, players must always fight. The goal of the game is not just to level up. The main goal of the players is to create very strong characters since power is very important to establish dominance in the game. There is the possibility of encountering enemy players in any part of the game. War is inevitable when it comes to taking over an area.

In some MMORPGs, players are divided into regions according to their choice at the beginning of the game. There can be more than two regions and players in different regions live in different places called cities. According to the story of the game, there are conflicts between these regions, and they are enemies. Players from different regions fight each other on public maps. The objective in the game is to dominate the world by preventing the development of enemies. In some MMORPGs, the situation is slightly different. All players live in a common city and fight each other in unsafe areas.

Players call their friends when they are attacked by enemies. Players with different character types from both sides are involved in the fight. A small fight becomes a pitched battle and continues until one side gives up. These wars are also a demonstration of power by the groups. In addition to such informal battles, some battles are made in places called arena designated by the game. Leader players gather other powerful players into groups called clans. The clans can challenge each other and enter the arena with the other side's acceptance. Here, after players of the same clan have determined their strategy, the battle begins. If one party kills the other's players or reaches a certain score, the battle is over. These battles are recorded by the game and the clans are sorted. Players try to improve themselves to join more powerful clans.

Players cannot accurately measure their power in collective battles. Players can offer a duel to a player they see as opponents. This is called Player versus Player (PvP). PvP can be made in private or public areas. The battle begins when the other player accepts the duel. Both players try to use their skills in the most effective way to win. In order to win a duel, it is very important to analyze the strength and abilities of the rival and to use his abilities at the right place and time. Effective use of debuff capabilities in particular brings victory. This fight continues until one of the players dies.
Other strategies of the MMORPGs are given in Appendix A.2 as well.

2.7.3 Common MMORPG Character Actions

Moving Every character moves in the game world. It has the ability to move right, left, up and down. The ability to jump is omitted in this work because not all MMORPGs have such a feature. With this ability, players can move their characters in a specific coordinate or direction.

Attacking Attacking is a character's ability to fight with a gun or bare hand without the use of skills. Depending on the characteristics of the weapon in hand, the character may need to approach the enemy. For example, a character with a pistol can shoot at the maximum distance allowed by this pistol, while a character with a sword must reach the distance that can come in contact with the enemy. This feature is included in all characters and is used by players to deal more damage when skills are unavailable.

Using Skills In all MMORPGs, the characters have unique skills. Players try to use these skills effectively. A character can have between five and ten skills. Players line up these skills by frequency of use and use them with shortcuts.

Using Heal Health is the most important property for players. When the amount of health decreases to zero, their characters die, and they are defeated. Therefore, they use health potions to increase the amount of health. The amount of health increases according to the type of the potion used. Players with a full health level cannot use these potions.

Using Mana Mana is the second most important feature after health. Players use their skills with mana. Players with insufficient mana levels cannot use their skills and have a hard time facing their opponents. Therefore, they use mana potions to increase the amount of mana. The amount of mana increases according to the type of the potion used. Players with a full mana level cannot use these potions.

2.8 Artificial Intelligence in MMORPGs

MMORPGs are becoming widespread and played by a lot of people day by day. There are many studies to increase the attractiveness of these games and solve their problems. By using AI techniques MMORPGs constitute a proper testbed for AI research since these games are similar to the real world in many aspects such as competition, war, economy, trade, and sociability. Moreover, the interaction of a large number of PCs, NPCs and other objects creates a wide dataset, which provide the data needed by AI developers. There are many studies in the literature focusing on these games but, in this section, we will give an overview of the ones which are most relevant to the content of this thesis. In general, we divided these studies into three groups as non-player characters, player characters and cheats.

2.8.1 AI for NPCs in MMORPGs

Non-player characters are managed by the game itself. Before the recent advances in artificial intelligence, these characters were simply controlled by finite state machines or rule-based algorithms. In this case, the behaviors of NPCs were very limited in the game and did not seem natural. With the improvement of artificial intelligence algorithms, several studies were conducted for achieving smarter NPC behavior in MMORPGs, with empty decision trees and learning algorithms.

In order to realize decision tree, behavior tree and rule-based system methods, it is necessary to define the action, rule and state elements of the character and to determine the relationship between these elements. Ballinger et al. modeled the behavior of non-player characters with a rule-based system [10]. Their learning method allows the character to improve its behavior over time. Agents explore their environment and learn new behaviors in response to different experiences. In her work, Merrick designed non-player characters using a motivated reinforcement learning method [11] [12]. Lee et al. developed a framework of intelligent agents to realize NPCs [13]. They worked on Finding, Finite State Machine, Fuzzy State Machine, Script, Flocking, Decision Tree, A-life, Neural Network and Genetic Algorithm techniques to create this framework. In MMORPGs, players battle NPCs on the battlefield. NPCs' patrol-like movements are very easy for players to understand and overcome. Maggiorini et al. have developed smarter and more natural NPCs using behavior tree and blackboard techniques [14]. Some MMORPGs have NPCs that help player characters. These help the players by fighting with them, healing or buffing. Rhujittawiwat et al. used genetic algorithms to create learnable buddies which assist real players [15].

2.8.2 AI for PCs (Agents) in MMORPGs

In MMORPGs characters are usually controlled by real players. But it is also needed to control them with AI for test purposes or to maintain the visibility of the games when players are inactive.

Behavior trees are also used to implement PCs. Very complex behavior trees are created to imitate human behavior. But often it can be easily understood by players that they are not human players because of the repetition or unnaturalness in their behaviors. Tomai et al. used Naïve Bayes and Inverse Transform Sampling methods to develop a learning behavior tree [16]. In MMORPGs players try to perform tasks given by NPCs. In order to develop an AI capable of performing these tasks, the artificial player needs to know the game world, understand the tasks it takes, and plan how to do it. Maliković et al. designed a prolog based automated planning system and a belief-desire-intention (BDI) agent model [17]. In MMORPGs, players use their financial skills as well as their combat skills. Players strengthen their characters by trading items. In their study, J. Reeder et al. developed trading agents by using historical trading data from Eve Online with a reinforcement learning method [18].

2.8.3 AI for Cheat Detection in MMORPGs

The world of MMORPGs consists of thousands of players competing with each other. Everybody is racing and they spend time developing their own characters. In this competitive environment, players spend their hours on gaming. The more time they devote to the game, the greater their chances of progress. It is very difficult and tiring to rise to the top. For this reason, some players try to push forward in illegal ways. They sell the items they earn in the game for real money. Players who buy these items for real money without wasting any time in the game are unfair to others. A. Fujita et al. used a machine learning method on trading network to find real money traders [19]. Along with these, there are softwares called bot developed by malicious people. The bots continue to play instead of the players. So, players can make progress in the game without spending time on the computer. In their work, Kesteren et al. and Thawonmas et al. tried to find the differences between bots and human behavior [20] [21].

3 ARTIFICIALLY INTELLIGENT PLAYERS FOR PVP IN MMORPGS

3.1 Game Design

As described in Section 2.7, MMORPGs consist of many feature and strategies. In this thesis study we focus on PvP, therefore we implemented a simple MMORPG called PAPI Online which includes only player characters. In our game, player characters can battle with other characters by using their skills. Our implementation doesn't include real graphics for the sake of simplicity but it is modular enough to include graphics or other MMORPG features like NPCs, if needed.

Every player created in our game has the following properties:

- Position information refers to the player's location on the game map.
- Health information represents survival and the player with a health value of zero dies. The health value of the characters may increase depending on the level and the items used. The greater the health of a character, the more likely he is to survive and win the PvP.
- Mana information shows the amount of mana the player has. Player characters need mana in order to use skills. If the mana value is below the amount needed by the skill, the player cannot use the skill and will most likely lose the PvP. The mana value of the characters may increase depending on the level and the items used.
- Potions are used by the players to regain reduced health and mana. Players can regain reduced health and mana by using these potions. They are in constant battle which reduces health and mana and they can die when they don't have enough potions.
- The speed information determines how fast the player can move. Characters with high speed can easily run away from the ones with low speed or catch them.
- The damage information shows how much damage will be done when the player attacks. The damage value of the characters which makes them stronger than their opponents can be increased according to the items used and the level.

- The defense information shows how much damage will be absorbed. damage and defense values are very important for winning a PvP.
- The attack distance information determines how far the player can attack. Characters can attack from a long or close distance according to their type. The smaller the value, the closer the character must be while attacking.
- Stunned and dead information is used to get the current status of the player. If a player is stunned, he cannot move and use skills for a while. And when a player dies, it loses the PvP.
- Skills that the player can use. Skills are the most important part of a PvP, and they vary according to each player character type. Players must use the skills strategically to win the PvP.

Every player created in our game has the following actions:

- Moving the desired direction. Players actively move during PvP.
- Using skills. The player can use whatever skill he has. However, this usage is possible as long as he has sufficient mana and distance.
- Making attacks. Players try to increase the damage they applied on the opponent by performing normal attacks alongside the skills.
- Using health and mana potions. When players' health and mana decrease, they try to complement it with the potions they have.

In order to implement the skill mechanism, we first categorized the skills into three groups: attack skill, buff skill and debuff skill. Every skill created in our game has the following common properties:

- Name and kind information that distinguishes the skill.
- Mana consumption information indicates that how much mana needed to use a buff/debuff skill.
- After a skill is used, it becomes inactive for a certain period of time and cannot be used. The timeout information shows how long the skill remains inactive when used.

- Availability information shows the skills is ready to use.

Attack skills deal damage to the enemy. Each attack skill created in our game differs from other skills with the following properties:

- Power information indicates how much damage applied to the enemy.
- In order to apply attack skills, the player must be in close enough to the enemy. Range information shows the distance is needed to apply this skill.
- Attack skills can have a debuff effect and, when applied, can cause a debuff effect as well as damage to the enemy. Debuff and percentage information shows the presence of debuff effect of the attack skill and the possibility of its realization.

Buff/Debuff skills affect the properties of players. Buff skills make a positive contribution to the character. So, players can only use buff skills to themselves and their allies. In contrast, debuff skills make negative effect to the character. So, players can only use debuff skills to their enemies. Each buff/debuff skill created in our game differs from other skills with the following properties:

- Kind information determines which one of the health, mana, speed, damage and stun properties will be affected by the buff/debuff skill.
- Quantity of the contribution is obtained by the amount information.
- The affect of buff/debuff skills can be permanent, temporary or periodic. For example, even if a debuff skill that causes permanent damage loses its effect, the health that it reduced will not return. In contrast, when a buf skill that increases the speed of the character loses its effect, the speed of the character will be restored.
- Duration information shows how long the buff/debuff skill will be effective.
- Periodic information indicates the buff/debuff skill affects the applied character multiple times during duration.

With this infrastructure described above, several player character with desired features and skills can be created. We tested our AI methods, with a specialized player character whose properties are given in the Section 3.3.

3.2 Reinforcement Learning Design

3.2.1 Observation Design

Players must observe lots of information at the same time to be able to play games like MMORPG. Besides their own player information, they should observe the game world and the movements of their opponents. They decide within a very short time and take action by interpreting these instantly changing information. When making a decision, they keep an eye on the past and the future. In PvP the first and simplest task of an agent is to get close enough to the opponent for attacking. To do this, they need the position information of themselves and their opponent and the distance between them. They come closer or move away, depending on the situation they are in and the position of their opponent. Then, they find the direction to move and perform the action towards that direction. Many situation evaluations and calculations are made even for the simplest actions. In order for the AI agent to play the game like a human, all the observations made by the real players must be shared with it. In this way, it learns how the game environment is affected as a result of the action, and the relationship between actions and game objects. Besides the information not observable by human players should also not be given to the AI agent. Table 3.1 lists the collected information as observations.

Observation	**Description**
Distance	Distance between player and enemy.
Player Info	Properties of player. [Table 3.2]
Enemy Info	Observable properties of enemy. [Table 3.3]
Skill Info	Properties of player skills. [Table 3.4]
Raycast Info	Unity raycast information. [Figure 3.1]

Table 3.1: Observations

Player Info Observations In order to control the character effectively, it is necessary to know its characteristics well and follow them constantly. We call the observations covering these characteristics as player info observations which are listed in Table 3.2. Health of the character is the most important factor to win a war, because players lose the battle when their character's health value drops to zero. For this reason, it is a property that must be followed continuously. As the character's health decreases, the player can reinforce it using a health potion. However, as the health potion number is limited, it should be used carefully. Mana enables players to use their skills. Insufficient mana amount causes players to be unable to use their skills and lose the battle. With the mana potion, the amount of mana can be increased, but like the health potion, mana potion count is also limited. Speed, defense, and damage are properties that make one character superior to another. Players make their character stronger by using buff skills that increase these properties. At the same time, due to the debuff effects applied by the enemies, these properties may decrease. Keeping these properties at high values gives players an advantage in winning the war. During a PvP, melee attacks are also performed along with the skills to deal extra damage to the enemies. The attack range feature is the minimum distance required to make a melee attack on an opponent. In order to win a PvP, players constantly attack (both skill and melee attack) each other. They cannot take action only when they are under the effect of stun debuff and while another attack animation is already in progress. Like the real players, all the properties and states we mentioned above are given as observations to the RL system so that the AI algorithm will learn how to interpret these values.

Observation	**Description**
Player Position	Position of the player.
Health	Player's health information.
Health Potion	Health potion count.
Mana	Player's mana information.
Mana Potion	Mana potion count.
Speed	Speed of the player.
Defense	Defense of the player.
Damage	Damage of the player.
Attack Range	Attack range of the player.
Is Attacking	Indicator for player is attacking or not.
Is Stunned	Information about player is stunned or not.

Table 3.2: Player Info Observations

Enemy Info Observations Players need to observe their opponents constantly during PvP. They decide their own actions based on where enemies are and what they do. However only a limited number of features of the enemy is seen by the player during the game. Therefore, MMORPGs constitute partially observable environments. For example, while players can see how much damage they have inflicted on their opponents, they cannot see their remaining health. We call the observations covering these limited information about the enemy as enemy info observations (see Table 3.3). The first of these information is the position of the enemy. Apart from position information, debuff information can also be observed. Since visual animations for debuff effects are shown in MMORPGs, debuffs applied to competitors can be seen by the player.

Observation	**Description**
Enemy Position	Position of the enemy.
Debuff Info	Debuff kind applied to the enemy.

Table 3.3: Enemy Info Observations

Skill Info Observations To make an effective PvP, players learn using their skills at the right time and place. They try to turn the battle into their own favor with the strategies they set up. To do this, they master the features of the skills. There are 3 types of skills: Attack, buff and debuff, and can be distinguished by their kind value. Players know the skill types they have and decide which skill they will use according to the current situation of the game. Skills require mana to be used, whose amount determined by the mana consumption value. Players calculate the skills they can use according to their remaining mana and the mana consumption value. When a skill is used, it is deactivated for a certain time, called the timeout value, and cannot be used until that time has passed. Players follow the timeout value and meanwhile prepare themselves to re-use the skill. In order for the AI agent to learn how to use the skills effectively and produce strategies, we shared the general skill information we mentioned above and skill kind specific information as observations with the RL system. We call these observations skill info observations (see Table 3.4). The skills that the player characters have vary according to the character type and the game. In order for AI to adapt to all these changes, it must be trained with different skill combinations.

Observation	**Description**
Available	Skills availability information.
Kind	Skill kind information.
Mana Consumption	Mana consumption value of the skill.
Timeout	Timeout information of the skill.
Attack Skill Info	Properties of all attack skills. [Table 3.5]
Buff Skill Info	Properties of all buff/debuff skills. [Table 3.6]

Table 3.4: Skill Info Observations

Attack Skill Info Observations Players deal serious damage to their opponents using their attack skills. The use of attack skills is very important as it will be the winner of PvP, who first consumes his opponent's health. Each attack skill has different damage effects on the enemy and an application range. Players tend to use more damaging skills, and they adjust the distance between their opponents according to the required range. In addition to the damage that the attack skills apply, they can also have debuff effect. By appliying debuff, they can have a negative impact on the opponent with a certain percentage. Players concentrate more on using these skills and try to overcome their enemies. In order for the AI agent to learn how to use its attacking skills correctly, we shared the information we mentioned above as observations with the RL system (see Table 3.5).

Observation	**Description**
Damage	Damage of the attack skill, on the enemy.
Range	Application range of the attack skill.
Has Debuff	Attack skills can have debuff effect.
Debuff Percentage	Debuff percentage of the attack skill.
Debuff Kind	Debuff effect kind of the attack skill.

Table 3.5: Attack Skill Info Observations

Buff/Debuff Skill Info Observations Players use their buff skills to strengthen their own character. Buff skills, that affect different properties of the character according to their types and increase the values of these properties for a certain period of time. For example; The agility buff to be used before the battle will allow players to move quickly throughout the battle. On the contrary, debuff skills decrease the characteristics of the opponents. Some have a periodic feature which enables applying negative effects over and over again. For example, when using a debuff skill that deals periodic damage to an opponent, the opponent's health will decrease for a certain duration of time. Players tend to use their buff and debuff skills at the beginning of the PvP to gain longer advantage over their opponents. In order for the AI agent to learn how to use buff and debuff skills correctly, we shared the information we mentioned above as observations with the RL system (see Table 3.6).

Observation	**Description**
Duration	Duration of a buff skill.
Amount	Amount of the effect of a buff skill.
Buff Kind	Which property will be effected from a buff skill.
Is Periodic	Periodicity of a buff skill.

Table 3.6: Buff/Debuff Skill Info Observations

Raycast Observations Ray casting is to draw a line or vector from a certain point to another point in a 3D area. Ray indicates whether there are collision with other game objects. This method, which is widely used in FPS games, is used for the interaction of the player with the objects where he is looking. In MMORPGs, players interact with other players and game objects. Players see objects in the direction they are looking, and the human brain automatically identifies them. The AI agent also needs to be able to detect walls and other players on the playing field. We used the ML-Agent RayPerceptionSensor3D component, which transmits the information we mentioned above to the RL system so that the AI agent can interpret objects around it. In this way, we have provided the necessary information to find the enemy on the playground and not to hang on the walls.

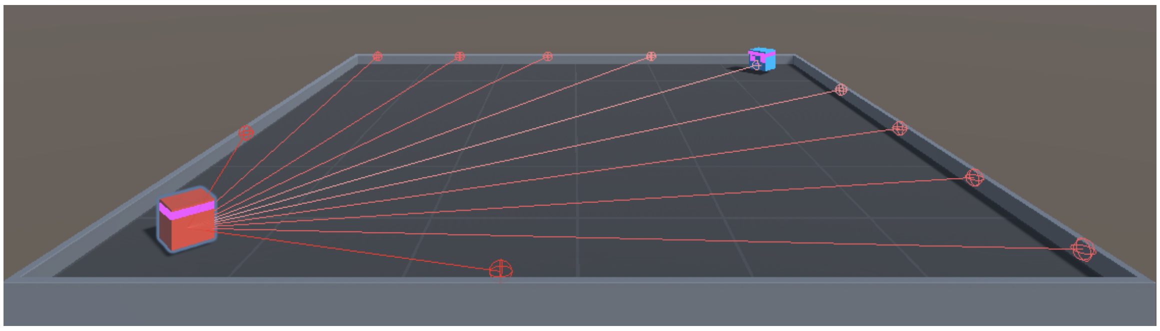


Figure 3.1: Ray Perception Sensor 3D

3.2.2 Manual Reward Design

We first design a simple reward function manually, which constitutes a basis for the combined reward strategy and also is used for comparison with the MCTS reward strategy (see Section 3.4) As mentioned in Section 2.2.1, reward functions can be categorized under two in means of their shape among the training steps (see Figure 2.2) In order to win a PvP battle in an MMORPG, the agent must learn complex action sequences composed of different kinds of actions like using skills, approaching enemies, attacking and using potions. It is crucial to take these actions in the right place and time one after another. Therefore feeding the reinforcement learning system with proper rewards during the intermediate steps of the battle is of great importance. For this reason, we employed a shaped reward strategy while designing our manual reward function.

In order to calculate the reward amount for each intermediate step we first categorized the rewards according to the action types that cause the same type of change in the game state. By preparing PvP scenarios (see Appendix C), we calculated the reward of each category according to its effect on winning the PvP. The reward categories are displayed in Table 3.7 and described below in detail.

Reward Category	**Reward Per Unit**	**Type of Change**	**Covered Actions**
Health	0.12	Increase in player's health.	Health potion usage and health-boosting buff skills.
Mana	0.04	Increase in players mana.	Mana potion usage and mana-boosting buff skills.
Property	0.1	Increase in player's speed, defense and damage.	Defense, speed and damage-boosting buff skills.
Damage	0.002	Applied damage to the enemy.	Attack skills, health debuffs and melee attacks.
Debuff	0.1	Applied debuff effect to the enemy.	Debuff skills and debuff effects applied from attack skills.

Table 3.7: Reward Categories

Health Reward Category It is the reward the agent receives when its health increases. To win a PvP, the agent must be able to survive first. For this reason, reward per unit for this category is the highest. This reward covers health potion usage and health-boosting buff skills.

Mana Reward Category It is the reward the agent receives when its mana amount increases. In addition to survival, the agent must dominate his opponent using his skills. The agent's mana value decreases as it uses skills, and if he does not refill it, he cannot use skills anymore. For this reason, it gains reward as the amount of mana increases, but not as much as health reward. This reward covers mana potion usage and mana-boosting buff skills.

Property Reward Category The character's defense, speed, and damage properties allow it to outperform his opponents throughout a PvP. The agent earns high reward when it increases these properties. This reward covers defense/speed/damage-boosting buff skills.

Damage Reward Category The agent must damage its enemy to defeat him. There are many actions to do this. The most used actions in a PvP are actions that damage the enemy. Therefore, the reward per unit is low. But the agent wins the biggest cumulative reward from this category. This reward covers attack skills, health debuffs and melee attacks.

Debuff Reward Category Although debuff effects are not powerful enough to bring victory, they are important to outperforming the opponent. During PvP between equal strength players, the debuff effects can be game changer. Therefore, when the agent applies a debuff to his opponent, he earns high reward. This reward covers debuff skills and debuff effects applied from attack skills.

We also feed the RL system with rewards independent of the action. In MMORPGs, actions need to be taken very quickly to win a PvP. We give a small negative reward for each episode step to make the agent take actions quickly which we called time penalty. In addition to rewards per intermediate steps, a final reward is given at the end of the PvP. If the agent reaches its goal and defeats its opponent, gets the biggest reward, whereas if it loses, it receives the lowest reward. The details of these rewards are displayed in Table 3.8.

Action Free Rewards	**Reward**	**Description**
Win	1	Big positive reward for winning the battle.
Lose	-1	Big negative reward for losing the battle.
Time Penalty	-1 / max step	Negative reward at each step for quick response.

Table 3.8: Action Free Rewards

By putting them all together, the reward at each intermediate step is calculated with the formula below:

$$RPS = \sum_{ForEachCategory} RPU * NU + timePenalty \tag{3.1}$$

Here RPS denotes the reward per step, RPU denotes the reward per unit (see Table 3.7) and NU denotes the amount of change in terms of number of units.

3.2.3 MCTS Reward Design

MCTS, a tree search algorithm, has yielded successful results in game environments with many actions and possible states [7, 2, 22]. It positions the current game state as the root node and simulates the game based on the randomly selected actions until it reaches a certain depth or terminal state. It transmits the result of the simulations to the parent nodes. It tries to determine the most promising action that can be performed by making maximum evaluation within a certain period of time. A more detailed explanation of the algorithm is given in Section 2.5.

As indicated in Section 2.2.1, how to write a proper reward function for RL problems, especially for the ones in partially observable environments, is a daunting open question. PvP battles in MMORPG games constitute a good example to this kind of problems, with its vast number of possible action combinations, which directly effect the success of the player. The timing and order of an action is critical, where the significance of an action may vary in different states. As an instance, using a stun effect just before the enemy's strongest skill, instead of using it at a random time, can change the course of the battle. It is quite a complicated and almost impossible task to try to cover all such cases and shape the reward function according to them. Hereby, writing complex reward functions that can handle different situations is challenging in terms of both maintainability and learning. In PvP battles with real players, the experience of a player affects its action decision a lot, which forms an internal reward mechanism capable of estimating the course of the battle. Considering that, we intended to design a similar reward mechanism which does not only consider the current state but also tries to foresee the future states. At this point, we decided to utilize MCTS through its forward simulation phase, not for action selection like in the similar works[7], but for shaping the reward of a state-action pair.

With our reward shaping scheme with MCTS, we basically evaluate how good it is to perform the action given by the RL system compared to other possible actions at the current state. As mentioned in Section 2.5, the MCTS algorithm calculates UCB values for each action by performing random forward simulations and after conducting the search for a designated amount of time, the actions with higher UCB values are preferred for performing better in the game. At each time a new action is selected by the RL system, we run the MCTS algorithm for a specified amount of time, by taking the current state of the game as root node. After the tree search is finished, we calculate a reward value for this state-action pair by comparing the UCB value of the selected action with the UCB values of the other actions that could be performed. The possible actions for a given state is masked same as the RL System as we mentioned in Section 3.3.3. We then return this reward value to the RL system. The overview diagram of the connection between RL system and MCTS is shown in Figure3.2. The UCB value comparison gives us a measure of how preferable it is to select the action. We gave the rewards (see 3.9) by comparing the UCB value of the selected action with the UCB values of the actions that can be performed.

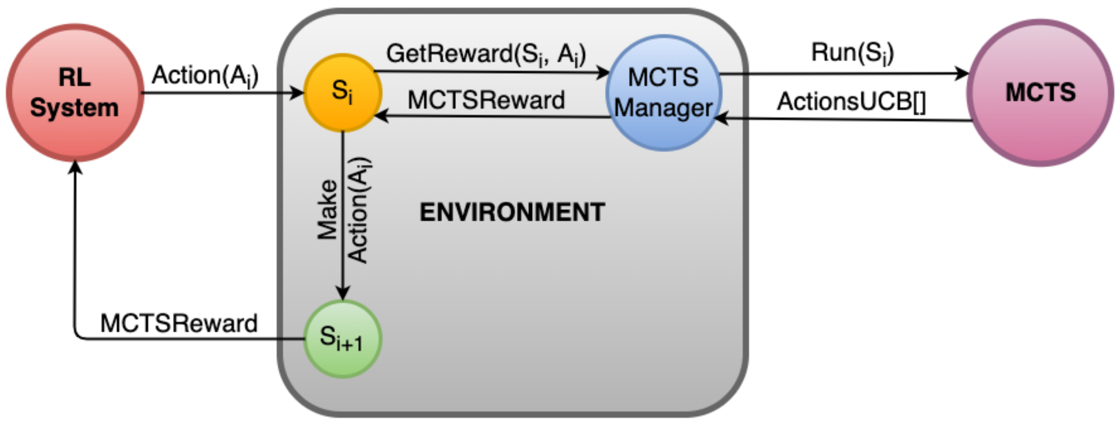


Figure 3.2: Reward Shaping with MCTS

The MCTS algorithm is used for reward shaping. MCTS runs for the current state (S_i) and UCB values are calculated for possible actions. The reward (MCTSReward) is calculated by comparing the UCB value of the action (A_i) selected by RL System with UCB values of the possible actions (Action[]). After the selected action (A_i) is performed, the calculated reward (MCTSReward) is sent to RL System.

Selected Action	**Reward**
Move action	0
Highest UCB action	0.05
Other actions	-0.05

Table 3.9: MCTS Rewards

One of the main concerns about MCTS is the need of compromise between level of accuracy and the time efficiency of the tree search. Number of forward simulations, branching factor and depth of the built search tree all effect this compromise. Since MCTS is an algorithm that performs random forward simulations, the accuracy of the resulting UCB values increases as the number of simulations increase. On the other hand, since MCTS is run after every action selection of the training, the time it consumes directly effects the training time. For this reason, we first limited the maximum number of simulations.

In a PvP, terminal state is reached when one of the players loses the battle. A random simulation is intended to end when a terminal state is reached, which determines the depth of the search tree. In games like MMORPG, it can take high number of actions to reach the terminal state, which increases the depth of the search tree. Therefore, we need to limit the depth for efficiency concerns, which causes the need for an evaluation function 3.2. We calculated the survival time by dividing the remaining health information of the players by the diminished health information. We assumed the player with bigger survival time as the winner of the game.

$$Player1SurvivalTime = Player1RemainingHealth \div Player1DiminishedHealth$$
$$Player2SurvivalTime = Player2RemainingHealth \div Player2DiminishedHealth$$
$$Winner = \begin{cases} \text{Player1}, & \text{Player1SurvivalTime} \geq \text{Player2SurvivalTime} \\ \text{Player2}, & otherwise \end{cases} \tag{3.2}$$

Both for optimization and also for more accurate results, we changed some behaviors and disabled some actions in certain situations. In MMORPGs, the player's mobility is very sophisticated, the player can move in any direction. In PvP battles, the players need to approach each other in order to apply skills and actions. In our RL system we want the player agent learn to approach its enemy on time, so any-direction random movements are included in the action list. But including them in the branches of MCTS leads to some serious problems. First of all, the branching factor increases excessively. Moreover, in the forward simulation phase, if the agent moves randomly, an astronomical amount of tree depth is needed for the agents to find each other. For this reason, we defined the behavior of the movement action in the MCTS as moving towards the enemy with an increased speed. In a similar way, during training, RL may choose to move in different directions while attacking the enemy, and we expect the agent to learn that it must attack constantly whenever possible. However, in MCTS, if move action is chosen when the agent has the opportunity to attack, then the battle can stuck in an infinite loop and it becomes nearly impossible for the agents to defeat each other within a reasonably limited tree depth. As a solution, we disabled the movement action of MCTS while it is capable of attacking. Apart from these all other actions and behaviors are the same as the RL System.

3.3 Implementation Details

3.3.1 Environment

We trained our agents on the following environment:

- Operating System: Windows 10 Enterprise 64-bit
- Processor: Intel(R) Core(TM) i7-8700 CPU @ 3.20 GHz (12 CPUs)
- Graphics: Nvidia GeForce GTX 1080
- Memory: 16 GB
- Unity: v2019.3.13f1
- ML-Agents: v1.0.2
- TensorFlow: v2.0.2

3.3.2 Unity

In an MMORPG it is very important that the characters interact with each other in real time. So, we needed a game engine and we used Unity with C# programming language. The main reasons why we chose Unity:

- It provides development and testing environment as well as game engine.
- We have a past Unity experience, and we are familiar with it.
- It has a large asset store and community.
- It enables cross-platform game development.
- It supports both 2D / 3D graphics.
- It provides C#, JavaScript and Boo programming language options.
- It has a great toolkit for AI developers called Machine Learning Agents (ML-Agents).

3.3.3 ML-Agents

ML-Agents is a bridge between machine learning libraries such as Tensorflow, providing easy and intuitive interfaces for artificial intelligence developers. Agents can be trained by reinforcement learning, imitation learning, neuroevolution, or other machine learning methods. The benefits of ML-Agents are as follows [23]:

- Unity environment control from Python

- 10+ sample Unity environments
- Two deep reinforcement learning algorithms, Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC)
- Support for multiple environment configurations and training scenarios
- Train memory-enhanced agents using deep reinforcement learning
- Easily definable Curriculum Learning and Generalization scenarios
- Built-in support for Imitation Learning
- Flexible agent control with On Demand Decision Making
- Visualizing network outputs within the environment
- Simplified set-up with Docker
- Wrap learning environments as a gym
- Utilizes the Unity Inference Engine
- Train using concurrent Unity environment instances

In ML-Agents the training consists of tests called episode, where the game is replayed every time. ML-Agents has 'Academy' which orchestrates the training steps and AI techniques. The academy communicates with external Python training process (Tensorflow) and passes collected data from episodes in order to optimize its ANN. The collected data is observations gathered from the game and the ANN outputs are actions that agents will do. It can control more than one playground at the same time which enables collecting more training data simultaneously. It saves time by simulating multiple playgrounds in an accelerated manner. It also resets playground and agents before each episode. The main responsibility of the academy is controlling agents with brain parameters called 'Behavior Parameters'. Like the human brain, the behavior parameters constitute the decision-making mechanism for ML-Agents. They act as a bridge between the academy and the agents. They also determines number of steps of the training and action space of the agents.

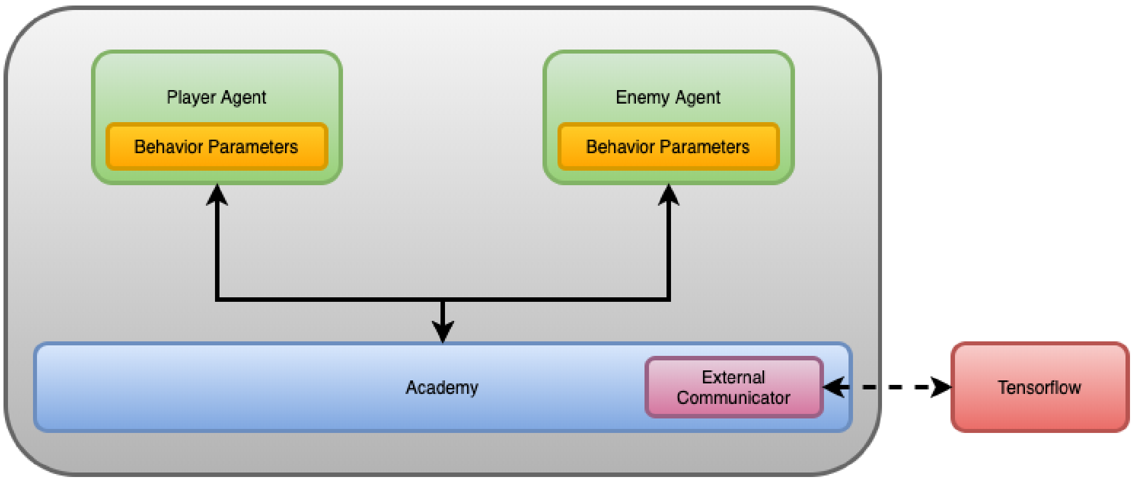


Figure 3.3: Learning Environment

The firs part of the ML-Agents integration was creating the 'BattleArena', which is a rectangular area representing the playground. It has a player and enemy character on it and is responsible for placing them in the arena and resetting them. In every reset, the battle arena randomly places the characters in the game area.

The second part of ML-Agents integration was the implementation of the Agent. We created the 'PlayerAgent', which inherits the 'Agent' component of the ML-Agents. The player agent performs the actions sent by the ANN, collects observations and determines the reward by evaluating the situation resulting from these actions. During the battle, the character's health, mana, and strength will change, so that at every episode, these properties must return to their original values. 'OnEpisodeBegin' method does this resetting job.

Player characters perform actions such as moving, attacking, and using skills during the battle. The Behavior Parameters provide the connection between these actions and the ANN outputs. Brain parameters map these actions into branches. Branches are used to group related actions. Actions that can not be done at the same time should be put into the same branch. Also branch provides option for discrete and continuous actions. It is decided if the actions are discrete or continuous according to the need in the game. For instance; if the height of the character is same level each time the jump action comes, discrete action should be selected. Conversely, if the height of the character changes according to the intensity of the jump action, continuous action should be selected. In PAPI online, we have mapped actions to the brain parameters as discrete actions. It consists of three branches. The first branch consists of 5 discrete actions, 4 for right, left, front and back movement actions, 1 for idle. The second branch consists of 9 discrete actions, 7 for skill actions, 1 for attack and 1 for idle. The third branch consists of 3 discrete actions, 1 for idle, 1 for health potion and 1 for mana potion. The reason we separated the move actions, skill actions and potion actions was that the player could use his skills or potions while on the move. We overrided the ’OnActionReceived’ method to process the actions given by the ANN. It finds and performs what action should be taken for each branch.

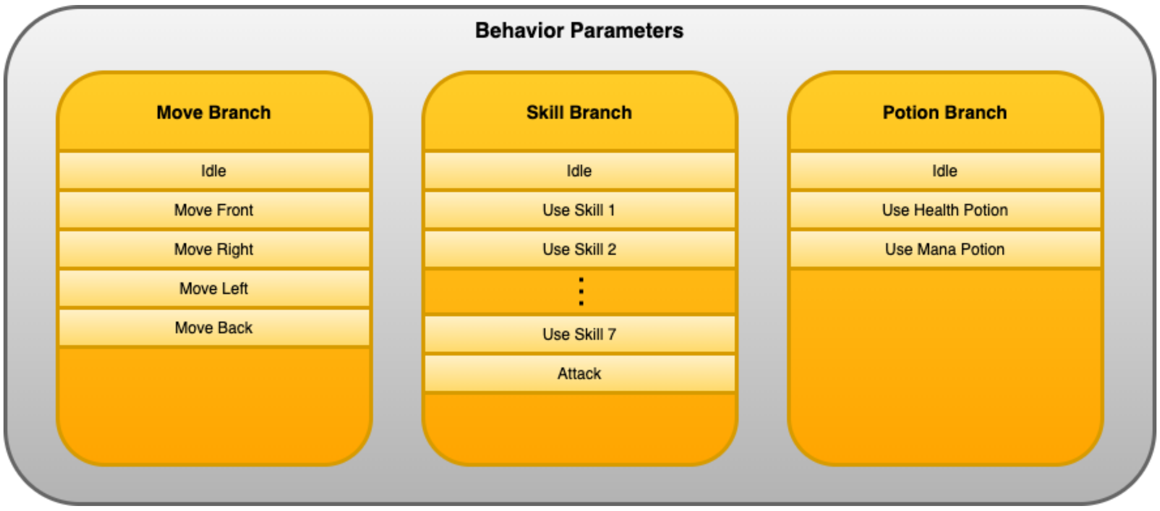


Figure 3.4: Behavior Parameters

In MMORPGs, when player characters use a skill, this skill is deactivated for a certain time. The skill becomes available again when the time out expires. In addition, a player cannot move and use any of his skills when the character is under the influence of the stun. When an action is not available, it will be a matter for artificial intelligence to try to apply it. This will result in the actions taken not working. Since each skill’s timeout will differ, it will make learning difficult.To achieve this, we overrided the ’CollectDiscreteActionMasks’ method. It provides ’DiscreteActionMasker’ object that allows you to mask inactive actions in the next step. It is not possible to mask all actions in a branch or continuous actions. If a character is under the influence of stun, we masked both the actions in the movement branch and the actions in the skill branch, except idle ones. If there was no stun effect, we masked the skills that could not be used due to the distance between the enemy and were inactive because of they were used. We also masked the potion actions if players health or mana is not reduced as enough as the potion fill amount.

3.3.4 Training Parameters

We trained our agents with the parameters called hyperparameters displayed in Table 3.10. We used the Proximal Policy Optimization (PPO) as *Trainer* to find the best policy by learning the MMORPG dynamics of the agent. As a result of trial and error, we determined the *Max Step* value as 50 million. Due to the complexity of the PvP problem, we increased the value of *Hidden Units* to 256. We increased the *Buffer Size* by 10 times in order to be able to do parallel training with 10 Unity instances. Since the training time is uncertain, we have set the *Learning Rate Schedule* value to constant. As actions taken in PvP can produce long-term results, we increased the *Time Horizon* value to 1000. We increased the *Save Steps* value to 50000 to wider the agent's range of skill levels and play strategies. As seen in Table 3.10, we did not make critical changes in the remaining parameters and we used the default values of ML-Agents.

Parameter Name	**Value**
Trainer	PPO
Max Steps	5.0e7
Hidden Units	256
Epsilon	0.2
Batch Size	1024
Buffer Size	102400
Lambd	0.95
Learning Rate	3.0e-4
Learning Rate Schedule	Constant
Memory Size	128
Normalize	False
Num Epoch	3
Num Layers	2
Time Horizon	1000
Sequence Length	64
Summary Frag	10000
Use Recurrent	False
Vis Encode Type	Simple
Reward Signals -> Extrinsic -> Strength	1.0
Reward Signals -> Extrinsic -> Gamma	0.99
Self Play -> Window	10
Self Play -> Play Against Latest Model	0.5
Self Play -> Save Steps	50000
Self Play -> Swap Steps	50000
Self Play -> Team Change	100000

Table 3.10: Hyperparameters

3.3.5 Warrior Character

We created a warrior character in our game and determined the characteristics of it. The amount of damage and health is high, the speed is moderate and attack range is close. The details of the properties are as follows:

Name	Health	Mana	Defense	Damage	Speed	Attack Range	Health Pot	Mana Pot
Warrior	500	500	10	12	1	1	10	10

Table 3.11: Warrior Properties

Warrior has various attack skills. Some attack skills have low damage and short timeouts. We designed the attack skills so that their timeout increases as their strength increases. In addition, we have added two debuff effects that create stun and damage effects to the opponent. The details of attack skills are as follows:

Name	Consumption of Mana	Power	Timeout	Range	Debuff	Percentage of Debuff
Attack1	10	15	3s	5	-	-
Attack2	15	20	4s	5	Debuff1	100%
Attack3	20	25	5s	5	Debuff2	10%
Attack4	25	35	6s	5	-	-
Attack5	35	45	7s	5	-	-

Table 3.12: Warrior Attack Skills

We have added buff skills to the warrior that increase the speed and amount of damage. It can move faster and deal more damage by using them. Details of the buff and debuff skills are as follows:

Name	Consumption of Mana	Buff Kind	Timeout	Duration	Amount	Periodic
Debuff1	-	STUN	-	2s	-	-
Debuff2	-	HEALTH	-	10s	4	true
Buff1	60	DAMAGE	60s	60s	2	false
Buff2	70	SPEED	60s	60s	0.5	false

Table 3.13: Warrior Buff/Debuff Skills

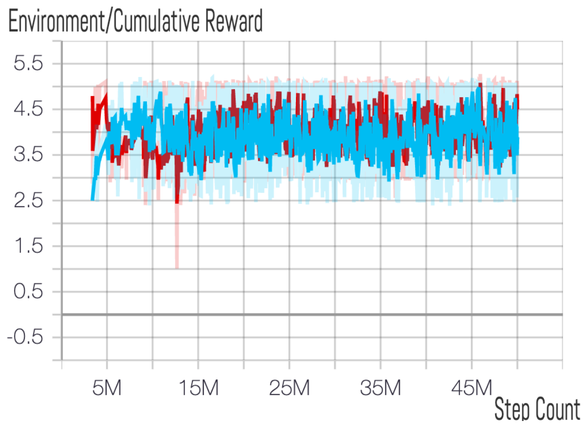
3.3.6 MCTS

For the MCTS integration of PAPI Online, we first created the "GameState" which contains the status of the player and the enemy character and turn information. Although actions in MMORPGs seem to be performed simultaneously, the game engine handles these actions in the order of reaching. That's why we created the MCTS structure by thinking like a turn-based game: Players take actions sequentially, when an action is taken, turn passes to the other player and this routine continues until the terminal state. Secondly, we created the "Game" that manages the possible actions that can be performed in a state in the game and provides the transition between the states. This class provides the realization of the game independently from the Unity game. This is because Unity objects that MLAgents work with should not be affected by the game simulation with MCTS. Then, we created the "MonteCarloNode" to build the tree structure. This class contains information about action, situation, parents and children. Finally, we created the most important part, the "MonteCarlo". This class, which is an MCTS implementation, simulates the game until it reaches the terminal state or a certain depth (Random Simulation) through the node it creates by randomly choosing among the actions that can be performed in the current situation (Expansion). Then it transmits the calculated information to the root node (Backpropagation). Since we set up the game simulation as turn-based, after each action, the turn passes to the opposite player and the chosen random action is performed on behalf of that player. As in MLAgents integration, the rules of the game had to be applied here. For instance; if the action results with skill usage, this skill is out of use for a certain time. Or the player cannot take action for a certain period of time if it is under the influence of the stun. This mechanism had to be done independent of the Unity timer, because MCTS simulation of the game had to be performed much faster than Unity plays. To do this, we updated the timers of players' and enemy characters' in every turn change.

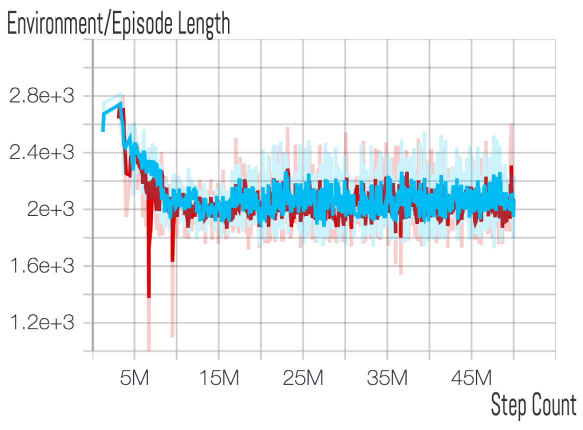
3.4 Results

In this section, we first give the results of the training by Deep RL with the proposed reward design strategies with relevant Tensorboard charts. For both of the trainings, we used the parameters mentioned in Section 3.3.4 and observations mentioned in Section 3.2.1. In training with MCTS reward design, we ran the Monte Carlo Tree Search with maximum 50 simulations and maximum depth value of 50.

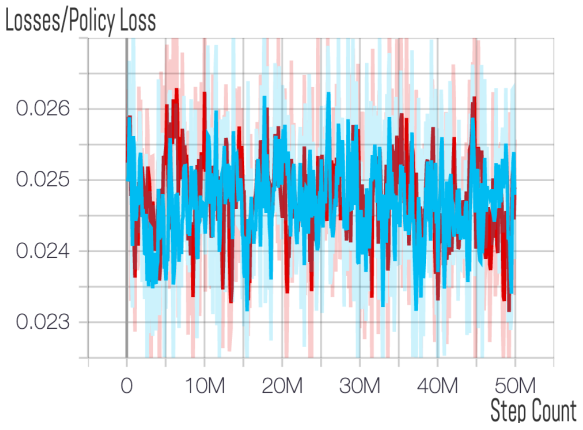
Tensorboard charts of training with manual reward design and MCTS reward design can be seen in Figure 3.5 and Figure 3.6, respectively. As it is understood from the graphics in both of the trainings, cumulative reward increased at the beginning and then continued at the same level. The episode length decreased as the agent learned how to use skills and a little increased as it learned how to use potions. The entropy value, which indicates the randomness of the policy, has approached zero over time. Value Estimate and Value Loss are increased as the Cumulative Reward increased. Value Estimate and Value Loss increased as expected according to the Cumulative Reward increase. Policy loss oscillated under the value of 1.0 as it's supposed to be.



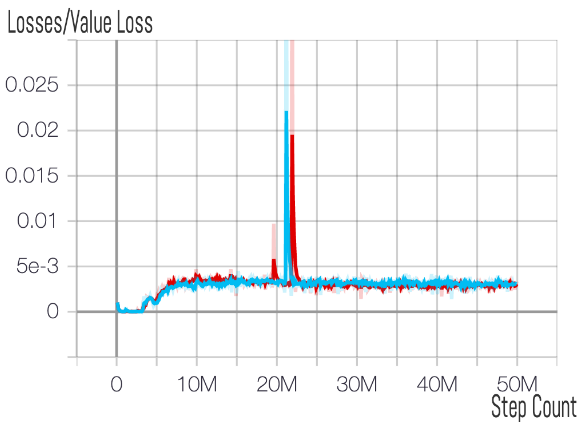
(a) Cumulative Reward



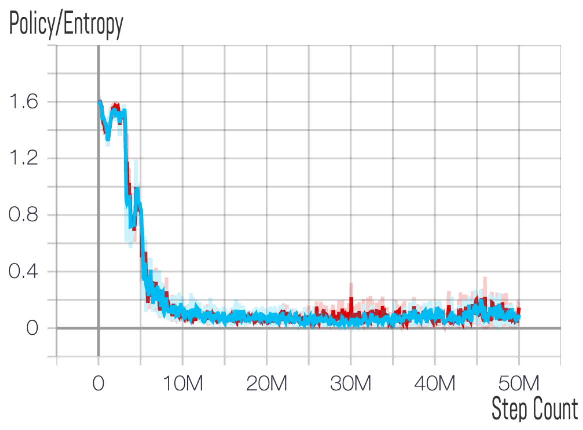
(b) Episode Length



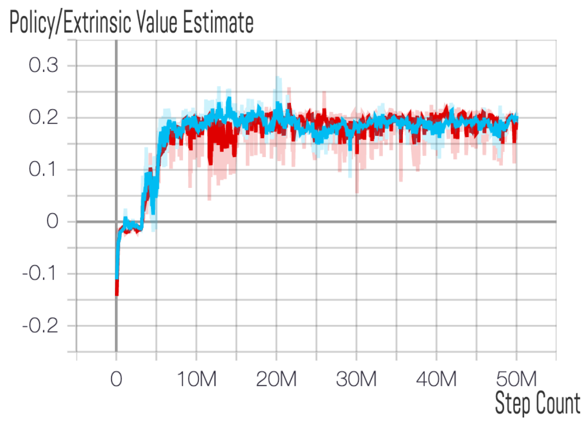
(c) Policy Loss



(d) Value Loss

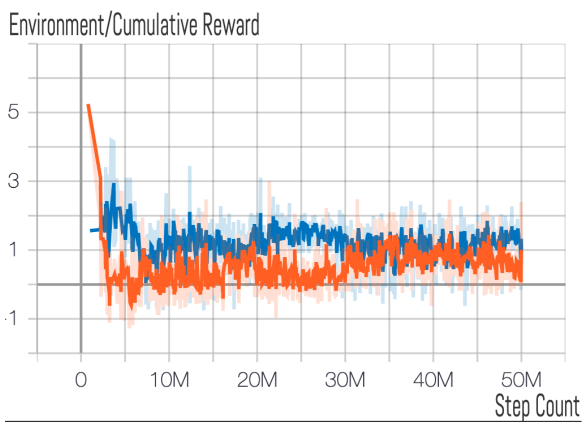


(e) Entropy

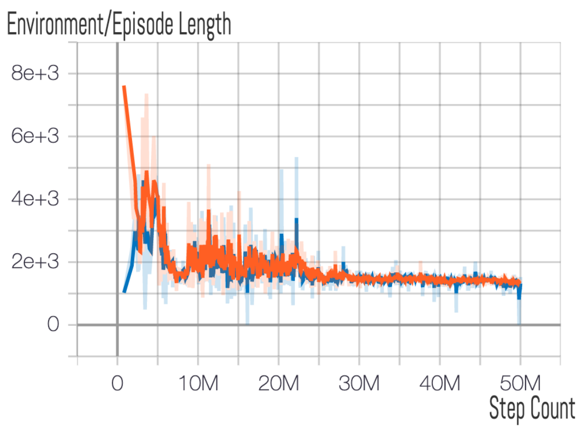


(f) Value Estimate

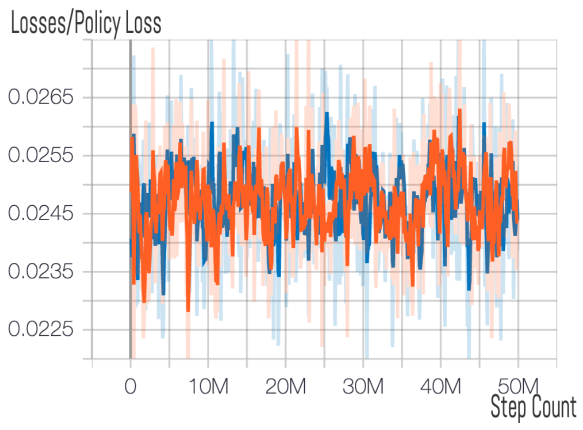
Figure 3.5: Tensorboard Charts of Training with Manual Reward Design



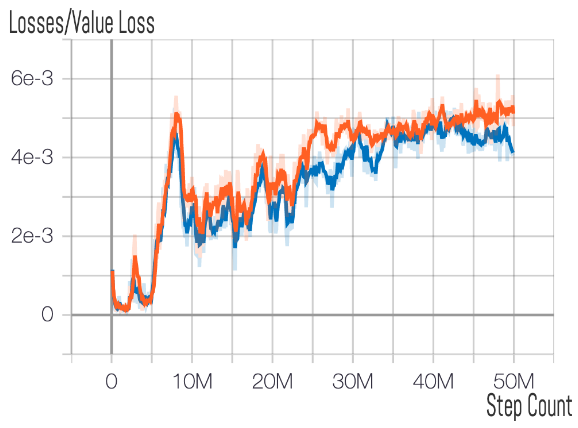
(a) Cumulative Reward



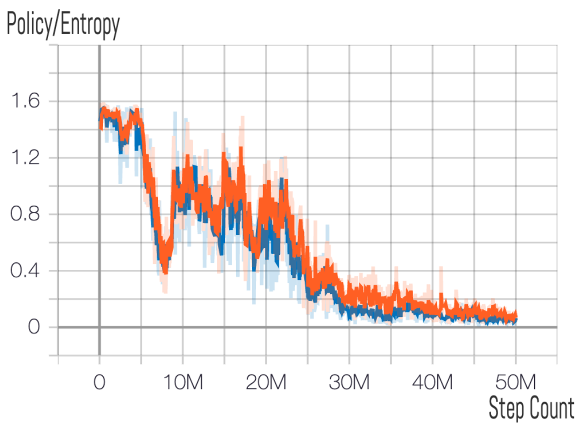
(b) Episode Length



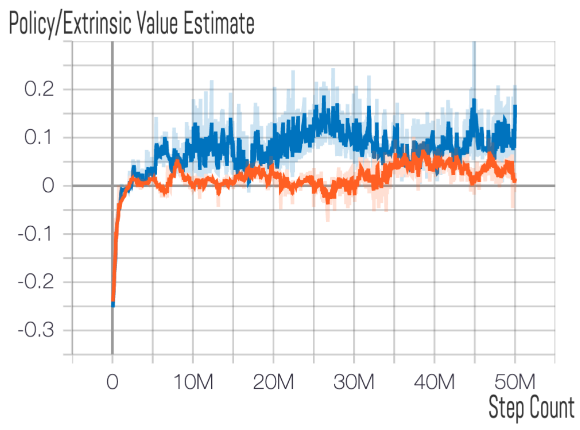
(c) Policy Loss



(d) Value Loss



(e) Entropy



(f) Value Estimate

Figure 3.6: Tensorboard Charts of Training with MCTS Reward Design

In order to evaluate success and similarity of the AI agent behaviors to real human players, we determined the following typical behaviors of an experienced human player. In Table

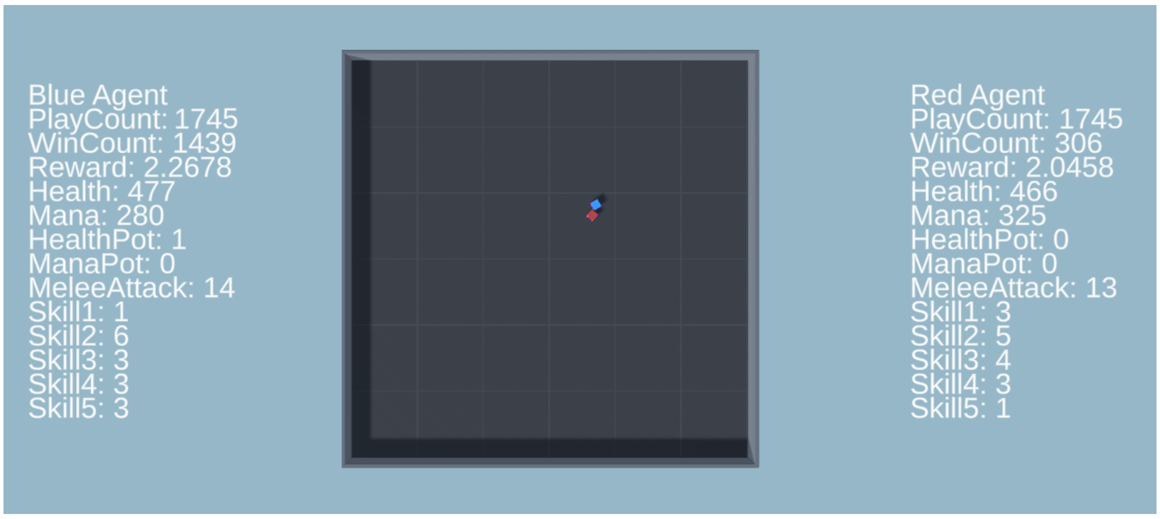
3.14, we displayed the results of both reward strategies in terms of compatibility with these determined behaviors.

Experienced Human Player Behaviors	**Manual Agent**	**MCTS Agent**
Using buff skills immediately and filling the lost mana.	✓	✓
Moving towards the enemy in order to attack.	✓	✓
Smooth mobility ability.	✓	×
When faced with the opponent, using the skill that has stun effect firstly.	✓	✓
After applying the stun effect to the opponent, using other skills starting with the strongest one.	✓	✓
Starting melee attacks after using all the skills.	✓	✓
Ability to use the maximum number of skills and make melee attacks.	×	✓
Filling health and mana when their value fall below a certain level.	✓	✓

Table 3.14: Agent Behaviors

Manual vs MCTS Results

TestCase1: After training the agents with the manual and MCTS reward shaping methods separately, we had them make PvP battles with each other. The agent trained with MCTS reward design won 82.4% of these PvPs (See Figure 3.7).

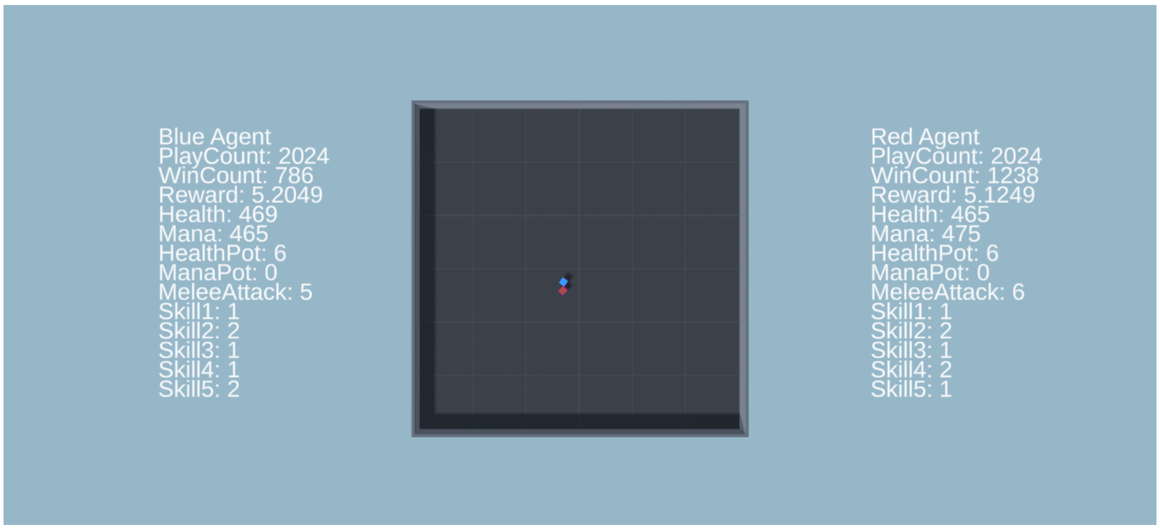


```
Blue Agent
PlayCount: 1745
WinCount: 1439
Reward: 2.2678
Health: 477
Mana: 280
HealthPot: 1
ManaPot: 0
MeleeAttack: 14
Skill1: 1
Skill2: 6
Skill3: 3
Skill4: 3
Skill5: 3
```

```
Red Agent
PlayCount: 1745
WinCount: 306
Reward: 2.0458
Health: 466
Mana: 325
HealthPot: 0
ManaPot: 0
MeleeAttack: 13
Skill1: 3
Skill2: 5
Skill3: 4
Skill4: 3
Skill5: 1
```

Figure 3.7: Manual (Red Agent) vs MCTS (Blue Agent) PvP Result

TestCase2 Later, we thought that manual agent could use the skills better if we increase the damage reward in the rewards mentioned in the 3.2.2 section. For this purpose, we increased the damage reward 10 times. As a result, even though the manual agent performed better than before, it lost 61.1% of the battles (See Figure 3.8).

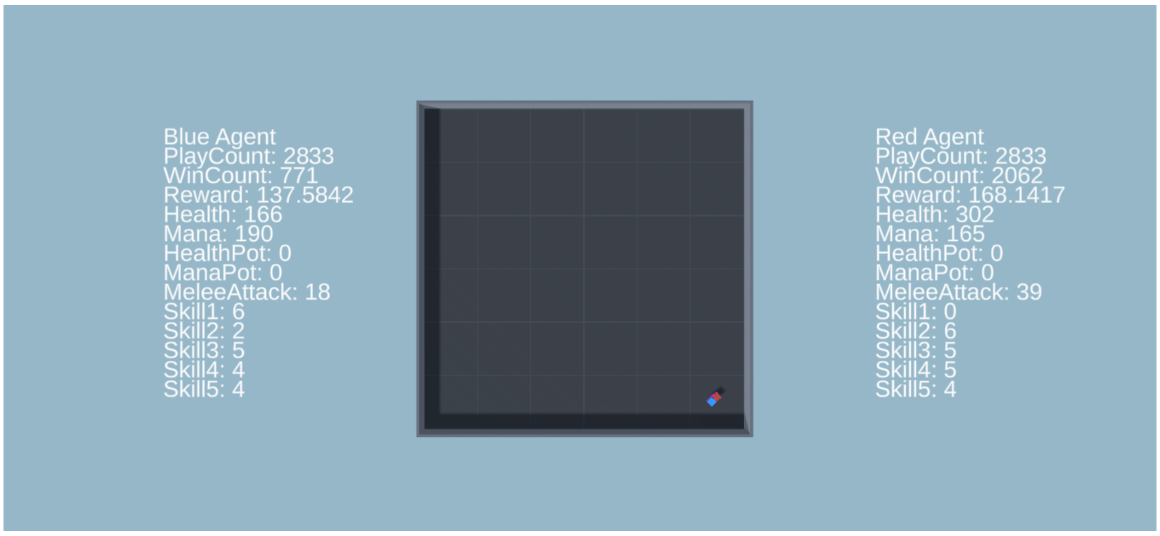


```
Blue Agent
PlayCount: 2024
WinCount: 786
Reward: 5.2049
Health: 469
Mana: 465
HealthPot: 6
ManaPot: 0
MeleeAttack: 5
Skill1: 1
Skill2: 2
Skill3: 1
Skill4: 1
Skill5: 2
```

```
Red Agent
PlayCount: 2024
WinCount: 1238
Reward: 5.1249
Health: 465
Mana: 475
HealthPot: 6
ManaPot: 0
MeleeAttack: 6
Skill1: 1
Skill2: 2
Skill3: 1
Skill4: 2
Skill5: 1
```

Figure 3.8: Manual (Blue Agent) vs MCTS (Red Agent) PvP Result 2

TestCase3 Based on the increased wins of the manual agent, we increased the damage reward 100 times than the reward mentioned in the 3.2.2 section. But this time, on the contrary than expected, the winning percentage of the manual agent decreased. It lost 72.7% of the battles (See Figure 3.9).

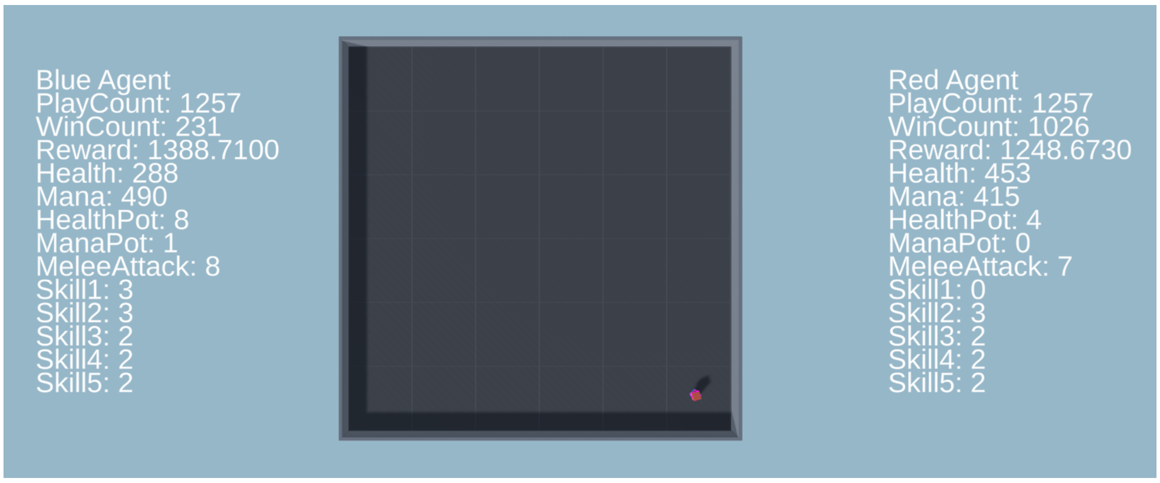


Blue Agent
PlayCount: 2833
WinCount: 771
Reward: 137.5842
Health: 166
Mana: 190
HealthPot: 0
ManaPot: 0
MeleeAttack: 18
Skill1: 6
Skill2: 2
Skill3: 5
Skill4: 4
Skill5: 4

Red Agent
PlayCount: 2833
WinCount: 2062
Reward: 168.1417
Health: 302
Mana: 165
HealthPot: 0
ManaPot: 0
MeleeAttack: 39
Skill1: 0
Skill2: 6
Skill3: 5
Skill4: 5
Skill5: 4

Figure 3.9: Manual (Blue Agent) vs MCTS (Red Agent) PvP Result 3

TestCase4 In order to find the optimum damage reward, we increased the damage reward 20 times than the reward mentioned in the 3.2.2 section. But it lost 81.6% of the battles (See Figure 3.10).

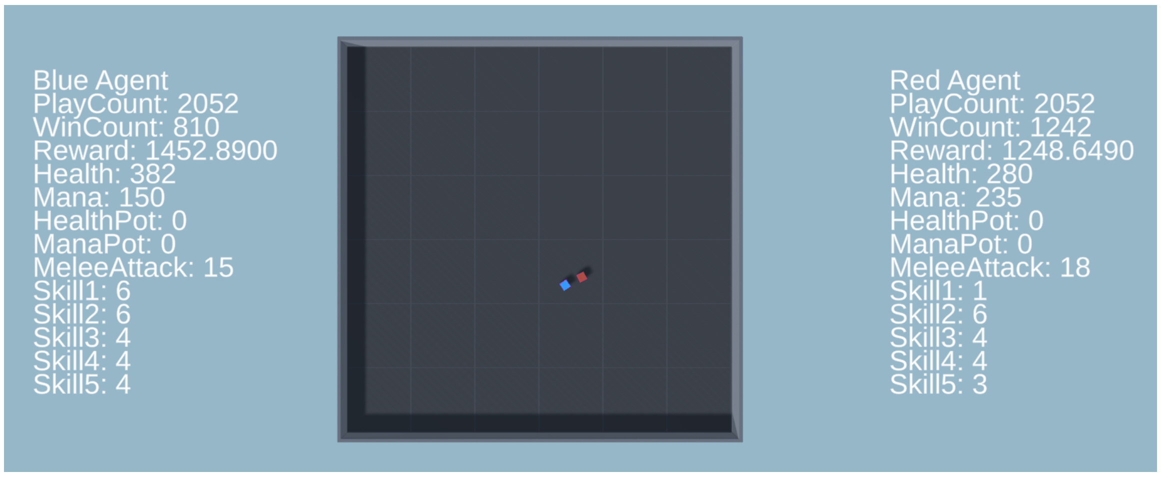


Blue Agent
PlayCount: 1257
WinCount: 231
Reward: 1388.7100
Health: 288
Mana: 490
HealthPot: 8
ManaPot: 1
MeleeAttack: 8
Skill1: 3
Skill2: 3
Skill3: 2
Skill4: 2
Skill5: 2

Red Agent
PlayCount: 1257
WinCount: 1026
Reward: 1248.6730
Health: 453
Mana: 415
HealthPot: 4
ManaPot: 0
MeleeAttack: 7
Skill1: 0
Skill2: 3
Skill3: 2
Skill4: 2
Skill5: 2

Figure 3.10: Manual (Blue Agent) vs MCTS (Red Agent) PvP Result 4

TestCase5 We also combined 10 times more damage (best manual damage reward so far) with 10 times more debuff and 20 times more stun debuff reward than the reward mentioned in the 3.2.2 section. As a result, even though the manual agent performed better than before, it lost 60.5% of the battles (See Figure 3.11).



```
Blue Agent
PlayCount: 2052
WinCount: 810
Reward: 1452.8900
Health: 382
Mana: 150
HealthPot: 0
ManaPot: 0
MeleeAttack: 15
Skill1: 6
Skill2: 6
Skill3: 4
Skill4: 4
Skill5: 4
```

```
Red Agent
PlayCount: 2052
WinCount: 1242
Reward: 1248.6490
Health: 280
Mana: 235
HealthPot: 0
ManaPot: 0
MeleeAttack: 18
Skill1: 1
Skill2: 6
Skill3: 4
Skill4: 4
Skill5: 3
```

Figure 3.11: Manual (Blue Agent) vs MCTS (Red Agent) PvP Result 5

As can be understood from the test results (See Figure 3.12), the MCTS reward shaping method has developed better strategy and prevailed against the manual reward shaping method. The most important factor behind its success was that it used its skills better. It gained advantage by using its high-damaged skills faster and more than his opponent. It dealt more damage to his opponent by using multiple melee attacks instead of the least damaged skill (Attack1 in Table 3.12). It showed the same performance with its opponent in the actions other than these. It increased its power and speed by using buff skills at the beginning of the battle. It stunned its opponent by using the skill with stun effect when encountered to it. Then it used its skills, starting with the skill with the highest damage. It gained longer durability by using potions when its health or mana drops below a certain level. It put all this together, set up a good battle strategy and succeeded.

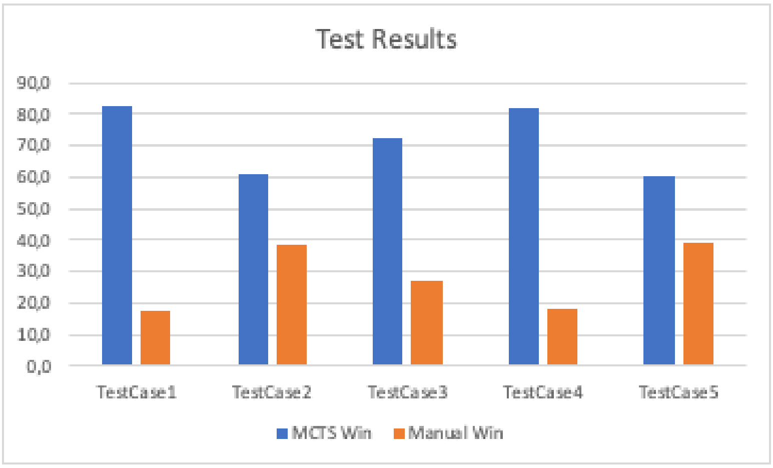


Figure 3.12: Test Results

4 CONCLUSION

In this thesis, we have implemented two reward shaping methods for reinforcement learning algorithms developed for PvP in MMORPGs. In complex games, the agent has to fulfill many tasks in order to reach the goal. When applying the RL method the common technique is to direct the agent to the desired goal by manually shaping a reward function. Most of the time it is quite complicated to write a proper reward function, and the written complex methods may not meet the need in time and it becomes increasingly difficult to get the desired result. We trained several agents that can perform PvP with manual reward shaping method. We also used the Monte Carlo Tree Search algorithm, which has proven itself in complex problems, for reward shaping in reinforcement learning. In a very simple way, we directed the RL System by giving a positive reward if the performed action is found as the best action by MCTS, and negative reward if not. We compared the agents that we trained with both manual reward and MCTS reward designs on the testbed we developed, which is a general PvP battle of MMORPGs. We have shown that MCTS can be used for reward shaping in RL systems developed for PvP in MMORPGs. In addition, we made our test environment open source [1] so that everyone can benefit from it and develop different methods on it.

The proposed study has some limitations. These limitations and intended future studies are given below:

We guided RL System and MCTS not to use potions immediately. We disabled potion actions if their health or mana value is not decreased by the potion fill amount. Through this way, the potions were used effectively and the duration of the war was prolonged.

In our study, when the MCTS reaches a certain depth, we used the formula in 3.1 to calculate the winner, if the leaf node is not a terminal state. Instead, winner can be calculated with the value which is gathered by sending the current state to ANN.

In the MCTS reward shaping method, instead of only giving the best action a positive reward, we also tried to give a weighted reward in a decreasing way from good to bad. But with this method, the agent didn't learn to fight well enough.

We trained the agents solely on the basis of the warrior character and made them battle with theirself. Since there are many types of characters available in MMORPGs, agents who can play and fight with any character type can be trained as a future study.

We implemented the MCTS reward system to support using skills and potions. Agents movement ability was learned by the RL System without any reward support. In order to accelerate learning time of movement ability, MCTS can also be used.

This method, which we tested in our own testbed, can be tested by different benchmarks.

REFERENCES

[1] Karsi, T., Papionline source code, **2020 (accessed October 24, 2020)**. URL https://github.com/tarikkarsi/PAPIOnline.

[2] Coulom, R., Efficient selectivity and backup operators in monte-carlo tree search, H.J. van den Herik, P. Ciancarini, H.H.L.M.J. Donkers (eds.), Computers and Games, Springer Berlin Heidelberg, Berlin, Heidelberg, **2007**, 72–83.

[3] Yannakakis, G.N., Togelius, J., Artificial Intelligence and Games, Springer, **2018**. http://gameaibook.org.

[4] Sazaki, Y., Primanita, A., Syahroyni, M., Pathfinding car racing game using dynamic pathfinding algorithm and algorithm a$*$, 2017 3rd International Conference on Wireless and Telematics (ICWT), **2017**, 164–169.

[5] Wang, H., Gao, Y., Chen, X., Rl-dot: A reinforcement learning npc team for playing domination games, IEEE Transactions on Computational Intelligence and AI in Games, 2(1), 17–26, **2010**.

[6] Fang, Y., Ting, I., Applying reinforcement learning for game ai in a tank-battle game, 2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC), **2009**, 1031–1034.

[7] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T.P., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D., Mastering the game of go with deep neural networks and tree search, Nature, 529, 484–489, **2016**.

[8] OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H.P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., Zhang, S., Dota 2 with large scale deep reinforcement learning, **2019**.

[9] Vinyals, O., Babuschkin, I., Czarnecki, W., Grandmaster level in starcraft ii using multi-agent reinforcement learning, Nature, **2019**.

[10] Ballinger, C.A., Turner, D.A., Concepcion, A.I., Artificial intelligence design in a multiplayer online role playing game, Proceedings of the 2011 Eighth International Conference on Information Technology: New Generations, IEEE Computer Society, USA, **2011**, ITNG '11, 816–821. URL https://doi.org/10.1109/ITNG.2011.142.

[11] Kasmarik, K., Maher, M., Motivated reinforcement learning for non-player characters in persistent computer game worlds, Proceedings of the International Conference on Advances in Computer Entertainment Technology, **2006**, 3.

[12] Kasmarik, K., Modeling motivation for adaptive nonplayer characters in dynamic computer game worlds, Computers in Entertainment (CIE), 5, 5, **2007**.

[13] Lee, B.K., Park, C.S., Kim, J.H., Youk, S.J., Ryu, K.H., An intelligent npc framework for context awareness in mmorpg, 2008 International Conference on Convergence and Hybrid Information Technology, **2008**, 190–195.

[14] Maggiorini, D., Ripamonti, L., Panzeri, S., Follow the leader: a scalable approach for realistic group behavior of roaming npcs in mmo games, Artificial Life Conference Proceedings, 706–712, **2013**. URL https://www.mitpressjournals.org/doi/abs/10.1162/978-0-262-31709-2-ch101.

[15] Rhujittawiwat, T., Kotrajaras, V., Learnable buddy: Learnable supportive ai in commercial mmorpg, 9th International Conference on Computer Games: AI, Animation, Mobile, Educational and Serious Games, **2006**.

[16] Tomai, E., Flores, R., Adapting in-game agent behavior by observation of players using learning behavior trees, FDG, **2014**.

[17] Maliković, M., Schatten, M., Artificial intelligent player's planning in massively multiplayer on-line role-playing games, 26th Central European Conference on Information and Intelligent Systems, **2015**.

[18] Reeder, J., Sukthankar, G., Georgiopoulos, M., Anagnostopoulos, G., Intelligent trading agents for massively multi-player game economies., Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE, **2008**.

[19] Fujita, A., Itsuki, H., Matsubara, H., Detecting real money traders in mmorpg by using trading network., Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE, **2011**.

[20] Kesteren, M., Langevoort, J., Grootjen, F., A step in the right direction: Botdetection in mmorpgs using movement analysis, Proceedings of the 21st Belgian-Dutch Conference on Artificial Ingelligence, BNAIC, **2009**.

[21] Thawonmas, R., Kashifuji, Y., Chen, K.T., Detection of mmorpg bots based on behavior analysis, Proceedings of the International Conference on Advances in Computer Entertainment Technology, ACE, **2008**, 91–94.

[22] Gelly, S., Schoenauer, M., Sebag, M., Teytaud, O., Kocsis, L., Silver, D., Szepesvari, C., The grand challenge of computer go: Monte carlo tree search and extensions, Communications- ACM, 55(3), 106–113, **2012**. URL https://hal.inria.fr/hal-00695370.

[23] Juliani, A., Berges, V.P., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D., Unity: A general platform for intelligent agents, CoRR, abs/1809.02627, **2018**. URL http://arxiv.org/abs/1809.02627.

Appendices

A Other MMORPG Information

A.1 Other MMORPG Features

World and Maps Every MMORPG has a world where all players and Non-Player Characters (NPCs) are placed in the game world. They are interacting with each other in real time. Players start at the specified position of the world and try to discover it. The world may contain buildings, vehicles, trees, flowers, sea etc. It also contains places like marketplaces and battle arenas. All these are placed in a way that makes players feel the game story.

The game world consists of maps. Monsters and NPCs are separated into these maps according to the story of the game. Players can teleport from one map to another at special points. Entering some maps may require money or level or some special items. Maps have special names and conditions and monsters are placed into these maps according to their level and type. For example; desert monsters are placed into desert map, ice monsters are placed into winter map, jungle monsters placed into jungle map.

Dungeons Dungeons are places that contain dangerous monsters and traps. Players have to go through these dungeons to complete a mission, gain experience or acquire special items. Players enter dungeons via NPCs in the maps. There may be special conditions to satisfy before entrance such as reaching minimum level, having special items or completing precondition missions.

One or a group of players can enter these dungeons according to its difficulty level. Players must be very careful about traps, powerful monsters and bosses. There may be different types of dungeons. Maze-like dungeons cause players to be lost. Puzzle-like dungeons force players' intelligence. Players should do teamwork to survive and pass challenging stages, because if one of them dies, the team will be out of balance and cannot finish the dungeon. In order to finish the dungeons, players must reach to the end or collect the necessary items.

Non-Player Characters (NPCs) NPCs are characters that are controlled by the game itself and placed in specified places in the game world. According to the game story each of them has a different mission. Quest NPCs give players quests according to their level. Trade NPCs help players to buy or sell items. Gate NPCs enable players to teleport between maps and entering dungeons are also done through them. Dialog NPCs guide players with conversations. Craft NPCs are responsible for crafting items to make them stronger. Each MMORPG has rules which are applied by these NPCs. To give a few examples: Entering some maps or dungeons are forbidden for the players whose level is lower than the limit level. Taking some quests is not enabled before completing previous ones.

Monsters A large part of the NPCs in MMORPG games are monsters. Like PCs, monsters have physical properties and skills. These abilities are managed by finite state machines to battle against PCs. Monsters can be found alone or in groups in the game maps. According to their characteristics, monsters can attack a PC approaching them or respond to them when they attack. Monsters have levels that show their strength.

In MMORPG games, players spend a great deal of their time on fighting with monsters. They shall attack monsters in the direction of their power, otherwise they can be killed. As the PCs kill the monsters, they gain experience, money and items. Earned experience and money are calculated based on the level of the monster that was killed and the level of the player. The earned items are selected from a certain set of items. The earned item is randomly and probabilistically selected from a particular set.

Monsters called bosses are the head of a certain group of monsters. They are usually found at specific locations on maps or at the end of dungeons. They are very strong and skillful. A player may not be able to deal with the bosses. For this, multiple players come together to hunt bosses. There are very rare and valuable items that bosses bring to players. All monsters are recreated after being killed. This is called spawning. Normal monsters and bosses have different spawning times.

Items Items are the most commonly used objects in MMORPGs. Players can obtain these items from monsters, NPCs or other players. Owned items can be bought/sold at marketplaces, disposed of or exchanged with other players. Players carry their items in their inventory. Players need to use some of the items because the inventory has limited capacity. MMORPGs have a wide variety of items, and players learn what they should use over time. We can divide these items into two categories: consumable and wearable.

Consumable items often have a buff effect and are run out when used. The most consumable items used by players are potions. In MMORPGs, life and mana properties are vital. The potions of health and mana are the items that each player must keep and use at all the time. Potions like these have a permanent effect. In contrast, the effects of other consumable items that create a buff effect are temporary. Items that increase power, speed, or defense are effective for a certain period of time from the moment they are used. In addition, there are consumable items for a particular purpose. A key can be used to enter a map and dungeon or open a chest. Craft items are used to make other items stronger.

Wearable items are usually used by players to strengthen their character. MMORPGs have special wearable items for each character type. For example, there are weapons, shields, helmets and armor for warriors, and wands, hats and cloaks for magicians. So, character types can be easily distinguished from each other. There are level limits required to equip wearable items. Players need to wear items at their level to become stronger as they level up. Wearable items become stronger by combining them with some consumable items. This is called craft. In some MMORPG games, wearable items also have levels, and these are leveled in a craft-like way. Failure to craft may result in the loss of consumable craft items and money or the complete destruction of the wearable items. Wearable items change shape according to their level and craft. Thus, it can be understood how strong the equipment is used by the opposing players.

Quests Quests are tasks taken from NPCs in MMORPGs. It is shaped according to the story of the game and aims to convey this story to the players. Tasks also have levels within themselves and become increasingly difficult. When a player fulfills the task taken, he is rewarded with money, experience, or special items. Collecting specific items, finding somewhere in the world, talking with other NPCs, completing dungeons, hunting monsters and defeating bosses are examples of these tasks. Some special kind of tasks can require certain conditions like doing prerequisite tasks or reaching a certain level.

While it is very easy to do tasks at the beginning, it can take days to perform difficult tasks later. Players may find it unnecessary to fulfill quests, but they are very important for them. Since developing their characters depends on these tasks. When some difficult tasks are completed, rare items are gained, or a dungeon unlocked.

Sociality Thousands of players in the game world need to socialize while living together. MMORPGs have personal, group and global messaging capabilities. Players try to communicate with the players from the same country in their own language and with foreigner players in common language. They can chat about the game or personal or current issues. Just like in the real world, they start making friends. They reflect their own characters to the game and groupings begin to form accordingly. Players with similar character help each other to move forward in the game. Players with bad character can try to cheat and scam to go ahead of other players.

In MMORPGs, human relationships vary as in the real world. Players can build relationships that are good enough to meet each other in real life. On the contrary, they can debate with each other, be offended, and even become enemies. The groups can be dispersed, players can join the opponent groups. What makes MMORPGs so beautiful is that there is a small simulation of real life and the players try to prove themselves there.

MMORPGs organize events to motivate and entertain players. These events can be held worldwide or on days that are important to the country in which the game is played. During Christmas, Easter and Feasts some special items may be won. At such times, the amount of experience gained by players may also increase. With these activities, the players have fun with each other.

Economy In MMORPGs, the players have their own money and they use it to buy the items they need. They can exchange items with each other. As players level up, they need more powerful items and try to get them. When they have new equipment, they aim to sell old ones and make money. Like in real life, one may need something that someone doesn't need. MMORPGs have areas called marketplaces for buying and selling items. Everything you need can be found there. In addition, players can communicate via global or private messages and then trade items.

So, how is the value of the goods determined? The value of the items is determined by the seller player according to its rarity and need. Buyer players choose the cheapest among the sellers who sell the same item. Thus, there is competition between the sellers and the sellers make discounts on the prices of the items. The items used in the craft of very important equipment are more valuable and have higher prices. As the demand for an item falls, its price decreases, on the other hand, the price of goods whose demand increases increases gradually. As in the real world, there is a supply / demand balance in the economy of MMORPGs. Some players make money by trade. They try to make a profit by buying the goods cheaper than the market price.

A.2 Other MMORPG Strategies

Leveling Player characters have a level property, and their strength increases as players level up. In addition, a certain level is required to access some maps and dungeons. So, players want to level up. The required experience must be gained to reach the next level. As the amount of experience to be gained increases exponentially at advanced levels, it becomes increasingly difficult to pass to the next level.

There are two ways to gain experience in MMORPGs. Hunting the monsters or fulfilling the quests. Players can gain experience by doing the quests, but this is usually not enough for leveling up. Players who want to jump level must fight with monsters. In MMORPGs experience gained from monsters is balanced according to their level. Players gain little experience when they kill monsters that are lower than their levels. When monsters that are on the same level as the player are killed, the level can be skipped for the time specified by the game. But in advanced levels, this can take days or even weeks. Players use various strategies to shorten this time. They try to gain more experience by killing monsters above their levels. But in order to do this, their characters must have sufficient power and equipment. They can use power-enhancing items for this purpose. In addition, some MMORPGs have items that enhance the experience gained. Players obtain them at the end of the quests or by purchasing them from game markets. When these experience enhancing items are used, they increase the experience gained for a certain period of time. Players try to use these items effectively by hunting the maximum number of monsters during this time.

As a result, leveling is inevitable in MMORPGs and every player needs it. With the right strategy and method, it is possible to leave other players behind in a short time. Although the level gives a certain amount of power, the real source of power comes from the character's abilities and equipment.

Questing Questing is a player's ability to fulfill tasks taken from NPCs. Performing a quest consists of three stages. The first step is to take a quest from NPC. The second step is to perform the required actions. These actions include killing monsters, collecting specific items, interacting with other NPCs, going to another place in the map and so on. Once the task requirements are met, the player must deliver the task to the NPC as the final step. Once the quest has been delivered, the player will receive rewards such as experience, items or money.

Performing quests is very important in MMORPGs. Access to some maps or dungeons may depend on the completeness of certain tasks. As a result of some quests, characters can learn new skills. Players can also win very rare items from these missions. As the players perform the tasks, the tasks become increasingly difficult, so they may need help. For example, when it comes to the task of killing a very powerful boss, players who take the same task form a group by communicating and perform the task as a group. Thus, everyone can perform quests that cannot be done alone.

Farming Farming means continuously killing monsters which gives more money and valuable items. To be able to farm, the player must be very patient. First of all, he finds the monsters and bosses who give the valuable items and determines the area where he will farm. Then, he kills monsters in this area for hours or even days, and tries to find valuable items. In MMORPGs, items that will emerge when monsters die are determined probabilistically. Drop of very rare items can be even one per thousand. Therefore, the player has to kill too many monsters. To increase this possibility, some MMORPGs provide special items which are used to increase the chance of dropping items.

Trading Trading is an important feature of MMORPGs. Every game has its own economy. Players can sell items with some prices. Determining the best price is an important task because players can incur losses. Every player wants to profit, but inflation is affecting the game world's economy like the real world. The player needs to think like a trader to be able to trade well. Depending on the needs of the players, there may be fluctuations in the prices of the items. With the increase in the number of very expensive goods, the need in the market is met and the price decreases. Similarly, when an item that is affordable is consumed by players, its price increases. Trader players should follow and evaluate these fluctuations well.

B PAPI Online Implementation Details

B.1 Interfaces

B.1.1 Player Properties

Player property contains common player character features in MMORPGs. It allows creating a character with the desired feature. Every player character created in our game should have these features. Players have physical and situational properties. Physical properties such as health, defense, speed, and damage contribute to the player's battle with other players and monsters. Situational properties such as stun, dead, experience and level determine the situation the player is in during and after these battles.

```
public class PlayerProperties
{
    // indicates player's health
   public int health;

   // indicates player's speed
   public float speed;

   // indicates player's defense
   public int defense;

   // indicates player's damage
   public int damage;

   // indicates player's level
   public int level;

   // indicates player's attack range
   public int attackRange;

   // indicates player's money
   public int money;
```

```
  // indicates player's experience
  public int experience;

  // indicates player's mana
  public int mana;

  // indicates player's health potion count
  public int healthPotionCount;

  // indicates player's mana potion count
  public int manaPotionCount;

  // indicates player stunned or not
  public bool stunned;

  // indicates player dead or not
  public bool dead;

  // indicates player's position
  public Vector3 position;
 }
```

B.1.2 Player Interface

The behavior and actions of a player character are determined through the player interface which includes common player character behavior in MMORPGs. Each player character created in our game must implement this interface. The properties of the players may vary during the game. They are constantly in motion on the game world. They can use their skills to attack. They can gain experience and level up.

```
public interface IPlayer
{
  // getter for player's name
  string GetName();
```

```
// getter for player's health
int GetHealth();

// increase player's health by the given amount
void IncreaseHealth(int amount);

// decrease player's health by the given amount
void DecreaseHealth(int amount);

// getter for player's speed
float GetSpeed();

// increase player's speed by the given amount
void IncreaseSpeed(float amount);

// decrease player's speed by the given amount
void DecreaseSpeed(float amount);

// getter for player's damage
int GetDamage();

// increase player's damage by the given amount
void IncreaseDamage(int amount);

// decrease player's damage by the given amount
void DecreaseDamage(int amount);

// getter for player's defense
int GetDefense();

// increase player's defense by the given amount
void IncreaseDefense(int amount);

// increase player's defense by the given amount
```

```
void DecreaseDefense(int amount);

// getter for player's level
int GetLevel();

// increase player's level
void IncreaseLevel();

// getter for player's position
Vector3 GetPosition();

// setter for player's position
void SetPosition(Vector3 position);

// getter for player's attack range
int GetAttackRange();

// makes and attack to given target
void Attack(IPlayer target);

// getter for player's attack animation
bool IsAttacking();

// moves player to given direction
void Move(Vector3 direction);

// getter for applied buffs on the player
IList<IBuffSkill> GetAppliedBuffs();

// add a buff effect to the player
void AddAppliedBuff(IBuffSkill buff);

// remove previously added buff effect from the player
void RemoveAppliedBuff(IBuffSkill buff);
```

```
// getter for applied debuffs on the player
IList<IBuffSkill> GetAppliedDebuffs();

// add a debuff effect to the player
void AddAppliedDebuff(IBuffSkill debuff);

// remove previously added debuff effect from the player
void RemoveAppliedDebuff(IBuffSkill debuff);

// getter for player's mana
int GetMana();

// increase player's mana by given amount
void IncreaseMana(int amount);

// decrease player's mana by given amount
void DecreaseMana(int amount);

// getter for player's experience
int GetExperience();

// increase player's experience by given amount
void IncreaseExperience(int amount);

// decrease player's experience by given amount
void DecreaseExperience(int amount);

// getter for player's money
int GetMoney();

// increase player's money by given amount
void IncreaseMoney(int amount);
```

```
// getter for player’s health potion count
int GetHealthPotionCount();

// use health potion
void UseHealthPotion();

// getter for player’s mana potion count
int GetManaPotionCount();

// use mana potion
void UseManaPotion();

// getter for player’s skills
ISkill[] GetSkills();

// setter for player’s skills
void SetSkills(ISkill[] skills);

// getter for player’s skill count
int GetSkillCount();

// use the skill with given index
void UseSkill(int skillIndex, IPlayer target);

// getter for player’s skill animation
bool IsUsingSkill();

// setter for player’s stun information
void SetStunned(bool stunned);

// getter for player’s stun information
bool IsStunned();

// setter for player’s dead information
```

```
  void SetDead(bool dead);

  // getter for player's dead information
  bool IsDead();

  // creates clone of player
  IPlayer ClonePlayer();

  // resets player properties
  void ResetPlayer();

  // getter for player's available information. !this.IsStunned()
     && !this.IsUsingSkill() && !this.IsAttacking() &&
     !this.IsDead();
  bool IsAvailable();

  // updates players timers, skills and position
  void UpdatePlayer(float elapsedTime);

}
```

B.1.3 Skill Interface

Each player character has various skills. In general, we divided these skills into 3 groups. Skill kind represents these 3 groups and determines which category a skill belongs to.

```
public enum SkillKind
{
  ATTACK,
  BUFF,
  DEBUFF
}
```

All the skills in our game must implement the skill interface which includes common features found in all of the skills. Each skill has a timeout value and sets the minimum duration between consecutive uses. Skills need mana to be used. They cannot be used if there is not enough mana.

```
public interface ISkill
{
  // getter for skill's kind
  SkillKind GetSkillKind();

  // getter for skill's name
  string GetName();

  // getter for skill's mana consumption
  int GetManaConsumption();

  // getter for skill's timeout in seconds
  float GetTimeout();

  // getter for skill's availability
  bool IsAvailable();

  // use this skill from source to destination
  bool Use(IPlayer source, IPlayer target);

  // update buff
  void Update(float elapsedTime);

  // reset skill
  void ResetSkill();

  // creates clone of skill
  ISkill CloneSkill();

}
```

B.1.4 Attack Skill Interface

Features that distinguish attack skills from other skills are found in the attack skill interface. All attack skills in our game must implement this interface. The aim of the attack skills is to damage the opponent and they need a certain closeness to use it. Attack skills can have a debuff effect. This effect may occur with a certain probability with the application of the attack skill.

```
public interface IAttackSkill : ISkill
{
  // getter for attack skill's power
  int GetDamage();

  // getter for attack skill's range
  int GetRange();

  // getter for attack skill's debuff effect
  IBuffSkill GetDebuff();

  // getter for attack skill's debuff effect possibility
  int GetDebuffPercentage();

  // getter for attack skill's debuff effect existence
  bool HasDebuff();

}
```

B.1.5 Buff/Debuff Skill Interface

Buff and debuff skills can affect a variety of players' features. These properties are determined by buff kind.

```
public enum BuffKind
{
  HEALTH,
  SPEED,
  MANA,
  DAMAGE,
  STUN
}
```

Features that distinguish buff/debuff skills from other skills are found in the buff skill interface. All buff/debuff skills in our game must implement this interface. Buff skills and debuff skills work with the same logic, although they have different effects on players. Buff has a positive effect, while debuff has a negative effect. Buff/Debuff skills have a certain amount of impact during duration. This effect can be permanent, temporary or periodic. While the effect of temporary ones comes back when they are deleted, in permanent ones, this effect does not come back. Finally, periodic ones have more than one effect during duration.

```
public interface IBuffSkill : ISkill
{
  // getter for buff skill's duration in seconds
  float GetDuration();

  // getter for buff skill's amount
  float GetAmount();

  // getter for buff skill's buff kind
  BuffKind GetBuffKind();

  // getter for debuff skill'hs periodic property
  bool IsPeriodic();

  // update buff for given target
  void UpdateBuff(IPlayer target, float elapsedTime);

  // called each second until duration runs out
  void ApplyBuff(IPlayer target);

  // called at the end of duration
  void ClearBuff(IPlayer target);
  // each buff will be added to character as clone
  IBuffSkill CloneBuffSkill();

}
```

B.2 Creating Warrior Character

We have created classes that implement the interfaces in Appendix B.1 and created the PvP and skill system. By using these classes, characters with the desired abilities and features can be created to make PvP with each other. The following is an example of how to create a warrior character:

```
Player warrior = new Player(
  "Warrior",            // name
  new PlayerProperties // properties
    .PlayerPropertiesBuilder()
    .Health(500)
    .Mana(500)
    .Defense(10)
    .Damage(12)
    .Speed(1)
    .HealthPotionCount(10)
    .ManaPotionCount(10)
    .Build(),

  new ISkill[]          // skills
  {
    // attack skill with low damage
    new AttackSkill(
      "Attack1",   // name
      10,            // mana consumption
      3,             // timeout
      15,            // damage
      5             // range
    ),

    // attack skill with stun debuff effect
    new AttackSkill(
      "Attack2",  // name
      15,           // mana consumption
      4,            // timeout
      20,           // damage
      5,            // range
      new DebuffSkill(  // debuff
        "Debuff1",      // name
        0,                // mana consumption
        0,                // timeout
        BuffKind.STUN,  // buff kind
        2,                // duration
        0,                // amount
        false             // periodic
      ),
      100         // debuff percentage
    ),

    // attack skill with periodic health debuff effect
    new AttackSkill(
      "Attack3",  // name
      20,           // mana consumption
      5,            // timeout
      25,           // damage
      5,            // range
      new DebuffSkill(  // debuff
        "Debuff2",      // name
        0,                // mana consumption
        0,                // timeout
        BuffKind.HEALTH,// buff kind
        10,                // duration
        4,                 // amount
        true               // periodic
      ),
      10       // debuff percentage
    ),

    // attack skill with moderate heavy damage
    new AttackSkill(
      "Attack4",  // name
      25,           // mana consumption
      6,            // timeout
      35,           // damage
      5             // range
    ),

    // attack skill with heavy damage
    new AttackSkill(
      "Attack5",  // name
      35,         // mana consumption
      7,          // timeout
      45,         // damage
      5           // range
    ),

    // buff skill which enhances damage
    new BuffSkill(
      "Buff1",           // name
      60,                // mana consumption
      60,                // timeout
      BuffKind.DAMAGE, // buff kind
      60,                // duration
      2                  // amount
    ),
        // buff skill which enhances speed
        new BuffSkill(
          "Buff2",              // name
          70,                   // mana consumption
          60,                   // timeout
          BuffKind.SPEED,   // buff kind
          60,                   // duration
          0.5f                  // amount
        )
      }
    );
```

C Manual Reward Adjustment

We prepared test scenarios for the different behaviors of AI adjust the manual rewards which used in Section 3.2.2.

C.1 Manual Reward Test Scenarios

C.1.1 Passive Player vs Passive Player

In this test scenario, both players remain passive and cannot beat each other. Since they do not use their skills too much, they receive few positive reward. So that, the cumulative reward they collect is negative.

	Player 1		**Player 2**	
Action	**Count**	**Reward**	**Count**	**Reward**
Skill1	2	0,0293	1	0,0147
Skill2	1	0,0159	1	0,0159
Skill3	0	0,0000	1	0,0171
Skill4	0	0,0000	0	0,0000
Skill5	2	0,0440	0	0,0000
Skill6	0	0,0000	1	0,0850
Skill7	1	0,0850	1	0,0850
Attack	0	0,0000	0	0,0000
Debuff	1	0,0175	2	0,0350
Health Pot	1	0,0150	0	0,0000
Mana Pot	1	0,0050	1	0,0050
SubTotal	9	0,2117	8	0,2577
Win	0	0,0000	0	0,0000
Lose	0	0,0000	0	0,0000
Each Step	50000	-1,0000	50000	-1,0000
Total	50009	-0,7883	50008	-0,7423

C.1.2 Active Player vs Passive Player

In this test scenario, one player stays passive while the other actively attacks his opponent. While passive receives very few positive rewards, active collects too many positive rewards. At the end of the battle, the active one gains +1 reward for victory and the passive one gains -1 reward for the defeat. When the cumulative reward is considered, passive one gets big negative value whereas active one gets big positive value.

	Player 1		**Player 2**	
Action	**Count**	**Reward**	**Count**	**Reward**
Skill1	7	0,1027	1	0,0147
Skill2	6	0,0953	1	0,0159
Skill3	5	0,0856	1	0,0171
Skill4	4	0,0782	0	0,0000
Skill5	4	0,0880	0	0,0000
Skill6	1	0,0850	1	0,0850
Skill7	1	0,0850	1	0,0850
Attack	90	0,1440	0	0,0000
Debuff	6	0,1050	2	0,0350
Health Pot	10	0,1500	0	0,0000
Mana Pot	8	0,0400	1	0,0050
SubTotal	142	1,0588	8	0,2577
Win	1	1,0000	0	0,0000
Lose	0	0,0000	1	-1,0000
Each Step	25000	-0,5000	25000	-0,5000
Total	25143	1,5588	25009	-1,2423

C.1.3 Active Player vs Active Player

In this test scenario, both players actively attack each other. The player who applied more debuff effects, manages to beat the other. Since they both actively use their skills, they receive a large number of positive rewards. At the end of the war, the winner receives +1 and the loser -1 reward. The cumulative reward received by the winner takes a great positive value, while the loser gets a medium negative value. The loser does not take big negative value because he struggles well.

	Player 1		**Player 2**	
Action	**Count**	**Reward**	**Count**	**Reward**
Skill1	8	0,1173	7	0,1027
Skill2	7	0,1112	6	0,0953
Skill3	6	0,1027	5	0,0856
Skill4	5	0,0978	4	0,0782
Skill5	4	0,0880	4	0,0880
Skill6	1	0,0850	1	0,0850
Skill7	1	0,0850	1	0,0850
Attack	100	0,1600	90	0,1440
Debuff	8	0,1400	6	0,1050
Health Pot	10	0,1500	10	0,1500
Mana Pot	8	0,0400	8	0,0400
SubTotal	158	1,1770	142	1,0588
Win	1	1,0000	0	0,0000
Lose	0	0,0000	1	-1,0000
Each Step	35000	-0,7000	35000	-0,7000
Total	35159	1,4770	35143	-0,6412