Universiteit Leiden
The Netherlands

Bachelor AI

Adapting Model-Free Reinforcement Learning for Boss Fights in Hollow Knight: a reward shaping approach

Jennifer Lee

Supervisors:
Joost Broekens

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

06/03/2023

Abstract

This paper researches the effect of reward shaping in a complex and dynamic environment. A challenge in reinforcement learning is having sparse rewards. As the agent gets a reward only when completing an objective. This can make it very difficult for the agent to associate actions with the reward. Reward shaping helps the agent by giving it more frequent feedback. This can help the agent in the learning phase and help it to find desirable behaviors. These agents are implemented in a Hollow Knight boss fight as a good test case. This is a good test case as the boss fights in Hollow Knight are complex and have dynamic environments, where each boss fight incorporates stochasticity in their attack patterns. To understand the effect of certain types of rewards, the rewards are divided into base, sub, and instrumental rewards. Where base reward is a win, sub-rewards are rewards leading to a win, and instrumental rewards are rewards that do not necessarily lead toward a win but can lead to desirable behaviors. All combinations of these categories will be implemented as agents and trained. Our results show that reward shaping can help find more desirable behaviors in the learning phase however, it does not change our end phase result. When not including the base reward the models performed worse in the end phase. When including instrumental rewards, only with all the types of rewards it performed well. When combined with the sub-reward it had no effect on the wins or losses, and on its own or combined with the base reward it showed convergence to a policy that did not focus on winning the boss fight. When trying to test the effect of transfer learning on another boss, it showed that the agents needed more training time to converge to an optimal policy and further conclusions could not be made. Reward shaping can have a beneficial influence on the training phase when dealing with a complex and dynamic environment. Nonetheless, it is important to acknowledge that reward shaping has the potential to introduce biases, potentially diverting the agent from its optimal policy.

Contents

1 Introduction

Reinforcement learning has emerged as a powerful framework for teaching agents to make decisions in a complex and dynamic environment. Reinforcement learning is a type of machine learning that involves training an agent based on its interactions with its environment, by trial and error [1]. An agent in reinforcement learning learns from feedback in the form of rewards and penalties for its actions.

There are two main types of reinforcement learning algorithms, model-based and model-free reinforcement learning [2]. Model-free reinforcement learning is a type of reinforcement learning in which an agent learns to make decisions through trial and error, without explicit knowledge of the dynamics or rules of the environment. The agent interacts with the environment and uses feedback to update its policy or value function. Model-free reinforcement learning does not require prior knowledge about its environment to make predictions or learn the optimal value. While model-based reinforcement learning has a model of the environment.

In this paper, we explore the effectiveness of reward shaping in a model-free reinforcement learning agent. Reward shaping is the process of modifying the reward signal in a reinforcement learning problem to improve the performance of the learning agent [3]. A problem usually has a natural reward. For most games, this reward is based on winning or losing. The basic idea of reward shaping is to add additional rewards to the natural reward in order to guide the agent towards desirable behaviors and away from undesirable ones. This involves modifying the reward function of the agent to shape its learning process. Reward shaping in reinforcement learning is a technique that takes inspiration from the principles of animal training [4]. In animal training, trainers often provide additional rewards or incentives to animals in order to enhance their learning and improve their performance. These supplemental rewards can be used to guide the animal towards desired behaviors and help them understand the desired outcomes more easily. The same goes for a reinforcement learning agent. Reward shaping provides more frequent feedback on the agent on its behavior. This can help an agent in large environments to find promising behavior in the early learning phase. While reward shaping can lead to learning faster performance, it can also lead to sub-optimal behavior. This is due to biases introduced by reward shaping. The agent may optimize for the shaped reward and can overlook the base natural reward of the problem.

The reinforcement learning agents are trained in the game Hollow Knight. Some common benefits of implementing a reinforcement learning agent in games are standardized environments, well-defined goals, and rich interactive experiences[5]. Hollow Knight offers boss fight environments where the environment is complex and dynamic. It is a good environment for a study of reward-shaping, as without reward-shaping the agent will receive sparse rewards. Which can make it difficult for the agent to learn an optimal policy. The agent requires precise timing, pattern recognition, and strategic decision-making. The introduction of new attack patterns or the alteration of boss behavior adds an additional layer of complexity. Implementing a reinforcement learning agent in these boss fights allows an agent to learn and adapt in these complex and dynamic environments. By training a reinforcement learning agent to tackle Hollow Knight boss fights, there exists potential for enhancing skills and adaptability. The agent can continually refine its strategies, potentially surpassing human-level performance by discovering optimal patterns, exploiting boss weaknesses, and making precise and effective decisions.

1.1 Motivation and related work

Researching reinforcement learning in games offers numerous benefits and opportunities in the field of artificial intelligence as in game development. In the paper "Reinforcement Learning in Video Games" they discuss the application of reinforcement learning techniques in video games[5]. The authors present their work on using deep reinforcement learning to play Atari2600. An Atari 2600 is an old-school video game console, a few classic games on this console were Breakout, space invaders, pong, and Mario. A few benefits of using games as a research platform for reinforcement learning are standardized environments, well-defined goals, and rich interactive experiences. Researching reinforcement learning in games offers benefits such as benchmarking and comparison, simulating real-world complexity, rich interaction and learning opportunities, generalization and transfer learning exploration, engagement and collaboration between fields, and improved user experience.

Reinforcement learning faces several challenges. A game environment provides sparse rewards, meaning that meaningful feedback is only received upon achieving certain objectives or reaching specific milestones[6]. Sparse rewards make it difficult for the agent to learn an optimal policy, as they struggle to associate actions with their long-term consequences. This leads to slower learning and exploration struggles. With the sparse rewards, exploration becomes challenging as agents rely solely on the sparse feedback provided by the environment. With a sparse reward, the Curse of dimensionality becomes more pronounced, as it becomes more difficult for the agent to sample informative experiences from the state-action space.

Reward shaping can help the agent in speeding up the learning process by providing more frequent and informative feedback to the agent[4]. Sparse or delayed rewards make it difficult for the agent to learn optimal behaviors. This can close the gap between action and reward, allowing the agents to get more immediate feedback and accelerate the learning progress. Adding reward shaping can also help guide the agent into more desirable behavior and away from undesirable behaviors.

To explore this issue, we turn our attention to the game Hollow Knight, a popular platformer released in 2017. Hollow Knight offers a variety of boss fights. These boss-fight environments are complex and dynamic. Boss fights have a well-defined goal, which is winning the boss fight. When setting this as our natural reward, without reward shaping the rewards are sparse. Which could lead to slow convergence to the optimal policy. This makes Hollow Knight great for a reward-shaping study, as we can see the effect of the performance of an agent in a complex and dynamic environment with sparse rewards. By shaping rewards in innovative ways, we can explore how an agent's decision-making changes in response to a more complex gaming environment.

For this research paper, we searched for suitable implementation examples. Two repositories had already tried to implement reinforcement learning in the game Hollow Knight. These repositories feature the development of a reinforcement learning agent that could recognize the boss HP bar, the agent's HP, and could control the agent's actions. Zhantao Yan repository [7] is inspired and built further on the repository of Ruifeng Cui [8]. The difference between the two repositories is that Ruifeng Cui extensively employed CE and Windows API, whereas Zhantao Yan explored the extensions to the Deep Q-Network algorithm. In this study, we will mainly use the repository of Zhantao Yan and make some alterations to study the effect of reward shaping.

1.2 Thesis overview

This chapter contains the introduction, motivation, and related work; Section 2 includes the research questions and the hypothesis; Section 3 describes the approach, setup, and experiment that are used to train our reinforcement learning agent in the game Hollow Knight; Section 4 shows the results of the performance of our agents; Section 5 concludes our research.

2 Research question

This paper aims to address the following research question:

What is the effect of reward shaping on a model-free reinforcement learning agent?

Several sub-research questions are formulated to delve into this research question:

- How does the reinforcement learning agent perform without reward shaping?
  Without reward shaping, learning in scenarios with a large state space and delayed rewards can be slow and inefficient. Reward-shaping techniques are necessary to provide more frequent feedback, helping the agent learn faster and more effectively.
- What kind of rewards can we define for the reward shaping function?
  There are different kinds of rewards we can add to the reward function. Can we categorize the rewards to understand how those added rewards influence the agent's performance?
- What kind of performance metric could we use to measure reward shaping?
  When comparing agents that all have a different reward function, we can not measure it by the accumulated reward over time. What is a good way to measure and compare those agents?
- What are the effects on the learning phase and end phase of reward shaping?
  Reward-shaping can have a different effect on the learning phase and the end phase
- Does reward shaping help transfer learning? To see the effect of reward shaping on transfer learning. When learning a policy on one boss can it help to defeat another boss?

2.1 Hypotheses

To understand the effect of reward shaping for a model-free reinforcement learning agent, different models are trained on a boss fight in Hollow Knight. These different models have different reward functions. The win rate was employed as a performance metric for the reward-shaped agents. This metric was selected due to its effectiveness in measuring the agent's ability to achieve the clearly defined objective of defeating the boss in boss-fight scenarios. The training phase and the end phase of these models will be compared. To compare these models against each other a null hypothesis and an alternate hypothesis are defined.

2.1.1 Training phase

H_0: There is no significant difference in the win rates between the different models in the training phase.

H_1: There is a significant difference in the win rates between the different models, indicating that the added reward has an effect on the model in the training phase.

2.1.2 End phase

H_0: There is no significant difference in the win rates between the different models in the end phase.

H_1: There is a significant difference in the win rates between the different models, indicating that the added reward has an effect on the model in the end phase.

3 Method

We implement our agent in the game Hollow Knight to test reward shaping for multiple boss fights. Hollow Knight is a 2D hand-drawn platformer game that came out in 2017. The game is about a small agile knight, which the player or in our case the agent can control, that is thrown into the deep as the knight has no memories about the place he is in. The game takes place in a town called Dirtmouth, where you will explore a vast ruined kingdom of insects and heroes. The game features a large underground world, where the player can explore different areas, fight bosses, and unlock new abilities. When completing the game you can reach the hall of gods. This area is filled with statues of passed boss fights. These can be activated and selected to fight them. What makes Hollow Knight a suitable game for a study in reward shaping lies in its complex and dynamic environment. In the absence of reward shaping, the rewards in this complex and dynamic environment are sparse. In such a complex and dynamic environment, it becomes challenging for the agent to establish a clear association between its actions and the corresponding rewards. Reward shaping can help the agent to overcome this obstacle.

3.1 Materials

The programming language that is used to implement the reinforcement learning agent is Python. To train this agent on the game "Hollow Knight", a working version of the game is required. To add an HP bar for the boss fights and to read this the Satchel and EnemyHPBar Mod needs to be installed and run correctly. This is run on a computer system with Windows.

3.2 Experimental setup/approach

In this experiment, specific settings are configured in the game Hollow Knight to implement our agent.

To measure the win rate, an experimental setup is made where the agent autonomously plays Hollow Knight against a boss. Each episode is recorded and measured the reward and the win. These bosses whom the agent is trained against are either "Marmu" or "The traitor lord". The boss fight "Marmu" is used to test the performance of each of the different reward models and to see the effect the reward has. On the boss fight "The traitor lord" the effect of reward shaping will be tested. This will be done by taking the latest model of either the "base" model or the model with all the rewards and training this further on this boss. To test the effect of transfer learning a model that has not yet seen either boss fights will also be trained.

During each episode, the agent's actions are guided by a reinforcement learning algorithm, and reward shaping were applied to facilitate learning. These reward-shaping rewards aim to provide additional guidance for the agent to achieve desirable behavior.

In order for the agent to perceive the game environment, including its own HP and the boss's HP, additional configurations are implemented

3.2.1 Environment

The boss fights in Hollow Knight are modeled as a factored environment. The environment is factored as the state is represented as a matrix of variables and there is a relation/overlap between the states. The environment is a 192x192 black-and-white matrix. The boss fights that are used are fairly simple as the environment does not include falling platforms, spikes where the agent can not walk as it will take damage, or any other environmental surprises. Figure 1 shows the environment of the boss fight Marmu. This boss fight is used to test our agent with the different reward functions. We will use this boss fight to see the effect of reward shaping. The state includes the agent which is the white knight which stands in the middle of the figure. The health of the agent which can be seen in the left corner, each head of the knight stands for 1 health. The boss Marmu can be seen on the top right. When hitting the boss the state will also include a health bar for the boss.

Figure 1: Environment boss fight Marmu.

The boss Marmu has only one attack and one ability. It attacks by curling into a ball and hurling itself at the knight from various angles. The ability it uses is a teleport, this is used in combination with its attack and will teleport itself to a random location in the environment. The boss fights in Hollow Knight incorporate stochasticity through randomized attack patterns. This will make it more challenging for the agent. Because the boss Marmu is fairly simple as the agent only needs to take into account one attack of the boss, transfer learning is tested on the boss fight the traitor lord. The environment of the boss the traitor lord can be seen in figure 2.

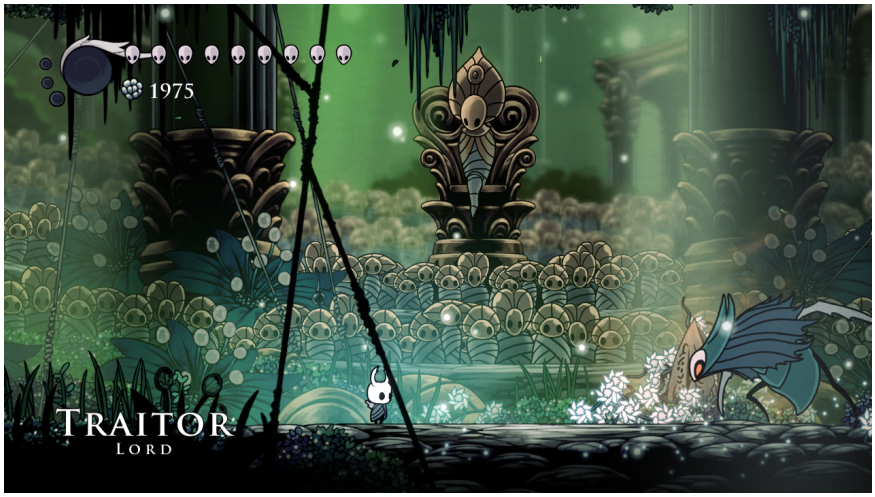

Figure 2: Environment boss fight Traitor lord.

The state space is almost the same only the background and the boss are a little bit different. This boss fight however is much more difficult, as it attacks faster, does more damage, and has more attack moves than the boss Marmu. The traitor lord has four attacks. The dive attack let the boss leap through the air and dive towards the knight. The dash attack let him dash toward the knight and will swipe with his claws. The dancing glaive attack let the traitor lord throw two spinning wind-scythes in tandem. Which will travel across the environment. And finally, the ground pound attack where it will smash its claws feverishly into the ground, which will generate a massive shockwave in both directions. These shockwaves will span from the floor to the ceiling of the environment.

The state is captured by taking screenshots of the game screen. The information about the knights and boss HP is extracted from the screenshot. The hp of the knight and the boss are a part of the state space. When the state is captured the observation is preprocessed. The observation is still 192x192 matrix. This observation will be scaled, so the values are normalized in the range of minus two and zero.

3.2.2 Action space

The action space in the context of reinforcement learning refers to the set of possible actions that the agent can take in an environment. In this study, the action space for the reinforcement learning

agent is defined to perform a range of actions that are relevant to the gameplay mechanics and objectives in the game. The game features the option for charms. These charms provide various abilities, statistical enhancements, or unique features that can influence the knight's gameplay style. The charms that are equipped are mostly statistical enhancement. One charm let the dash action do damage, which can increase the use of this action.

The specific action space of our experiment consists of a discrete set of actions, which include movement actions, combat actions, and displacement actions. One action is a combination of the three categories. This way the agent can perform an attack in the air to the right for example. The movement category exists out of no movement, moving left, and moving right. The combat category exists out of no attack, normal attack, spell, and upp spell. When doing a normal attack the knight slashes with his nail in front of him, the spell shoots a spirit in front of him that will fly forward and do damage to the enemies in its way, and the upp spell does shoot a cone of spirits above itself doing damage to every enemy above itself. The displacement category exists out of no displacement, short jump, long jump, and dash. The dash action in this case will let the knight displace himself for a certain distance in a short timeframe. It dashes through and damages enemies in its way.

The action space exists out of an array of movement, combat, and displacement. Each combination of the three categories is an action, which gives us an action space of $3 * 4 * 4 = 48$ actions. An action with the number one exists for example out of no movement, no combat, no displacement.

3.2.3 Reward

The reward function plays a crucial role in guiding the agent's learning process by providing feedback on its actions. In this study, we designed different reward functions to differentiate the models and understand the effect it has on the training and end phase of our models. The reward is categorized into either base, sub, or instrumental rewards. In the game Hollow Knight, the natural reward for fighting against a boss is winning. The natural reward or winning the boss fight we will call the base reward. The sub-rewards are the rewards leading to the base rewards. An example of a sub-reward could be the reduction of the boss's health points (HP) during the boss fight. And finally, instrumental rewards are rewards that will not necessarily lead to the base reward but can help with guiding the agent to more desirable behavior, for example, certain actions.

With those reward functions, we aimed to shape the agent's behavior by providing additional rewards based on desirable states or actions in the game and see what the effect is of adding those rewards. The base reward we defined is a high reward that the agent receives when winning the boss fight. This reward is set at a thousand to give our agent a high reward ceiling, as the reward-shaping rewards should never reach a higher value than this reward[9]. For the sub-rewards, the agent will receive a reward for hitting the boss. This reward will be a value between one and 500, this value depends on the percentage the boss is hit. The total that the agent can get by defeating the boss, is 500. Another sub-rewards is getting hurt by the boss. When getting hit by the boss the agent gets a negative reward of minus 50. In total, the agent has nine lives, which means the agent can get when losing the boss fight to get a total of minus 450 as a punishment. The punishment is set lower than the reward of the sub-rewards as we want to have a positive reward when defeating the boss even if it survives with only 1 life. Then for the instrumental reward, we added a small reward of one for actions that can hurt the boss. These actions are attacking, using a spell, using the up

spell, or using the dash ability. The dash action in our setup is also a damaging ability as it can hurt the boss fight when dashing through it. This is a bias as we think doing damage will get the agent to converge faster to the optimal policy and defeat the boss faster. The different categories of rewards with their rewards can be viewed in table 1.

Base reward		Sub rewards		Instrumental rewards	
Win	1000	Hurt	-50	Attack	1
		Hit	500*(percentage health lost)	Spell	1
				Up spell	1
				Dash	1

Table 1: The different rewards the agent receives when doing a certain behavior for each reward category.

The reward function will be a combination of all the different reward categories. There will be seven different agents with each a different reward function. These reward functions differ by having different combinations of reward categories. For example, there is an agent with only the base reward or an agent with the base and the sub rewards. These reward functions will shape the agent's behavior and facilitate learning in their own manner. This study will research the effect the different reward categories has on our model in the learning and end phase.

3.2.4 Model

For our reinforcement learning agent, we use Deep Q-learning. The "deep" in deep Q-learning represents the neural network. Our model exists out of two key components: the representation of the environment and the Q-value prediction.

The first component is responsible for extracting features from the input observations. It takes an observation, which is a screen of the environment, as input and extracts features through a series of convolutional layers with relu activation layers in between. See figure 3 for the architecture of the model.

The second component takes extracted features from the first component as input and performs further computations to estimate the Q-values for each action in the environment. This represents a dueling network architecture. The "Dueling Network Architectures for Deep Reinforcement Learning" introduces the dueling network architecture[10]. In traditional Q-learning algorithms, the action-value function is used to estimate the value of each action in a given state. The dueling network architecture takes a different approach by decomposing the action-value function into two separate components, namely the value function and the advantage function. The value function estimates the value of being in a particular state regardless of the action taken, while the advantage function measures the importance of each action in relative to the others. Eventually, the outputs of the value and advantage stream are combined to obtain the Q-value for each action in the state. Figure 4 shows the dueling network architecture.

The "Dueling Network Architectures for Deep Reinforcement Learning" paper highlights the limitations of Q-learning algorithms, where the action-value function is used to estimate the value

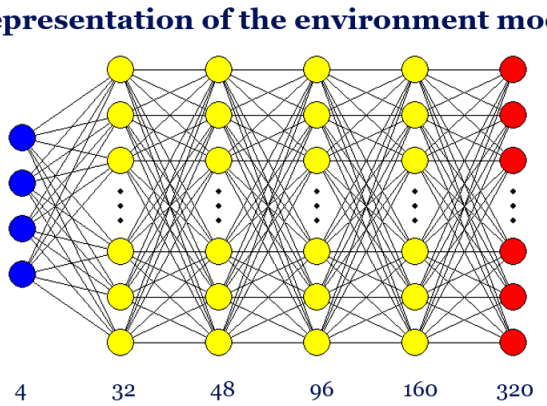

Figure 3: Convolutional network for the representation of the environment

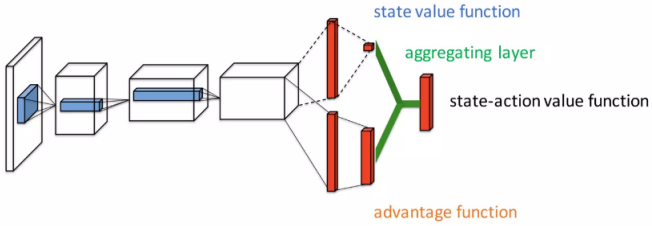

Figure 4: Dueling network architecture [11]

of each action in a state[10]. One of the limitations mentioned in the paper is the difficulty in distinguishing between the value of each action, as traditional Q-learning algorithms estimate the value of each action in a state. This is however in many situations challenging to accurately assess the value of individual actions. Another limitation mentioned is the high variance in action values, as it can exhibit high variance in the estimates of action values. Which can lead to a slower convergence rate and slower learning. And finally, inefficiency in learning value-based policies as they require the estimation of action value for each state-action pair. The authors propose the dueling network architecture, which addresses the issues by separating the estimation of the value and advantage functions. This could result in a more accurate and stable estimation of action value, which leads to improves learning efficiency and performance.

3.2.5 Reinforcement learning implementation

This paper makes use of reinforcement learning. The implementation of the agent runs it for one run with 1000 episodes. For every step in an episode, the agent will get an action either random or the action with the highest Q-value. In every tenth episode, the agent will perform all random actions, in every other run it will perform the action with the highest Q-value that the model gives back. It then performs the action in the environment, and it receives back the next observation, the reward it gets from performing that action, and if the boss fight is done. The reward is then added to the total reward of the episode. The next observation, the reward, and the flag that shows if the

boss fight is done are added to the replay buffer. For every four actions the model will learn, this is used to update the model and get the loss.

The agent learns by using a multi-step replay buffer, a main model, and a target model. The multi-step replay buffer is a component in reinforcement learning algorithms that enables the agent to capture long-term dependencies and make informed decisions[12]. Normally, replay buffers store individual experiences in a memory buffer and randomly sample them for learning. However, in the scenario of boss fights where the consequences of an action may not be immediately visible, capturing long-term effects becomes important. It extends the standard replay buffer by incorporating the concept of N-step returns. Instead of considering the immediate reward obtained for an action, the N-step returns the accumulated rewards over multiple steps in the future, with a future discount. This allows the agent to estimate the cumulative impact of an action by considering the consequences over a longer time.

Similar to the approach described in the paper "Playing Atari with Deep Reinforcement Learning," the use of a main model and a target model is being used [5]. The main model is trained to approximate the Q-value, which represents the expected future rewards for each action given a state. The main model is the primary decision-making component of the agent and makes choices based on the learned Q-values. The target model is a separate copy of the main model and plays a crucial role in stabilizing the learning process. The main model can be sensitive, during training, to the correlation between consecutive samples, which can lead to fluctuations and instability in the learning process. The target model helps mitigate this issue by providing stable target values. The target model is updated less frequently compared to the main model, this allows it to become more stable and less sensitive to fluctuations observed during training. The weights in the main model are periodically copied to the target model, this is done to align their representations. This ensures a more consistent learning process. This stability improves the learning performance and the ability to generalize in new situations. The target model provides a fixed reference for updating the model, which enables it to converge toward the optimal Q-value more reliably. Which will lead to improved performance and better adaption to the given task.

During the learning process, a random sample is selected from the multi-step replay buffer, which is a collection of past experiences. This exists out of the agent's observation, action, reward, next observation, and a flag indication if the episode is done. Then it calculates the target Q-values with the following equation.

$$Target = R + \gamma * max(Q(s',a'))$$

Where;

- Target represents the target value for updating the Q-value of the state-action pair
- R is the immediate reward received by the agent after taking action a in state s
- γ is the discount factor, this determines the importance of future rewards. This is a value between 0 and 1
- max(Q(s',a') represents maximum Q-value over all possible actions in the next state

This expresses the relationship between the current Q-value and the maximum expected future Q-value. This equation takes the immediate rewards, the discount factor which determines the importance of future rewards, and the maximum Q-value that is estimated by the target model. It then approximates Q-values from the main model. The loss is then calculated between the estimated Q-values from the main model and the target model by the Mean Squared Error(MSE). The MSE loss measures how well the main model approximates the optimal Q-value. It then updates the main model using the calculated loss. It optimized the model's parameters by backpropagation, this aims to minimize the difference between the estimated Q-values and the target Q-values.

The learning step is iteratively repeated using multiple samples from the replay buffer. This will allow the agent to learn and refine its decision-making policy based on past experiences and the interaction within the environment.

3.3 Measures

There are several ways to measure the performance of a reinforcement learning agent. The most common practice to measure the performance of a reinforcement learning agent is by measuring the speed of convergence to optimality [1]. One way to do this is by using the accumulated reward the agent receives. However, our reward-shaping approach would not work with this method. With our reward-shaping approach, the agent will get extra or less reward than the agent without reward-shaping would receive.

The performance of the agent is evaluated based on its win rate over time. Each agent is evaluated for one run over 1000 episodes. This can be seen as a fairly low training time for a reinforcement learning agent in an environment this large. Due to limited time availability, it is not possible to conduct a greater number of runs or episodes for the agent within the given timeframe. The agent is only allowed to run once, which means it cannot be evaluated based on the average number of wins per episode across multiple runs. This is why the win rate is defined as an exponential weighted moving average of the wins over the episodes. An exponentially weighted moving average performs smoothing by applying weights of the data points, with more recent points receiving higher weights [13]. The win rate provides a quantitive measure of the agent's ability to successfully complete the objective, by defeating the boss, and reflects on its overall performance.

Finally, to measure the different models against each other we used the chi-square statistic to test our hypothesis for the different models. The hypothesis measures if there is a difference between the models in wins and losses. This is performed on the training and the end phase of the models. The training phase will be defined as the episodes where each model has still not yet converged. The end phase will be defined as the episodes where each model has reached convergence. The first 300 episodes will be called the training phase and The last 300 episodes will be called the end phase. These phases were identified based on our tests, which showed their existence between the episodes. This way we will see the different performances of our models in the training phase and the end phase. With hypothesis testing, we can evaluate the significance of the observed changes in the win rate over time to identify certain trends or patterns in the agent learning and end phase.

4 Results

The agents will be tested against two boss fights: Marmu and The Traitor Lord. In the boss fight Marmu we compare the agents with different reward functions. This will be compared in the training and end phase of the agents. In the boss fight The Traitor Lord we will investigate the transfer learning capabilities of a reinforcement learning agent that includes only the base reward and one with the base, sub, and instrumental rewards.

4.1 Reward shaping

The agents with the different reward functions are run for 1000 episodes on the boss fight Marmu. The win rate over the episodes per agent can be seen in figure 5.

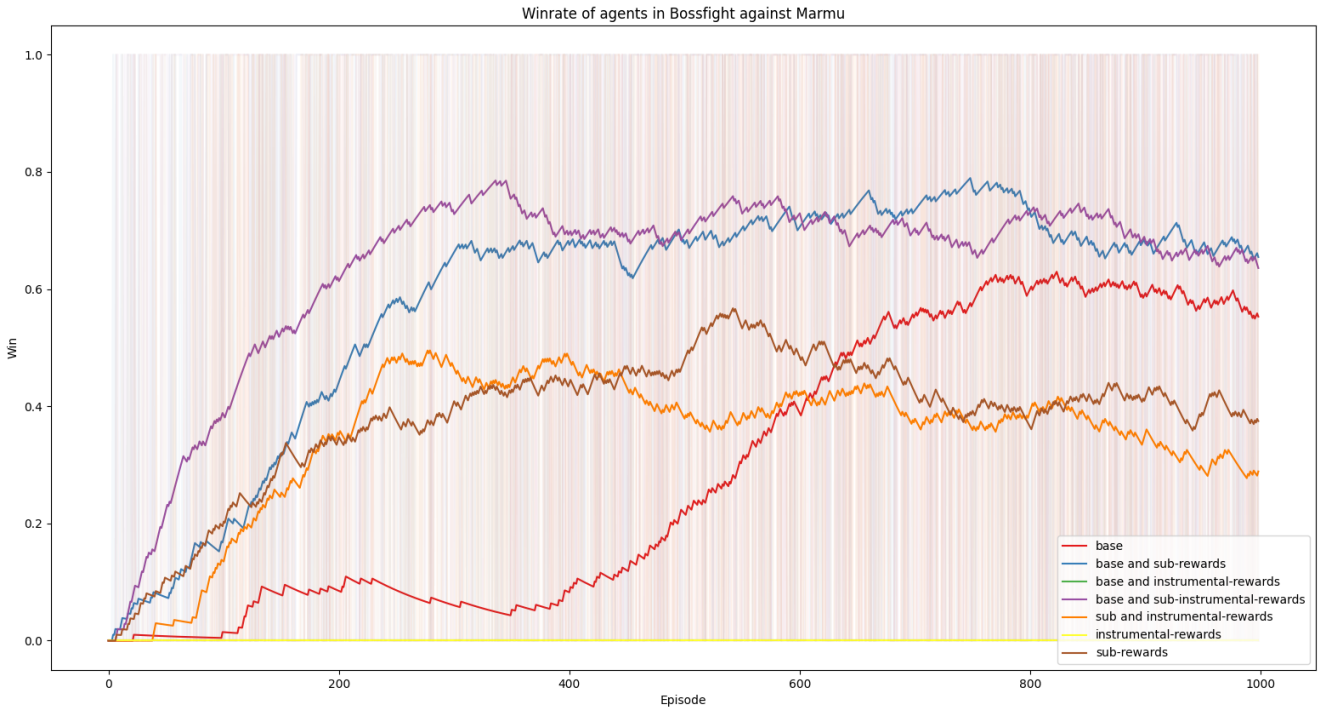

Figure 5: Winrates of agents in the boss fight against Marmu over 1000 episodes

The analysis of the agent's performance reveals some interesting patterns and insights. It shows that the agent that is solely trained with instrumental rewards or a combination of base and instrumental rewards shows zero wins over 1000 episodes. The agent which is trained with instrumental rewards does not include the natural reward which is winning the boss fight. Instead of winning the boss, the agent tries to survive as long as possible to collect as much reward as possible. It tries to perform as many possible actions that are giving him a reward. The agent that is trained with the

instrumental and base reward, has the base reward of winning the boss fight. But the instrumental rewards are prioritized as it does not see the base reward in any run. It converges into a policy that optimizes instrumental rewards. These agents are stuck on a local optimum.

The agent with only the base rewards shows a slower learning phase but achieves quite a high win rate in the end phase. It seems like it outperforms the agents with only sub-rewards and a combination of sub and instrumental rewards in the end phase. And it is just under in performance with the agents that have a combination of all the rewards and with the base and sub reward. This is logical, as with only the base reward it should learn the slowest but when converged should show a higher win rate than the agents which had not included the base reward. This is cause the agent with the base reward included tries to optimize the goal of winning the boss fight. When letting this reward out of the equation can lead to optimizing a different goal.

Figure 5 shows that there is little to no difference in the training and end phase of the agent with sub-rewards and the agent with the combination of sub and instrumental rewards. When looking at the agents that include instrumental rewards, they almost always perform worse with instrumental rewards than without. The only exception is when combining all the rewards. Only there it shows a faster learning phase and a higher win rate in the end phase. This can be due to the rewards reinforcing each other in combination with all the rewards. As the instrumental rewards encourage the agent to damage the boss, which gives us a positive sub-reward, which eventually leads to winning the boss fight and thus winning.

The agent with all the rewards seems to be performing the best as it shows the highest win rates in the early training phase and in the end phase. While the agent with the combination of base and sub-rewards converges around the same win rate in the end phase, it shows a lower win rate in the training phase. The only difference between those models is the added instrumental rewards.

4.1.1 Statistical test

To statistically test if there is a significant difference between the models during the training and end phase, a chi-squared statistical test is performed. A contingency table is constructed, which captures the wins and losses for the various reward models. Table 2 shows the contingency table for the training phase and table 3 shows the contingency table for the end phase.

When following our hypothesis the null hypothesis states that there are no significant differences between the wins or losses between the different models. We can reject the null hypothesis if the p-value falls below the significant level. A significant level of 0.05 is chosen to determine the acceptance or rejection of the null hypothesis. When the p-value falls below 0.05, the null hypothesis is rejected and indicated that there is a significant difference between the wins and losses.

The chi-square statistic and the corresponding p-values for the training and end phases are represented in table 4. The p-values for both the training phase and the end phase were found to be remarkably low, this leads to the rejection of the null hypothesis for both the training and end phase. This provides strong evidence in the existence of a significant difference in wins or losses among the different models.

Model	Wins	Losses
Base	22	278
Sub	115	185
Instrumental	0	300
Base & Sub	162	138
Base & Instrumental	0	300
Sub & Instrumental	120	180
Base, Sub & Instrumental	215	85

Table 2: Contingency table with the wins and losses for the different reward models in the first 300 episodes.

Model	Wins	Losses
Base	177	123
Sub	115	185
Instrumental	0	300
Base & Sub	205	95
Base & Instrumental	0	300
Sub & Instrumental	98	202
Base, Sub & Instrumental	201	99

Table 3: Contingency table with the wins and losses for the different reward models in the last 300 episodes.

	Chi-square statistic	P-value
Training phase (episode 1-300)	682.547	< 0.001
End phase (episode 700-1000)	652.410	< 0.001

Table 4: Chi-square value and p-value of the training and end phase of all the reward models

To understand which specific model combinations show significant differences and those that do not, a statistical test was performed for each combination. This will be performed for the training phase and end phase of the models. Table 5 shows these statistics for the training phase. The values represented in red represent the p-values that exceed the threshold of 0.05. This indicates that we can not reject the null hypothesis. For such a combination of models, no significant differences between the wins and losses were observed.

	Base	Sub	Instrumental	Base & Sub	Base & Instrumental	Sub & Instrumental	Base, Sub & Instrumental
Base	-	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
Sub	< 0.001	-	< 0.001	< 0.001	< 0.001	0.738	< 0.001
Instrumental	< 0.001	< 0.001	-	> 0.001	1.000	< 0.001	< 0.001
Base & Sub	< 0.001	< 0.001	< 0.001	-	< 0.001	< 0.001	< 0.001
Base & Instrumental	< 0.001	< 0.001	1.000	< 0.001	-	< 0.001	< 0.001
Sub & Instrumental	< 0.001	0.738	< 0.001	< 0.001	< 0.001	-	< 0.001
Base, Sub & Instrumental	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	-

Table 5: P-values for the chi-square statistic for each combination of the different reward models in the first 300 episodes.

Table 5 shows that almost all the different combinations of models have a p-value under our significant level. This means we can reject the null hypothesis for those combinations of models, and say that there is a significant difference between the wins and losses. There are two combinations of models that can not reject the null hypothesis. One of the combinations of models that can not reject the null hypothesis is the Sub and Sub & Instrumental models. In figure 5 we can already see that they are fairly similar. But apparently, statistically, they have the same amount of wins and losses in the learning phase. Another combination of models that can not reject the null hypothesis is the Instrumental and Base & instrumental models. This gives back a p-value of one. This means that the wins and losses are exactly the same. This is logical, as both the models have found zero wins and 300 losses.

	Base	Sub	Instrumental	Base & Sub	Base & Instrumental	Sub & Instrumental	Base, Sub & Instrumental
Base	-	< 0.001	< 0.001	0.022	< 0.001	< 0.001	0.052
Sub	< 0.001	-	< 0.001	< 0.001	< 0.001	0.173	< 0.001
Instrumental	< 0.001	< 0.001	-	< 0.001	1.000	< 0.001	< 0.001
Base & Sub	0.022	< 0.001	< 0.001	-	< 0.001	< 0.001	0.794
Base & Instrumental	< 0.001	< 0.001	1.000	< 0.001	-	< 0.001	< 0.001
Sub & Instrumental	< 0.001	0.173	< 0.001	< 0.001	< 0.001	-	< 0.001
Base, Sub & Instrumental	0.052	< 0.001	< 0.001	0.794	< 0.001	< 0.001	-

Table 6: P-values for the chi-square statistic for each combination of the different reward models in the last 300 episodes.

The p-values of the chi-squared test for each combination of the models for the end phase are shown in table 6. Here it shows four combinations of models that can not reject the null hypothesis. These results show some interesting insight into the performance of our models. In figure 5 it seemed like the model with the base reward has converged to a lower win rate than the models with the base, sub, & instrumental reward and the model with the base & sub reward. But the p-value of these models is above our significant level of 0.05. This means we can not reject the null hypotheses and can say that the wins and losses between those models are not significant. We can say that the base reward model has the same amount of wins and losses as the models with the base, sub, & instrumental rewards and Base & Sub rewards.

The combination of models that could not reject the null hypothesis in the training phase, also could not reject the null hypothesis in the end phase. It is interesting to note that we can say that with the combination of the Sub and Sub & Instrumental model, the instrumental rewards did not influence the wins and losses at all.

4.2 Transfer learning experiment

The Traitor Lord boss fight is fairly more difficult than the boss fight against Marmu. To test the effect of transfer learning we will compare the win rate of four different models. These models are the base model, the reward shaping model, the transfer base model, and the transfer reward shaping model. The reward shaping model is the model with the base, sub, and instrumental rewards. With the transfer models, it means that these models take the latest model with the same reward function from the boss fights against Marmu.

We trained our agent on this boss fight for one run for 1000 episodes. The win rate over the episodes can be seen in figure 6.

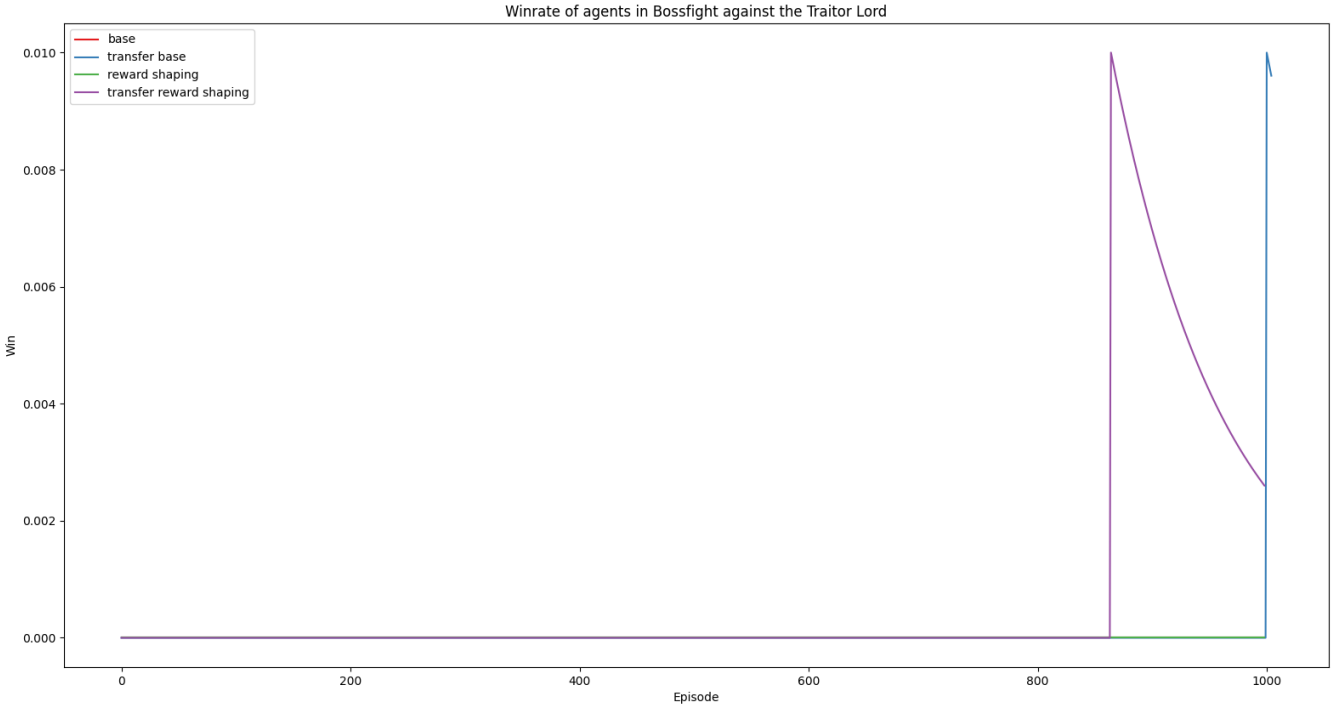

Figure 6: Winrates of agents in the boss fight against the Traitor Lord over 1000 episodes

Both the base reward model and the reward shaping have zero wins in 1000 episodes. It shows that with the transfer learning both the transfer base model and the transfer reward shaping model have shown one win in 1000 episodes. The transfer reward shaping model has found the win earlier than the transfer base model.

As there is not much to see in the win rate over the episodes, the rewards over the episodes are plotted in figure 7. This can show us a little bit more insight into the performance of transfer learning in a reward-shaping agent.

Figure 7 Shows that the transfer reward shaping model learns a little bit faster, but the rewards shaping model quickly is around the same win rate. It shows however that the transfer reward shaping is still not converged and needs a longer training time to find an optimal policy for the Traitor lord boss fight.

4.3 Discussion

The main result of our paper is that the models with base, sub & instrumental rewards, and sub & instrumental rewards both had a better training phase as they learned and converged earlier than

Figure 7: Reward of agents in the boss fight against the Traitor Lord over 1000 episodes

the model with only the base reward. However, in the end phase, the models all had no significant differences in wins and losses. This confirms that reward shaping can help the agent to learn and find desirable behaviors in the early learning phase of a complex and large problem. While it can help to learn faster and find desirable behaviors earlier, it can also steer the agent away from the natural reward. This can be seen in a few models with instrumental rewards, as it steered the agent away from the natural reward and came to a policy optimizing the instrumental rewards.

While reward shaping can lead to faster and better performance in the training phase, it can also lead to suboptimal behavior. The reward-shaping reward always has a bias in them, to shape the reward and to hundred percent be sure it has no negative effect on the policy you have to assume having all the knowledge about the problem. This could be seen when combining the reward without the sub-rewards, as we saw that the instrumental or the base & instrumental rewards both converged to a policy that did not focus on winning the boss fight. In papers where they train a reinforcement learning agent to play AlphaGO, Chess, or Shogi, they intentionally chose not to incorporate reward shaping [14][15]. They chose not to incorporate reward shaping as they had a desire for simplicity, generality, independence from expert knowledge, and autonomous learning and exploration. One way to incorporate reward shaping without the chance of negatively affecting the optimal policy is by potential-based reward shaping. The paper "Shaping and policy search in reinforcement learning," discusses the concept of potential-based reward shaping[9]. Potential-based reward shaping is a technique used to provide additional rewards to a reinforcement learning agent based on potential functions. Potential-based reward shaping involves defining a potential function that captures the desirability or progress towards a goal state. This potential function assigns

values to states in the environment, reflecting their proximity to the desired goal. By incorporating the potential function into the reward signal, additional rewards can be provided to guide the agent toward the goal state. Potential-based rewards provide a principled and effective approach to shaping the reward signal in reinforcement learning. By incorporating the notion of potential and proximity to the goal state, these rewards guide the agent's exploration and learning process while minimizing biases and aligning with the ultimate goal of reaching the desired state. An example of potential-based rewards is our sub-rewards or even other factors such as avoiding boss attacks.

4.4 Limitations and errors

One of the limitations is time. One run with 1000 episodes is not much training for such a complex problem. When having more time you would preferably have the agent run more runs and train the agent over more episodes. When conducting more runs you could average the win rate over those runs per episode.
In the early phase of the research, an agent with only the base reward was trained for one run with 550 episodes. In this run, the agent has found a win in episode 511. As this is almost at the end of the episodes that we ran, the agent had almost no time to train the agent. In figure 5 the base model found its first win on episode 22. The base model has a chance of luck when finding its first reward, as it performs purely random actions as does not find a reward until winning. When only performing one run for an agent it can not take into account the chance of luck.

In the boss fight again the Traitor Lord it also showed that it needed more training time. In the boss fight Marmu all the models had converged as it was a fairly easy boss fight. This however was not the case for the boss fight the Traitor Lord as the agents needed more time to converge to see the effect of transfer learning.

When looking at the errors made in this research there are two to be addressed. To explore the agent performs every tenth episode only random actions. What normally is done with reinforcement learning is the use of exploration/exploitation methods like ϵ-greedy. When only performing random actions every tenth episode it is highly likely that in the tenth episode, it would not find a win, as it is performing a series of random actions. While it seems to be learning with the help of the multi-step replay buffer and the training of an easier boss fight, it could converge easier to a local optimum by using this technique. When doing further research the setting of an epsilon is recommended.

Another error is in the target Q-value update equation that is used to calculate the target values. The formula should have been:

$$Target = R_t + \gamma^n * max(Q(s_{t+n}, a_{t+n}))$$

Where;

- Target represents the target value for updating the Q-value of the state-action pair where t represents the current time step
- R_t is the immediate reward received by the agent after taking action a_t in state s_t

- γ is the discount factor, this determines the importance of future rewards. This is a value between 0 and 1
- n is the number of steps considered for the update
- $max(Q(s_{t+n}, a_{t+n})$ represents the maximum Q-value over all possible actions in the state reached after n steps from the current state-action pair

As it takes the reward of the multi-step replay buffer, this reward is the accumulated future reward over n-steps. As this is the accumulated reward over n steps the future discount should also take this into account.

5 Conclusions and Further Research

We can conclude that reward shaping can have a positive effect on the training phase of a model. However wrong reward shaping can also lead the model away from the optimal policy and can get the model stuck in a local optimum. This research examines the categorization of rewards into three distinct types: Base, Sub, and Instrumental rewards. The Base reward represents the natural reward that the agent strives to achieve. Sub-rewards are intermediary rewards that serve as stepping stones toward obtaining the Base reward. On the other hand, Instrumental rewards are additional rewards that aid the agent in reaching the Base reward more efficiently, although they may not directly contribute to obtaining the Base reward itself. This study investigates and highlights the distinctions between these reward types. The base reward in large environments can take a while to find a reward. Even when the base reward finds a reward it has a hard time understanding which kind of actions eventually lead to the reward. As the reward is only shown after doing approximately 136 actions. When only using the sub-rewards it has been shown to learn faster, as it sees a reward not only when defeating the boss fight. But when not rewarding the agent when defeating the boss fight, this has an impact on the end phase of the model. As it has a lower win rate than with only the base model. The agent tries to maximize the sub-rewards and will not see the goal of winning the boss fight. When only using the instrumental reward it shows no ability on winning the boss fight. This is due to the fact that the model learns a policy where it tries to survive as long as possible and performs as many actions that give it a reward. Combining the instrumental rewards with other rewards in all cases performed worse than without it. Except for the model with all the rewards. This model worked because the instrumental, sub, and base rewards were amplifying each other. As the instrumental rewards are making the agent use more actions that will damage the boss fight, damaging the boss fight is a sub-reward, which in turn will lead to defeating the boss fight. When looking at the effect of transfer learning in another boss fight it shows one win in both models that had the latest model of the other boss fight. However, the agent had to learn a policy for this boss fight that was too difficult to learn in one run for 1000 episodes. To better understand the effect of transfer learning on a reward-shaped model and a base model further research is necessary. Either longer training time or choosing a less difficult boss fight where a model could learn a policy in 1000 episodes. Additionally, it would be interesting to train the agent on multiple boss fights and evaluate its performance in a boss fight that it has not encountered during training. This can assess the agent's ability to generalize its learned skills and strategies across different boss fights. By examining its performance in unfamiliar scenarios, we can gain insights into the agent's adaptability and its capacity to apply previously acquired knowledge to new challenges.

References

[1] Andrew W. Moore Leslie Pack Kaelbling, Michael L. Littman. Reinforcement learning : A survey. 1996.

[2] Ben Dickson. A gentle introduction to model-free and model-based reinforcement learning.

[3] Tim Miller. Reward shaping.

[4] Eric Wiewiora. *Reward Shaping*. Springer US, Boston, MA, 2010.

[5] David Silver Alex Graves Ioannis Antonoglou Daan Wierstra Volodymyr Mnih, Koray Kavukcuoglu and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013.

[6] Tianpei Yang Hongyao Tang Chenjia Bai Jinyi Liu Zhaopeng Meng Peng Liu Jianye Hao Member, IEEE and Zhen Wang. Exploration in deep reinforcement learning: From single-agent to multi-agent domain. 2013.

[7] Zhantao Yang. Dqn_hollowknight, 2021.

[8] Ruifeng Cui. Hollowknight_rl, 2022.

[9] Andrew Y. Ng. Shaping and policy search in reinforcement learning. 2003.

[10] Matteo Hessel Hado van Hasselt Marc Lanctot Nando de Freitas Ziyu Wang, Tom Schaul. Dueling network architectures for deep reinforcement learning. 2016.

[11] Taehoon Kim. Dueling network architectures for deep reinforcement learning.

[12] Richard S.Sutton and Andrew G. Barto. Reinforcement learning: An introduction. 2018.

[13] J. Stuart Hunter. The exponentially weighted moving average. 1986.

[14] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529, 01 2016.

[15] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017.

A Game settings

The game screen needs to be set to a resolution of 1280x720 pixels. The code has coordinated checks to identify the knight's health bar and the boss's health bar. Additionally, two mods need to be installed to recognize the boss's health bar, namely Satchel and EnemyHPBar mod.

Keybindings, which can be seen in figure 8, are adjusted to the keybindings in our code to let the agent perform actions.

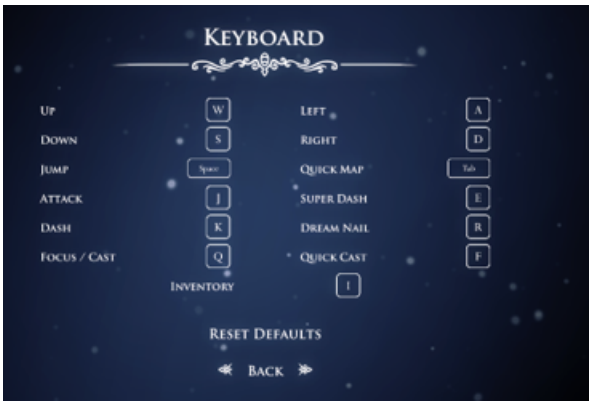

Figure 8: Settings for the keybindings.

The game Hollow Knight features the option for the player to equip charms onto the knight. These charms provide various abilities, statistical enhancements, or unique features that influence the knight's gameplay style. In this experiment the following charms are equipped: Sharp Shadow, Mark of Pride, Quick Slash, and Unbreakable Strengts these are illustrated and further explained in figure 9 and 10. While it is interesting to experiment with those charms and see what different kinds of tactics will use to defeat the boss, for this experiment these charms are equipped as a baseline for the agent.

Figure 9: The Sharp Shadow charm

The Sharp Shadow charm empowers the knight's dash move to inflict damage. Normally, the dash move is primarily used for evasion and traversing through enemies, however, with this charm, it becomes an offensive maneuver. The dash move is significant and most often used in boss fights, when equipping this charm we hope to make the ability even more rewardable for the agent to use it.

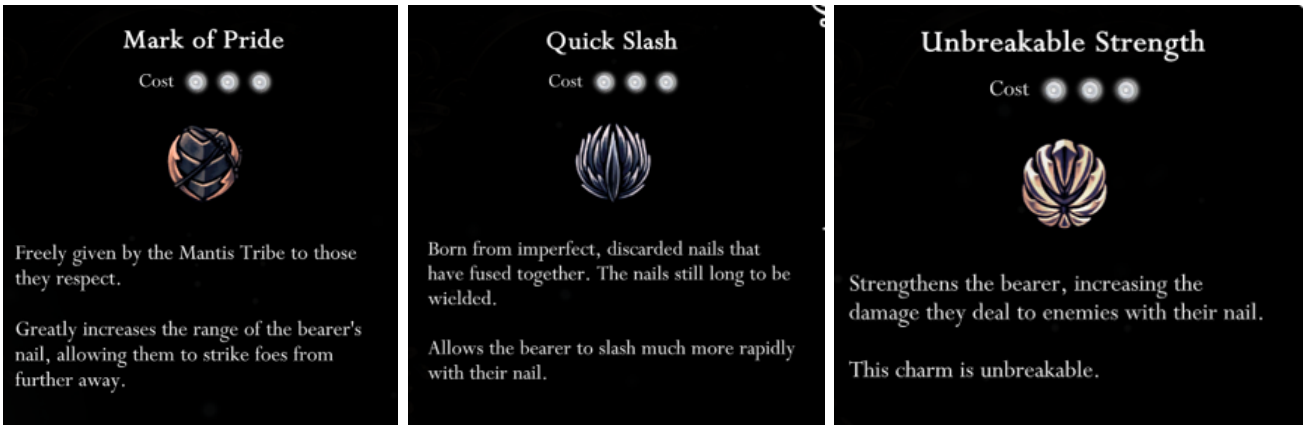

Figure 10: The Mark of the Pride, Quick Slash, and Unbreakable Strength charm

The charms depicted in figure 10 are charms that enhance the knight's strength, attack range, and attack speed.

B Model definition

In this appendix, we present the code for two classes, SimpleExtractor and DuelingMLP, which are part of the implementation.

The SimpleExtractor class is a subclass of AbstractExtractor and serves as a feature extractor for the model. It takes as input the observation shape, number of input channels, activation function (defaulting to ReLU), and an optional spectral normalization flag. The class initializes a series of convolutional layers with specified kernel sizes, strides, and padding. The output is obtained by flattening the last layer and returning the result. It has four input channels and the output layer exist out of 320 channels.

The DuelingMLP class is a subclass of AbstractFullyConnected and represents a dueling network architecture. It takes an instance of AbstractExtractor, the number of output units, activation function (defaulting to ReLU), and optional flags for noise and spectral normalization. The class initializes two sets of fully connected layers for the value and advantage streams. It also includes the necessary forward method for passing the input through the network, applying the appropriate activation functions, and combining the value and advantage streams. It returns the Q-values for all 42 actions.

The detailed implementation of these classes can be seen in the source code below.

```
class AbstractExtractor(nn.Module):
    def __init__(self, obs_shape: tuple, in_channels: int,
                 activation='relu', sn=False):
        super(AbstractExtractor, self).__init__()
        self.activation_name = activation

    def forward(self, x):
        raise NotImplementedError

class SimpleExtractor(AbstractExtractor):
    def __init__(self, obs_shape: tuple, in_channels: int, activation='relu', sn=False):
        super(SimpleExtractor, self).__init__(obs_shape, in_channels, activation, sn)
        if self.activation_name == 'relu':
            act = nn.ReLU(inplace=True)
        elif self.activation_name == 'leaky_relu':
            act = nn.LeakyReLU(inplace=True)
        else:
            raise NotImplementedError(activation)
        out_shape = np.array(obs_shape, dtype=int)
        out_shape //= 32
        final = nn.Conv2d(160, 320, kernel_size=3, stride=2, padding=1)
        if sn:
            final = spectral_norm(final)
        self.convs = nn.Sequential(
            nn.Conv2d(in_channels, 32, kernel_size=3, stride=2, padding=1),
            act,
            nn.Conv2d(32, 48, kernel_size=3, stride=2, padding=1),
            act,
            nn.Conv2d(48, 96, kernel_size=3, stride=2, padding=1),
            act,
            nn.Conv2d(96, 160, kernel_size=3, stride=2, padding=1),
            act,
            final,
            act,
            nn.Flatten(),
        )
        self.units = 320 * np.prod(out_shape)

        for m in self.modules():
            param_init(m)

    def forward(self, x):
        x = self.convs(x)
        return x
```

```
class DuelingMLP(AbstractFullyConnected):
    def __init__(self, extractor: AbstractExtractor, n_out: int,
                 activation='relu', noisy=False, sn=False):
        super(DuelingMLP, self).__init__(extractor, n_out, activation, noisy, sn)
        self.linear_val = self.linear_cls(extractor.units, 512)
        self.linear_adv = self.linear_cls(extractor.units, 512)
        self.val = self.linear_cls(512, 1)
        self.adv = self.linear_cls(512, n_out)

        if sn:
            self.linear_val = spectral_norm(self.linear_val)
            self.linear_adv = spectral_norm(self.linear_adv)

        if noisy:
            self.noisy.append(self.linear_val)
            self.noisy.append(self.linear_adv)
            self.noisy.append(self.val)
            self.noisy.append(self.adv)

        self.resetable = nn.ModuleList([
            self.linear_val,
            self.linear_adv,
            self.val,
            self.adv
        ])

        self.reset_params()

    def forward(self, x, adv_only=False, **kwargs):
        x = self.extractor(x)
        x = torch.flatten(x, 1)
        adv = self.linear_adv(x)
        adv = self.act(adv)
        adv = self.adv(adv)
        if adv_only:
            return adv
        val = self.linear_val(x)
        val = self.act(val)
        val = self.val(val)
        x = val + adv - adv.mean(dim=1, keepdim=True)
        return x
```

C Hyperparameter values

Hyperparameter values replay buffer:

size	= 180000
n	= 10
gamma	= 0.99
prioritized	= None

Hyperparameter values environment:

obs_shape	= (192,192)
rgb	= False
gap	= 0.99

Hyperparameter values dqn:

env	= env
replay_buffer	= replay_buffer
n_frames	= 4
gamma	= 0.99
epsilon	= 0
target_steps	= 8000
lr	= $8e^{-5}$
lr_decay	= False
criterion	= torch.nn.MSELoss()
batch_size	= 32
device	= 'cuda'
is_double	= True
drq	= True
svea	= True
reset	= 0
n_targets	= 1
save_suffix	= 'HornetV2'
no_save	= False

D Use of chatGPT

In this paper, chatGPT is been used as a writing aid and for research purposes. In the case of research purposes, chatGPT is used to understand and gain more insight into certain parts of the code. In which techniques were used and how they worked. As a writing aid chatGPT is used to rephrase certain self-written sentences in a more scientific way. The generated text from chatGPT is then again rephrased to still incorporate my own writing style.

E True p-values chi-square tests

	Base	Sub	Instrumental	Base & Sub	Base & Instrumental	Sub & Instrumental	Base, Sub & Instrumental
Base	-	4.643305e-19	8.009614e-06	1.175988e-34	8.009614e-06	1.549127e-20	1.175907e-57
Sub	4.643305e-19	-	4.932331e-32	1.694826e-04	4.932331e-32	7.384288e-01	5.051140e-16
Instrumental	8.009614e-06	4.932331e-32	-	2.384255e-49	1.000000e+00	1.022730e-33	6.899536e-74
Base & Sub	1.175988e-34	1.694826e-04	2.384255e-49	-	2.384255e-49	8.137869e-04	1.161899e-05
Base & Instrumental	8.009614e-06	4.932331e-32	1.000000e+00	2.384255e-49	-	1.022730e-33	6.899536e-74
Sub & Instrumental	1.549127e-20	7.384288e-01	1.022730e-33	8.137869e-04	1.022730e-33	-	1.213246e-14
Base, Sub & Instrumental	1.175907e-57	5.051140e-16	6.899536e-74	1.161899e-05	6.899536e-74	1.213246e-14	-

Table 7: True P-values for the chi-square statistic for each combination of the different reward models in the first 300 episodes.

	Base	Sub	Instrumental	Base & Sub	Base & Instrumental	Sub & Instrumental	Base, Sub & Instrumental
Base	-	6.554946e-07	1.125232e-55	2.217296e-02	1.125232e-55	1.769381e-10	5.224282e-02
Sub	6.554946e-07	-	4.932331e-32	3.573480e-13	4.932331e-32	1.730807e-01	3.963066e-12
Instrumental	1.125232e-55	4.932331e-32	-	9.305291e-69	1.000000e+00	1.481041e-26	8.895995e-67
Base & Sub	2.217296e-02	3.573480e-13	9.305291e-69	-	9.305291e-69	5.576990e-18	7.938771e-01
Base & Instrumental	1.125232e-55	4.932331e-32	1.000000e+00	9.305291e-69	-	1.481041e-26	8.895995e-67
Sub & Instrumental	1.769381e-10	1.730807e-01	1.481041e-26	5.576990e-18	1.481041e-26	-	9.216696e-17
Base, Sub & Instrumental	5.224282e-02	3.963066e-12	8.895995e-67	7.938771e-01	8.895995e-67	9.216696e-17	-

Table 8: True P-values for the chi-square statistic for each combination of the different reward models in the last 300 episodes.

F Reward plots boss fight Marmu

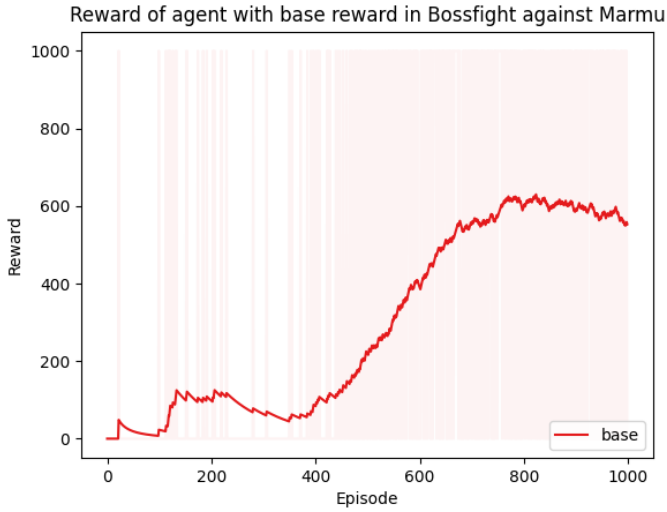

Figure 11: Reward over 1000 episodes of agent with the base reward in the boss fight against Marmu

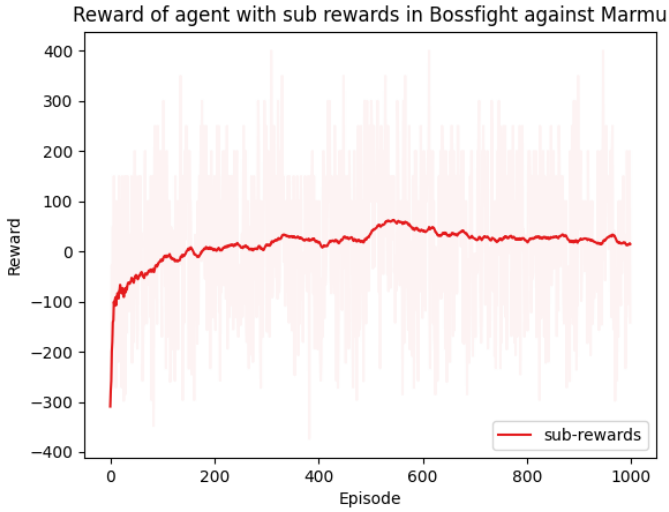

Figure 12: Reward over 1000 episodes of agent with the sub-reward in the boss fight against Marmu

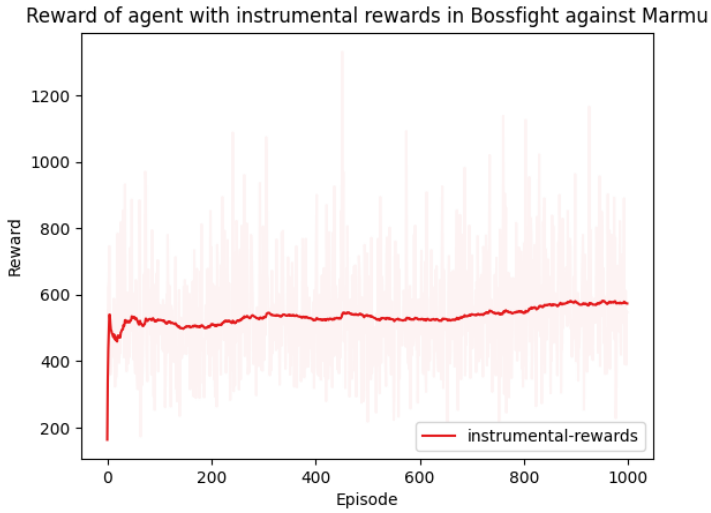

Figure 13: Reward over 1000 episodes of agent with the instrumental reward in the boss fight against Marmu

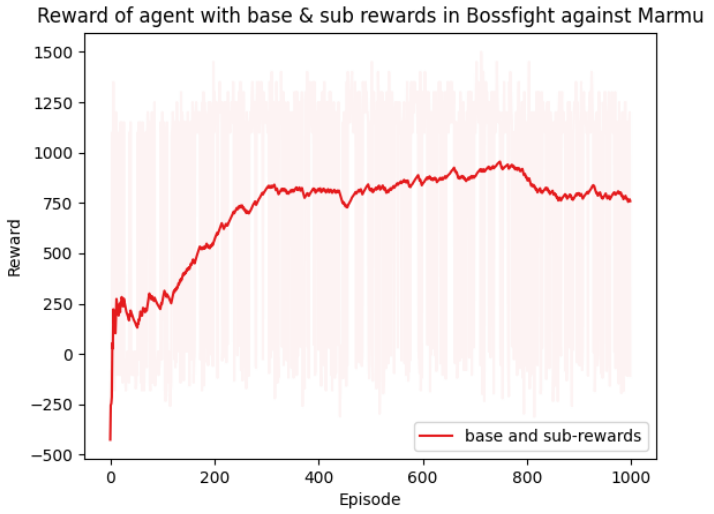

Figure 14: Reward over 1000 episodes of agent with the base & sub rewards in the boss fight against Marmu

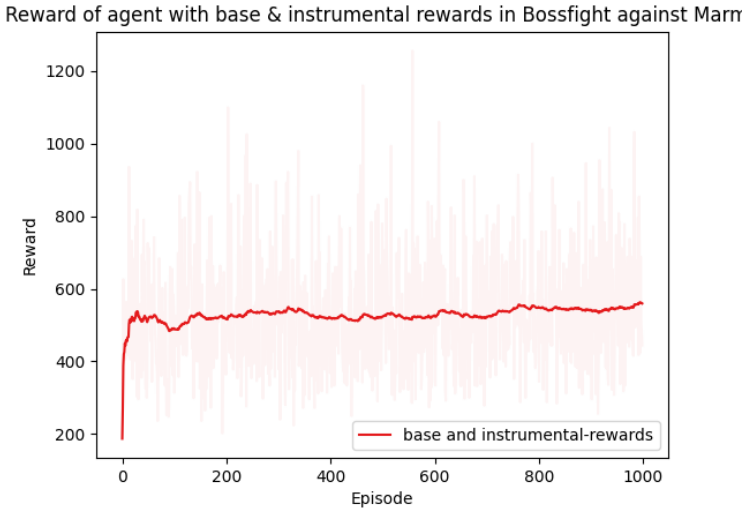

Figure 15: Reward over 1000 episodes of agent with the base & instrumental rewards in the boss fight against Marmu

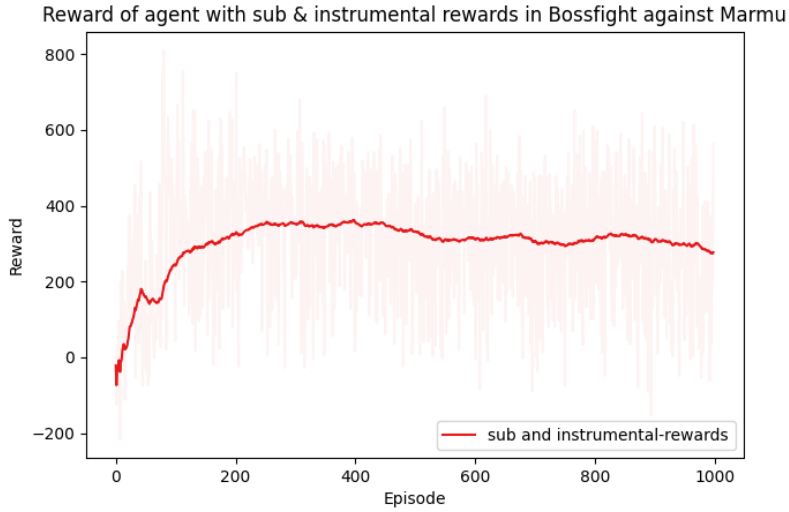

Figure 16: Reward over 1000 episodes of agent with the sub & instrumental rewards in the boss fight against Marmu

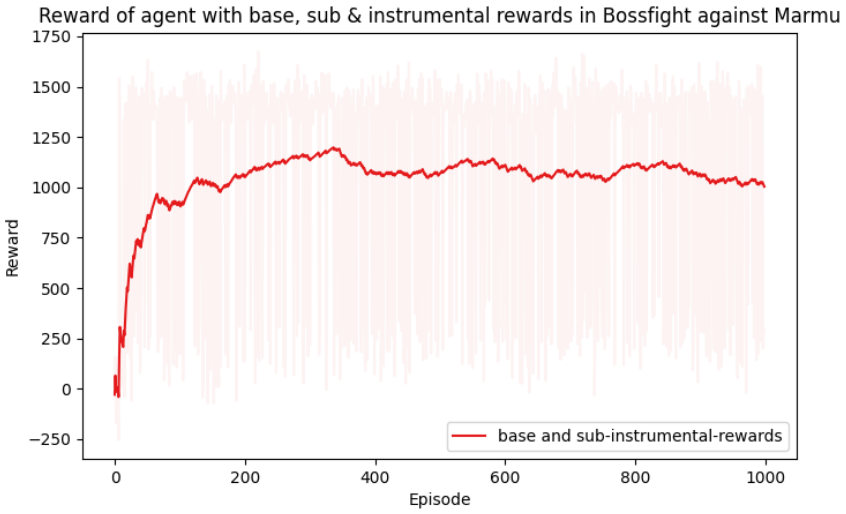

Figure 17: Reward over 1000 episodes of agent with the base, sub & instrumental rewards in the boss fight against Marmu