REINFORCEMENT LEARNING IN ELDEN RING: HOW PROXIMAL POLICY OPTIMIZATION DEALS IN COMPLEX ENVIRONMENTS

By

Vladimir Angelov

Submitted to Central European University – Private University
Undergraduate Studies

In partial fulfilment of the requirement for the Bachelor of Science in Data Science and Society

Supervisor: Imre Fekete

Vienna, Austria
2025

Copyright Notice

Author's Declaration

I, the undersigned, **Vladimir Angelov**, candidate for the Bachelor's degree in Data Science and Society, declare herewith that the present thesis is exclusively my own work, based on my research and only such external information as properly credited in notes and bibliography. I declare that no unidentified and illegitimate use was made of the work of others, and no part of the thesis infringes on any person's or institution's copyright. I also declare that no part of the thesis has been submitted in this form to any other institution of higher education for an academic degree.

Vienna, 26 May 2025

Vladimir Angelov

Abstract

This thesis explores the challenges and capabilities of modern reinforcement learning algorithms when applied to complex, noisy, and dynamic environments. By training Proximal Policy Optimization (PPO) agents within the action RPG video game *Elden Ring*, this work investigates how well reinforcement learning can handle noisy input, high-stakes decisions, and unstructured real-time feedback. The results highlight both the promise of current algorithms and the significant engineering obstacles that remain, particularly in perception and reward design. Ultimately, this research aims to explore how to implement modern reinforcement methods in complex environments and what importance that has on real-world implementations.

Acknowledgments

I would like to thank my supervisor, Imre Fekete, for helping me through the process of writing my thesis, the creators of EldenRL for making their codebase available and enabling this work to build on a solid foundation, my friend, Georgi Ivanov, for lending me his Boosteroid account where I could run my experiment and last but not least, I'd like to thank Micky van de Ven for his goal-line clearance in the Europa League final (2025), which indirectly allowed me to focus on my thesis in the final days before the deadline.

Contents

Chapter 1 Introduction

Learning is a key part of human development from the very first moments of life. A child begins to explore the environment by reaching out, grasping objects and gradually starts to understand which actions yield pleasurable or useful outcomes and which actions do not. Actions that succeed are repeated, while those that fail are abandoned. After many trials and errors the child develops an understanding of how it is supposed to interact with the world and what outcomes it can expect.

This method of learning, designed by nature, has been implemented in one of the central concepts in artificial intelligence: Reinforcement Learning (RL). In RL, an artificial agent learns to make decisions by interacting with a simulated or a real-world environment by receiving feedback in the form of rewards or penalties. The only difference is that in reinforcement learning, someone else has to set the reward function for the algorithm. The agent's objective is to learn an optimal policy in order to maximize the rewards over time, in other words, the agent creates for itself rules of what action will maximize the reward according to the current state that the agent is in. As described by Sutton and Barto in *Reinforcement Learning: An Introduction* (2018), this learning process is modeled mathematically using the Markov Decision Process (MDP) framework, which sums up state transitions, action choices, reward functions, and discounting of future reward returns. Similar to how a child learns to navigate around the real world, the RL agent learns to navigate its own environment through experience.

Reinforcement learning has demonstrated remarkable success in structured environments and has achieved superhuman performance in dealing with constrained environments such as board games or arcade simulations, where the rules are well-defined and the outcomes are deterministic. Reinforcement learning agents like *AlphaGo*, developed by DeepMind, have already made history by mastering the complex strategic board game, *Go*. *AlphaGo* managed to achieve superhuman performance by grabbing three out of three victories against the world champion of *Go* in 2017, highlighting how reinforcement learning agents could surpass human skill in certain tasks and games. (BBC, 2017)

These achievements catalyzed a wave of research and experimentation. Reinforcement learning algorithms were applied to more complex environments, from robotic and autonomous vehicles to resource optimization, healthcare systems and interactive media. The goal behind these advancements is to create agents that are able to rival humans in making decisions in unpredictable, real-world situations and tasks requiring skill and precision.

The purpose of this thesis is to explore the current abilities of reinforcement learning when dealing with complex environments and analyze what might be its limitations. By applying RL algorithms to an unstructured environment to an environment as *Elden Ring*, this research tests how well these algorithms can learn, adapt and perform in conditions that mirror real-world uncertainty and sensory overload. It also investigates how design choices, such as shaping the rewards function, observation processing and environment feedback, impact the agent's learning outcomes. This thesis aims to provide insight into both the practical challenges of deploying RL in dynamic settings and the technological advancements that are pushing the field forward.

I chose to use *Elden Ring* as an environment because it presents challenges more similar to real-world systems, where the agent has to deal with imperfect information, fast-paced decision making, noise and randomness of the environment. By designing an RL pipeline that utilizes screen capture for observations and simulates keystrokes for actions, this project mimics human interaction with the game. The agent receives input as images, processes them through a convolutional policy and outputs discrete commands like dodging or attacking. The command choices have been limited in order to make the game simpler for the agent during training.

By replicating the kind of noisy, high-pressure decision-making found in real-world systems, this thesis explores both the capabilities and the limitations of modern reinforcement learning agents. *Elden Ring* may be just a video game, but it offers a complex and dynamic environment that closely mirrors the unpredictability and visual richness of real-world settings, making it a fitting testbed for evaluating the evolution of RL techniques. By pushing the limits of what is achievable in a game-based context, this work

highlights both the technical challenges and the transformative potential of reinforcement learning in the wild.

The thesis starts with a literature review, describing the evolution Reinforcement Learning went through to become what it is today and discusses the implications of different reinforcement learning models. Next, it provides a detailed description of the environment I have used for my experiment. Then it describes the methodology that was used to complete this work. This is followed by a results section, analyzing how the agents learned and behaved in the environment. Then, the limitations and future developments of the project are discussed to establish what might have gone wrong and propose solutions for the issues. Finally, the thesis ends with a short Social Aspect section and a Conclusion section.

Chapter 2 Literature Review

2.1 Introduction to Reinforcement Learning

Reinforcement Learning (RL) is an unsupervised machine learning approach where agents learn to make optimal decisions by interacting with an environment. By choosing different actions, the agents receive rewards or penalties and based on the results, they better their strategies in order to maximize the returns. In Reinforcement Learning: An Introduction, by Sutton and Barto (2018), describe reinforcement learning as a computational approach in which agents learn by trial and error and are guided by the outcomes of their actions instead of being explicitly taught correct behaviors.

At the very heart of reinforcement learning is a framework called Markov Decision Process (MDP), which helps describe how an agent makes decisions over time. An MDP is essentially a tuple ⟨S, A, P, R, ⟩, where "S" represents all the possible situations or states the agent can be in, "A" is all the actions the agent can take, "P" describes how a certain action will change the state the agent is in, "R" are the predicted rewards that the agent can get by the actions and " (gamma)" is a constant representing how much the agent cares about future rewards. This structure allows RL agents to deal with uncertainty and plan over longer periods of time (Zheng, Reinforcement Learning and Video Games, 2019).

The key idea in RL is maximization of the expected return. This means the agent learns a strategy, also called a policy, which tells it which actions are better, given the situation it is in. This approach helps to create algorithms that can handle complex and unpredictable environments.

2.2 From Classical Methods to Deep RL

The early RL algorithms like Q-Learning (Watkins and Dayan, 1992) and SARSA (Rummery and Niranjan, 1994) have shown how table-valued functions can be used in order to predict the long-term return and therefore create more optimal policies. However, these

methods were very effective in discrete, low-dimensional environments, but they struggled with more complex tasks, due to their inability to filter out noise and therefore deal with high-dimensional sensory inputs, and therefore weren't good enough for my experiment.

A major breakthrough in reinforcement learning occurred when Deep Q-Networks (DQN) were introduced by Volodymyr Mnih and others in *Human-level Control Through Deep Reinforcement Learning*, published in 2015. This work introduced a convolutional neural network (CNN), which is used for analyzing visual data, allowing agents to learn directly from image inputs in Atari 2600 games. The combination of deep learning and RL, also known as Deep Reinforcement Learning (DRL) marked the beginning of a new era where complex decision-making tasks were solvable with the help of powerful function approximators like CNNs.

Further advancements underlined some of the limitations of Deep Q-Networks. For example in *Deep Reinforcement Learning with Double Q-Learning*, Van Hasselt et al. (2016) introduced the double DQN, which reduced the overestimation bias in Q-learning, which stabilized the learning of the agents. In the same year, *Dueling Network Architectures for Deep Reinforcement Learning* by Ziyu Wanh et al. Dueling DQN was introduced, which split the Q-estimation (estimation of the expected total reward) into two parts: checking the state value to understand how good the current position really is, and then checking how much better one action is from the rest. This allowed for more efficient learning in environments where the value of states varies more than the value of actions.

I reviewed these techniques in the context of *Reinforcement Learning and Video Games* by Yue Zheng (2019), where various Q-learning extensions were applied to the game T-Rex Runner. The study highlights that combining DQN variants with techniques such as batch normalization and image preprocessing significantly improved the performance of the agent. These results influenced the preprocessing pipeline and architecture choices for my own experiment, however, I had to keep in mind that the environment of my choice is much more complex than the T-Rex Runner game and that I couldn't expect the same results.

2.3 Proximal Policy Optimization

Value-based methods, like Q-Learning and Deep Q-Networks, still underperformed in continuous or complex action spaces. In order to overcome this, policy gradient algorithms were developed, which are able to optimize the policy directly, instead of relying on action-value functions, making them better suited for environments with continuous or high-dimensional action spaces. One of these policy gradient algorithms is Proximal Policy Optimization (PPO), which allows only slight updates of the policy, in order to avoid destruction of previously learnt good behavior of the agent and therefore stabilize learning. (Schulman et al., Proximal Policy Optimization Algorithms, 2017). This design allows for more robust and efficient learning in complex environments.

Several studies have shown PPO's effectiveness in even a real-life environment. *Self-supervised Domain Transfer for Autonomous Driving with Reinforcement Learning* by Moni and Gyires-Tóth (2025) and *Autonomous Drone Navigation for Smoke Tracking* underline PPO's ability to achieve impressive results despite being deployed in visually obstructed environments. These works were a good example for me that what I want to achieve is possible with reinforcement learning, because the *Elden Ring* environment is highly unpredictable and fast-paced and therefore can mimic a real-world setting in which a more traditional approach will not be a good option due to its limitations.

2.4 RL in Noisy, High-Dimensional Environments

Reinforcement learning in visually complex environments introduces a lot of unique challenges. For example, some of the common challenges are noisy observations, partial observability, delayed and inconsistent feedback and the need for quick decision making. These issues are not only present in video game settings, but can also appear when agents are trying to make sense of the real world around them. For example, autonomous driving vehicles require precise feedback from the environment around them, otherwise, they will make mistakes on the road and endanger the lives of the driver or other people.

In my work, I adapted a GitHub repository called "EldenRL" to allow PPO agents to

interact with *Elden Ring*. The agent's inputs consist of screen captures processed through computer vision pipelines and actions were mapped to keystrokes such as dodging or attacking. A major part of the challenge was to achieve consistency in obtaining the inputs. Original implementations relied on the OpenCV (Open Source Computer Vision) python library to detect health bars, to identify boss damage, stamina and character health, these however proved to be unreliable due to the lighting, background changes in the environment and small changes to the boss health bar after a successful hit, meaning that the values obtained were often wrong and therefore rewards were unpredictable and learning was tripped up. Similar challenges were discussed in *A Counterintuitive Approach to Explainable AI in Healthcare by Phillips* (2025), where agents trained in real-world medical imaging systems faced difficulties in generalizing due to noise and data irregularities. In both contexts, an improvement of the perceptual pipeline and engineering reward functions was needed in order to achieve any meaningful outcomes.

Analyzing *Edge Computing Empowered Holographic Video Communication* by Song et al. published in 2025, also helped me understand the exploration-exploitation dilemma, which is the agent's decision making between exploring new options or sticking with what was already learnt in order to focus on getting better reward returns. This paper helped me understand why my agent sometimes stuck to doing one simple strategy for a long period of time and how I could smooth out my reward function in order to avoid that.

2.5 Future Applications

In recent years reinforcement learning has managed to transition from simulated environments to real-world applications. In *Integrated Sensing, Communication, Computation, and Intelligence Towards IoT*, Wang et al. (2025) argue for the convergence of sensing and intelligence in edge computing systems, a convergence made possible by adaptive RL agents. Similarly, Hammad et al. (2025) in *Adaptive Cyber Defense with PPO* demonstrate how PPO can be employed to create self-evolving security systems.

My work contributes to this evolving discussion by showing how Reinforcement Learning agents can learn to perform complex tasks in unpredictable, noisy, and partially ob-

servable environments. While the use case is gaming, the lessons learned, such as the need for perception refinement and reward calibration, are directly transferable to topics like robotics and autonomous systems.

2.6 Conclusion of the Literature Review

This literature review has synthesized foundational and contemporary research in reinforcement learning, emphasizing on applications in complex environments. From classical Q-learning to advanced policy-gradient methods like PPO, the field has matured into a robust framework capable of solving high-dimensional, real-time decision problems.

By reviewing these techniques and applying them to my own project in the Elden Ring environment, I have demonstrated how modern reinforcement learning algorithms can be adapted to handle noisy visual data, design nuanced reward structures, and ultimately teach agents to operate in dynamic, challenging environments. The insights gained from this project and its literature base lay a strong foundation for future work in applying RL to real-world, high-stakes domains.

Chapter 3 Context of the Environment

The environment I chose for my experiment is the video game *Elden Ring*, a famous action role-playing game developed by *FromSoftware*. It falls within the subgenre of action RPGs known as "soulslike", a term derived from an earlier *FromSoftware* series, Dark Souls. "Soulslike" games are recognized for their intricate level design, punishing difficulty and demanding combat mechanics that require players to develop precise timing, pattern recognition and resilience, which are qualities reinforcement learning algorithms are able to acquire with enough practice. These characteristics make it a compelling yet challenging environment for reinforcement learning experiments. The game's high degree of unpredictability, partially observable states and complex visual feedback pose significant challenges for traditional Reinforcement Learning algorithms, thus providing a good opportunity to evaluate the robustness of modern approaches such as Proximal Policy Optimization.

For the purposes of this thesis, I selected the first major boss of the game, *Margit, the Fell Omen*, as the learning target for the RL agent. When personally playing the game, I had a hard time defeating this boss, even though I was a higher in-game level than the agent, had better gear and was able to heal, which is something the agent will not have access to. In other words the agent will have a harder time dealing with the boss than a normal player. The reason behind this is that if I wanted the agent to be stronger I will have to spend time playing the game and making the agent's character stronger, which will not help the learning experience, just make the task easier. *Margit's* behavior features a diverse set of attacks, some of them hitting rapidly and others with delayed wind-up animations, which have the goal of confusing the player and messing with the timing of his dodges. This makes the encounter an ideal testbed for evaluating an agent's capacity to make rapid, high-stakes decisions in real time. Furthermore, the environment simulates many of the complexities seen in real-world systems, such as occlusion, changing lighting conditions, delayed feedback and the need for continuous spatial awareness.

For a better understanding of what the environment looks like, refer to Figure 1 below.

Figure 1: *Direct screenshot from Elden Ring.*

Chapter 4 Methods

4.1 Background

One of the initial concerns about this thesis was if it was possible to achieve with the resources I have. Running the reinforcement learning algorithm alone is computationally intensive for the computer, however, running a game like *Elden Ring* and running the RL algorithm locally seems impossible with even a high-end computer. I found some people like the GitHub user "ocram444" who was able to run a similar project, "EldenRL", on his own computer. Thankfully, he also made the project open source, which helped jumpstart my own experiment. In "EldenRL", the game is supposed to be run on the GPU of the machine and the code algorithm trains on the CPU, however, this requires a lot of CPU power that I don't have access to.

In order to achieve what "ocram444" had achieved, I had to get creative. Since the code uses computer vision techniques and direct keystrokes, technically, running the game locally isn't needed for the algorithm to run. That is why I decided to use the cloud service Boosteroid to run *Elden Ring* from there and therefore free my GPU for use. This allowed me to run everything smoothly and even suspect that I had more power for computation than "ocram444" did.

4.2 Environment Setup

The environment is implemented using OpenAI Gym and interacts with *Elden Ring* by screen capturing via mss (Multi-Screenshot) and using the pydirectinput library for automated keystrokes. It is important to set the game consistently on the same resolution and the same place on the computer screen, because the screen capturing is screenshotting only a specifically defined area of the monitor.

4.3 Observation and Image Processing

Each environment step begins with grabbing a screenshot of the game using the mss library in Python. This is important because the algorithm uses these screenshots as a means of 'seeing', since it does not have actual access to the game. This is why it is important that everything works smoothly, because otherwise, one step takes too long and the algorithm is blinded. The screenshot is then downscaled to 400x225 pixels to reduce the computational load and a mask is introduced in order to hide parts of the screen in order to reduce the noise of the environment and guide the agent to what is important. This is done by turning the unneeded pixel black. Then the new down-scaled and masked frame is fed to the agent as the main visual input.

Separate regions of the screen are also cropped out in order to extract specific information and feed it explicitly to the agent. The specially cropped zones are the player health bar and stamina bar, the area where boss damage appears and some other visual clues needed for understanding death cause or if the agent is in a loading screen. The important cropped regions are processed using HSV masking to filter the colors: red for hp, green for stamina. The percentage of non-zero pixels in these masks corresponds to the current health and stamina of the player. The calculation function looks like this:

$$\mathrm{HP}_{\mathrm{percent}} = \frac{\mathrm{count\_nonzero}(\mathrm{mask}_{\mathrm{red}})}{\text{total pixels}} + \epsilon$$

$$\mathrm{Stamina}_{\mathrm{percent}} = \frac{\mathrm{count\_nonzero}(\mathrm{mask}_{\mathrm{green}})}{\text{total pixels}} + \epsilon$$

Where ϵ compensates for color variation.

The Boss damage detection is done by using OCR-based numeric recognition from the damage pop-up just above the boss health bar using EasyOCR. I chose to do this in a different way because the original logic with HSV masking was too unreliable due to the size of the boss health bar and the little damage that the agent does with one attack, the HSV simply isn't consistent enough and catches noise from the background so the

percentage of health may go up and down, despite it not moving at all. Using a threshold helped fix this issue for the player's health but did very little for the boss's health. That is why I chose to detect the numbers that pop up upon damaging the boss.

This design is similar to how a real player plays the game and visually parses different HUD elements, instead of relying on reading game state or memory reading. Reading the game state directly is a much more reliable way to collect information, however in order to analyze the challenges of developing reinforcement learning algorithms in complex and real-world environments it is better to use computer vision, since for example an autonomous vehicle will use similar techniques for "seeing" the world around it.

4.4 Action Space and Control

The agent operates in a discrete action space of twelve to fifteen options. These options include, moving, sprinting and dodging in all four directions, as well as using light attacks. This is a significantly reduced moveset compared to what was designed for the player base of the game. The goal of defeating a boss becomes much harder for a human if they can't use half of the designed commands, however, this was done to ease the learning process of the agent. By limiting the available moves the agent runs around less using useless items or healing in dumb situations during the early stages of training. This limitation also reduces the complexity of the reward function's design, because it allows me to skip a few of the rewards, which makes it less likely for the agent to start exploiting unbalanced rewards. Also actions are translated into keypresses by using pydirectinput, which has the ability to hold down keys, press keys and release keys. This simulates how a human player would control the character if he/she was playing the game normally.

4.5 Resets and Navigation

Of course there needs to be a way for the agent to get back to the boss room after dying, so that many iterations can be run without human supervision. This is where the check for loading screen function and walkToBoss class come into play. The "check for loading screen" function is called upon player death, or in other words when the player health

bar drops below one percent or when it disappears completely for some reason. Then the function checks for a "Next" in a specified area of the screen and when that text disappears it calls "walk to boss". Then a hardcoded navigation script is run that brings the character into the boss room, locks the screen to the boss and leaves the rest to the agent. This avoids the need for learning open-world navigation and focuses the reward function explicitly on the boss fight.

4.6 Reward Function

The reward function includes survival bonuses and hit detection. The primary challenge lies in reliably detecting boss damage and attributing it correctly, which was done with the computer vision methods mentioned before.

During testing, a lot of different reward functions were implemented, some of them focused on attacking, others on survival. I ended up testing two different reward functions: one of them aggressive, rewarding dealing damage, and the other one, with normalized rewards.

The first function rewards 150 for the first hit and adds 150 to the reward of every next hit, meaning that hit number one gives a reward of 150, but hit number 2 - 300, together that scales up to 450. There is also a small reward for every step taken, which diminishes with the total steps taken in the run. The idea behind that is that in the future, the algorithm will not be rewarded for simply surviving in the environment without doing anything meaningful. There is a negative reward for taking damage, which is -40. Lastly, there is a negative reward for dying to the boss, but an even bigger negative reward for dying due to falling off the map. The large and scaling damage reward aims to make the algorithm aggressive, so it quickly learns that it needs to hit the boss.

The second function is similar. The only difference is that the damage reward is scaled down by multiplying the hit counter by 50 and not 150, so the first hit is 50, the second is 100 and together they are 150, instead of 450. However, the big change in this reward function is that it has normalized rewards. Normalized rewards ensure that the total return in each episode is scaled to a consistent range, typically centered

around zero. This prevents the agent from being overwhelmed by large absolute values and helps the critic network produce more stable predictions. In practice, this means that all rewards—positive and negative—are adjusted relative to their distribution across recent episodes. The normalization is implemented by subtracting the running mean and dividing by the standard deviation of the cumulative rewards.

4.7 Model Architecture

For the training of the RL agent I used Proximal Policy Optimization (PPO), a policy-gradient method introduced by Schulman et al. (2017). I chose to use PPO because it balances the robustness offered by trust region approaches and is also easy to implement, making it particularly suited for environments like mine, where data is limited and the environment feedback is noisy. PPO belongs to the class of policy gradient methods that optimize a policy $\pi_\theta(a \mid s)$. This is used to calculate the probability of taking action a when the agent is in a state s. The policy relies on parameters θ, which are the weights of the neural network. For example, the policy can be something like: "There is a 70 percent chance I'll move left when in state X", then after collecting enough data the agent may update the percentage of taking that action in state X, this is our θ, in order to start getting better rewards. Unlike value-based methods like Q-learning, PPO directly updates the policy based on sampled interactions with the environment and an estimated advantage function $\hat{A}_t$, which tells the agent how much better an action was than expected at a given state. (Schulman, 2017) PPO maintains a copy of the old policy $\pi_{\theta_{\text{old}}}$ and computes the probability ratio $r_t(\theta)$. The probability ratio is essentially the ratio between how likely the new policy would take an action, compared to how likely the old policy was to take that same action.

What distinguishes PPO the most from other policy gradient methods is clipped surrogate objective, which stabilizes training by limiting the size of the policy updates. This was implemented because PPO remembers only the last policy, so that the model uses less memory and has a lower complexity. This means that big policy updates may delete previously learned good behaviors and limiting the magnitude of the policy changes

prevents this from happening. (Schulman, 2017) The clipped objective is defined as:

$$L^{\text{CLIP}}(\theta) = \hat{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right]$$

Let's break this formula down:

- $\hat{E}_t$ is the empirical expectation over a batch of timesteps t, meaning that we take the average of the expression inside the brackets across all collected samples.
- θ represents the parameters of the current policy that are being optimized.
- $r_t(\theta)$ is the probability ratio.
- $\hat{A}_t$ is the estimated advantage at timestep t, which calculates how much the chosen action was better or worse compared to the expected action in this state.
- $\hat{A}_t = Q(s_t, a_t) - V(s_t)$
- $Q(s_t, a_t)$ is the Q-function, which tells us how good it is to take a specific action a in state s.
- $V(s_t)$ is the value-function that shows how good the state s is on average.
- ϵ is a small hyperparameter (commonly set to 0.2) that defines the size of the trust region. In other words, it determines how much the new policy is allowed to deviate from the old one.
- `clip` is used to restrict the probability ratio within a safe interval. This limits how much the policy is allowed to change in a single update by using the hyperparameter ϵ. If $r_t(\theta)$ is between $1-\epsilon$ and $1+\epsilon$, then $r_t(\theta)$ is unchanged. If it is out of bounds, it is clipped to either $1-\epsilon$ or $1+\epsilon$, depending on whether it is below or above the set range.
- The `min` function ensures that the final loss favors more conservative updates: it uses the smaller of the unclipped and clipped values, which means that PPO avoids overly optimistic updates even when the advantage is large.

The clipping mechanism is what gives PPO its robustness: if the new policy changes too dramatically, the update is penalized, allowing for a more stable learning (Schulman, 2017). See Figure 2 for a visual reference.

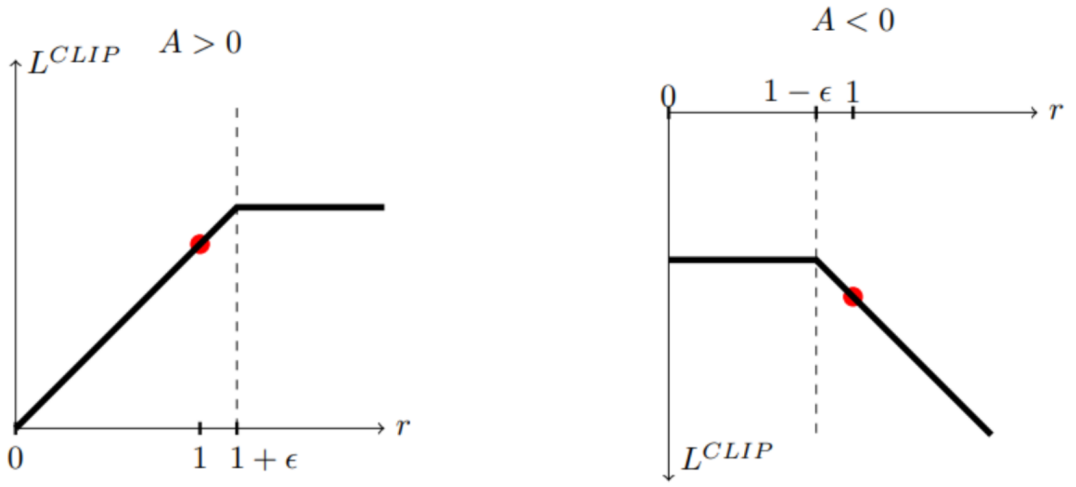

Figure 2: *Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms.*

A standard PPO model uses an actor-critic architecture that is implemented with a single neural network that outputs both a policy and a value function by splitting the input into two parts. The policy, or the actor, decides which is the most optimal action to take in the current situation and the value function, the critic, estimates how good the current state is. This allows the agent to both learn how to make decisions and how to evaluate them.

In environments with image-based inputs, the PPO model usually begins with a series of convolutional layers (Conv2D), which are used to extract spatial features from the given input image. As the image passes through deeper convolutional layers, the model captures increasingly abstract and higher-level features. So in order to be able to detect shapes, motions and characters better, I introduced a third convolutional layer to the model. After these convolutions, the resulting feature maps are flattened into a one-dimensional vector, which is then passed through a dense (fully connected layer to form a compact representation of the state, as you can see in Figure 3.

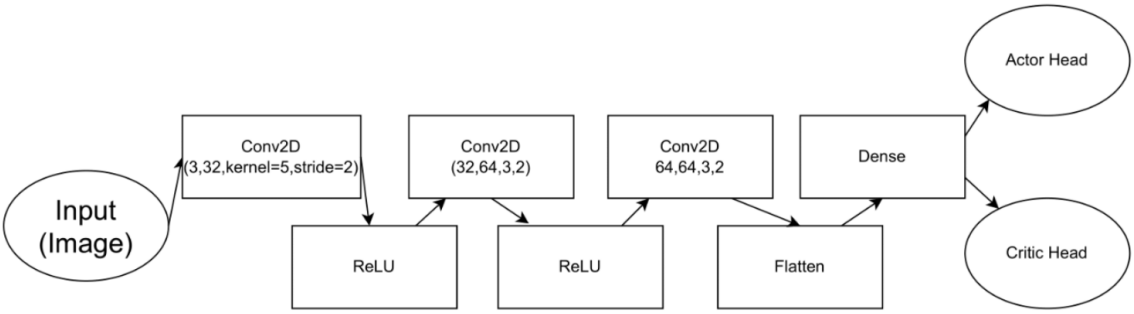

Figure 3: *Model Architecture*

Here is an explanation of all the different layers and parameters in Figure 3:

- **Convolutional layer (Conv2D)** – Applies filters to the image in order to extract features. The filters then slide over the input image and compute a dot product with local patches. This helps it detect shapes, edges, characters, color, and other visual patterns.
- **In channels** – This is the number of input channels; in the first Conv2D it is 3, for an RGB image.
- **Out channels** – The number of filters (feature maps) the layer will learn and output.
- **Kernel size** – This determines how much of the image each filter sees. In the first convolutional layer we have 5, which equals a 5 x 5 patch.
- **Stride** – Determines the number of pixels the filter moves each step. In our case this is two, meaning that the filter skips every second pixel.
- **ReLU** – Used to keep only the positive values and set the negatives to zero, which allows the model to keep only some of the neurons active, improving efficiency.
- **Flatten** – Turns a multi-dimensional tensor into a one-dimensional vector, in order to prepare the data for the dense layer.
- **Dense Layer** – Integrates the features and compresses the input into a meaningful representation.

- **Actor Head** – Outputs the policy $\pi(a \mid s)$.
- **Critic Head** – Outputs the value function.

In addition to processing the full vision frame, I created a custom multi-input policy in the model so that I could include a cropped image of the agent's health bar. This additional input gives the agent a clearer, more consistent signal for understanding the health system in the game. This cropped image is of lower resolution, so it is only passed through one separate convolutional layer before being added to the dense layer.

In conclusion, I have slightly modified the base PPO model to have an extra convolutional layer and to also receive multiple inputs. The logic behind the algorithm is the same, I have only tuned the model's architecture a bit to be more compatible for my experiment in *Elden Ring*.

4.8 Iteration Logs

In order to collect data from the training I used two different methods of logging information. The first one is to use Tensorboard to save training logs in a dedicated folder and visualize the information via Tensorboard. This allowed me to track things like value loss and policy loss, entropy and learning rate behavior. The other method is to log the information about the run in a custom death logs csv. In this file every death is saved by writing the trial number, total reward received for that run, the total damage done to the boss, the death cause, either by "Fall" or "Boss" and lastly it saves the steps taken inside that run. All of this information is important because it gives statistical context about how the agent learns in real time. By having the death logs csv it is incredibly easy to see if the agent is progressing in damaging the boss, not falling or surviving longer, even while training is ongoing. The two methods combined allow me to track both high-level training dynamics and low-level behavioral patterns, giving a complete picture of the agent's learning progress.

4.9 Evaluation Metrics in Tensorboard

To understand the training performance of the reinforcement learning agent, TensorBoard provides a set of evaluation metrics that reflect different aspects of how the PPO algorithm is learning. Each metric reflects a specific component of how the PPO algorithm is learning, so in order to analyze the results I will explain what every evaluation metric is and what values should be expected.

- **Approximate Kullback-Leibler Divergence** measures how much the current policy has changed after the update. The ideal behaviour of the value is to stay low but not be non-zero. The typical range is between 0.01 and 0.03. If it is too high, PPO changes the policy too rapidly and can destabilize learning.

- **Fraction of Updates Clipped** shows what percent of the updates were clipped. If this value is high, it means the algorithm had to discard more updates in order to avoid large changes to the policy. The ideal value is something lower than 0.25.

- **Clip Range** is a hyperparameter used for clipping. The default value is 0.2. Choosing a smaller value results in more conservative policy updates.

- **Entropy Loss** explains the explorational behavior of the agent. High entropy means more exploration and low entropy means a more deterministic behavior. Ideally this value starts high (e.g., -2.5), decreases slowly, and stabilizes around -1.0. If the entropy loss drops too early, the agent can become confident in suboptimal behaviors.

- **Explained Variance** compares the variance of the prediction error to the variance of the actual returns. The possible values range from -1.0 to 1.0. A higher value means the critic is predicting well the reward returns based on the actions it is doing, while a negative reward means it is worse than predicting nothing. Ideally, this starts low or negative and rises to at least 0.3 with training. In simple terms, explained variance indicates how well the critic can estimate how good a state is.

- **Loss** is the aggregate loss, calculated from policy loss (actor), value loss (critic), and entropy loss. The loss should trend downward with training.
- **Policy Gradient Loss** measures how well the policy improves based on the estimated advantage. A negative value will mean it is pushing the policy towards a higher reward.
- **Value Loss** measures how far off the critic's predictions are from the actual returns. The expected behavior of the value loss is to start high and steadily decrease.

In conclusion, these evaluation metrics offer insight on how effectively the PPO agent is learning over time. Monitoring indicators like explained variance and value loss helps evaluate the critic's learning quality, while entropy and policy gradient loss reveal how the policy evolves. A good training run should have gradually decreasing loss values, increasing explained variance and a slow reduction in entropy. I will use these evaluate metrics to analyze my results in the next section.

Chapter 5 Results

5.1 Overview

This chapter presents the results obtained from training the reinforcement learning agent using the Proximal Policy Optimization algorithm in the custom *Elden Ring* environment. During the past month, I managed to run a lot of models with different hyperparameters, different architectures and different reward functions. The biggest challenge for me was to get the reward function right, something that proved to be more challenging than expected, that is why I will be analyzing the results of the last two agents that I trained. The first agent uses a very aggressive reward function that heavily rewards dealing damage and the second one uses normalized rewards and a more scaled down version of the damage reward system. Results are drawn from both high-level metrics (TensorBoard logs) and low-level behavioral data (death logs) which provide insight into the agent's learning progression, survival patterns and combat performance.

5.2 High Damage Reward Agent - Tensorboard

Figure 4 shows the evaluation metrics of the High Damage Reward Agent (HDR Agent). The HDR Agent was trained for 253 iterations and it managed to complete a bit under nine thousand steps.

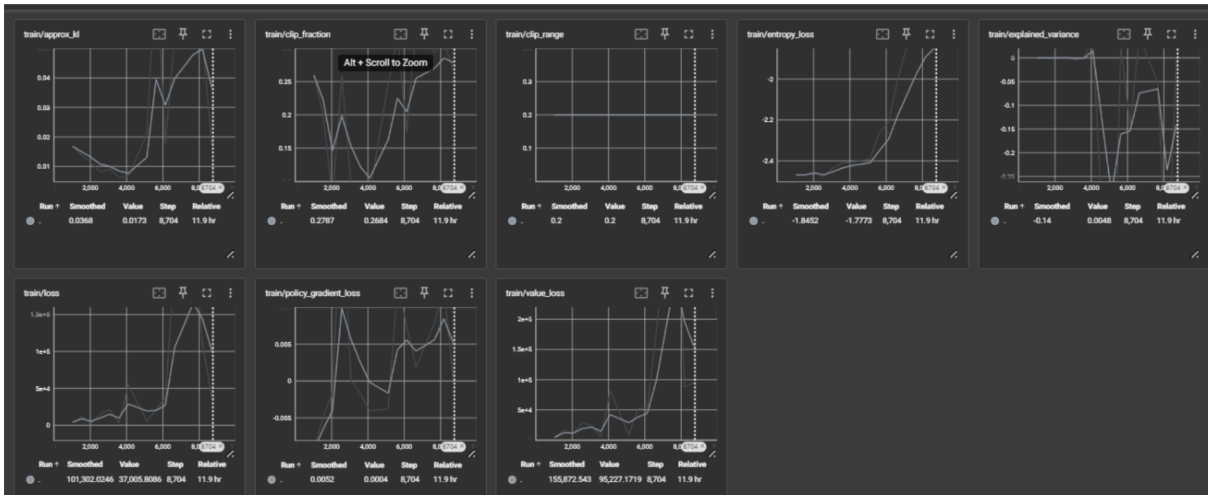

Figure 4: *High Damage Reward Agent - Tensorboard Evaluation Metrics*

As we can see from Figure 4 the results are underwhelming. It is important to note that the agent is still early in its training process at around nine thousand steps and it may become much better when it reaches around a hundred thousand steps but that will take a large amount of training time and the results that we see from the evaluation metrics don't imply that this agent will get much better that it is already.

Firstly, let's analyze the metrics that are in the expected intervals. Approximate KL Divergence is between 0.3 and 0.4, meaning that the policy updates aren't too aggressive. The entropy loss is around -1.8, which means that the agent is still exploring, however it is rising rapidly, suggesting that in the next few thousand steps, the agent may become deterministic, restricting future policy updates, by doing the same thing over and over again in its runs. This will result in the agent plateauing in its learning. The clip fraction is a bit high with a value jumping to 0.25, meaning that twenty five percent of the policy updates were clipped. This suggests that the policy may be changing too much, however a value of 0.25 is not a major problem, especially compared to the next few metrics. There is not much of a point discussing the loss and value loss separately, since the loss is just affected by the extremely large value loss. Incredibly the agent has achieved a value loss of over 150,000 after its last update, which means that the critic is massively wrong when predicting the reward return based on the state it is in. This of course affects the explainable variance, which is negative, meaning that the value function explains the actual returns worse than randomly predicting them.

By analyzing the results it is very noticeable that the main suspect for the bad results is the rewards system. The rewards for doing damage are simply too high and scale way too much. Of course I was also looking at the training process and it was easy to notice that the agent was really focusing on just using the attack. To be completely fair looking at the training process the agent seemed to be getting better at the game, sometimes even achieving as much as dealing more than one third of the boss's health bar. However, the result wasn't satisfactory because the agent valued dealing damage way too much and was dying way too quickly. For the next agent I decided to use reward normalization and scale down the damage reward.

5.3 Normalized Reward Agent - Tensorboard

According to Figure 5, the evaluation metrics for the normalized rewards agent are completely opposite. This time the Approximate KD Divergence is high for PPO's standards, suggesting that there is a significant difference between old and new policy, which means that the policy updates are way too large. This is confirmed by the clip fraction, which shows that around the ten thousand step, seventy-five percent of the policy updates are getting clipped. The entropy is relatively high for now, but it seems to be dropping, so it should not be an issue if future training is done. The critic is still underperforming as shown in explained variance, which seems to be jumping up and down from - 0.1 and - 0.3. However, the loss values have significantly improved, meaning that reward normalization is somewhat working, because the critic is now predicting values in a reasonable range. Of course, according to the explained variance, these predictions are still wrong; they are just much closer. This phenomenon was expected because the normalization centers the awards around zero, so that there won't be large differences between the returns, which helps the critic estimate the rewards, because it will have a lower range of numbers to guess from, not because it has a better understanding of why it is getting the rewards.

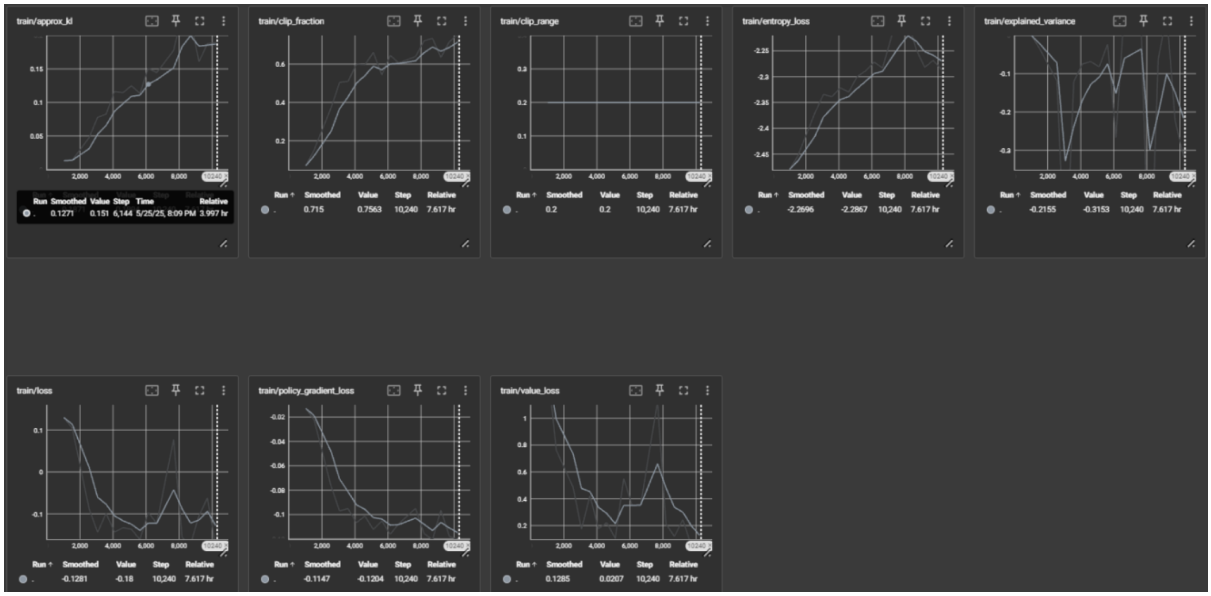

Figure 5: *Normalized Rewards Agent - Tensorboard Evaluation Metrics*

5.4 Death Log Trends

The idea behind the implementation of the death log was to prove that the agent was actually learning things, despite the bad learning metrics results. When I was overlooking the training of the agents, I noticed around the two hundred steps, they started showing signs of at least understanding the goal of the game. The agents started focusing more on dealing damage and stopped falling off the map so much, so I decided that I should investigate if there are any positive trends in the agents' performance over time.

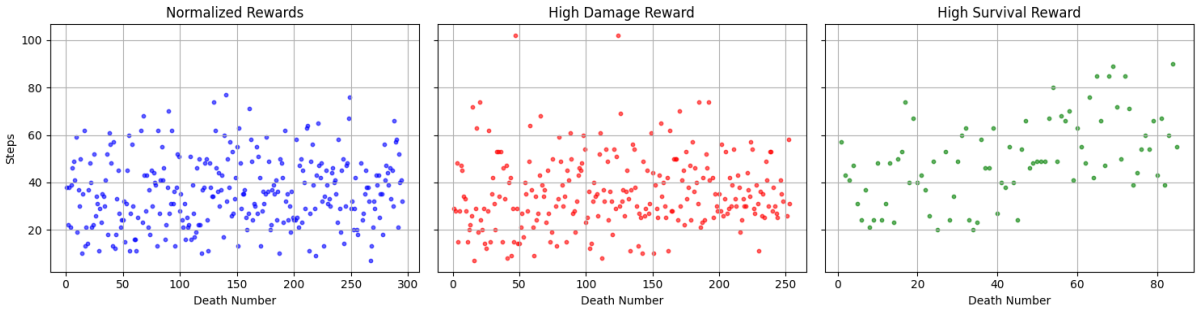

Figure 6: *Steps per death across reward schemes: Normalized, High Damage, and High Survival.*

The first trend I started looking for was if the agent survives longer over time. In Figure 6, three scatter plots of the steps per death iteration are shown. The first two plots show the previous two agents, who, despite having a negative reward for taking damage and a small reward for steps completed, don't focus on learning to survive at all. Both these agents show a correlation between steps and death iterations of under 0.10, proving that they don't learn to survive longer with time. The reward systems of these two models barely reward it for survival and so I had to analyze if the case is the same for one of my older agents that had a very high reward for surviving more steps. Here, the correlation is approximately 0.52, proving that there is a moderately positive correlation between steps and iterations. This model was trained for a much shorter period of time, because the reward function focused too much on survival and the agent was simply running away from the boss. In conclusion, if the agent is focused on surviving for longer, it is able to learn to do it, but at the expense of dealing damage.

It is also important to analyze the cause of the deaths, not only how many steps the

agents took. This is why in Figure 7, I visualized the causes of death by splitting them into 'Boss' and 'Fall'. For context, the algorithm for detecting the cause of death isn't 100 percent accurate and the 'Fall' death cause is inflated with around 20 percent in my estimation, but this doesn't influence the analysis over time. Also, it is important to note that the High Damage Rewards Agent was trained for 253 iterations and the Normalized version was for 295. Also, some of the iterations were removed because they were under 5 steps, which happens if the agent gets stuck before entering the boss room.

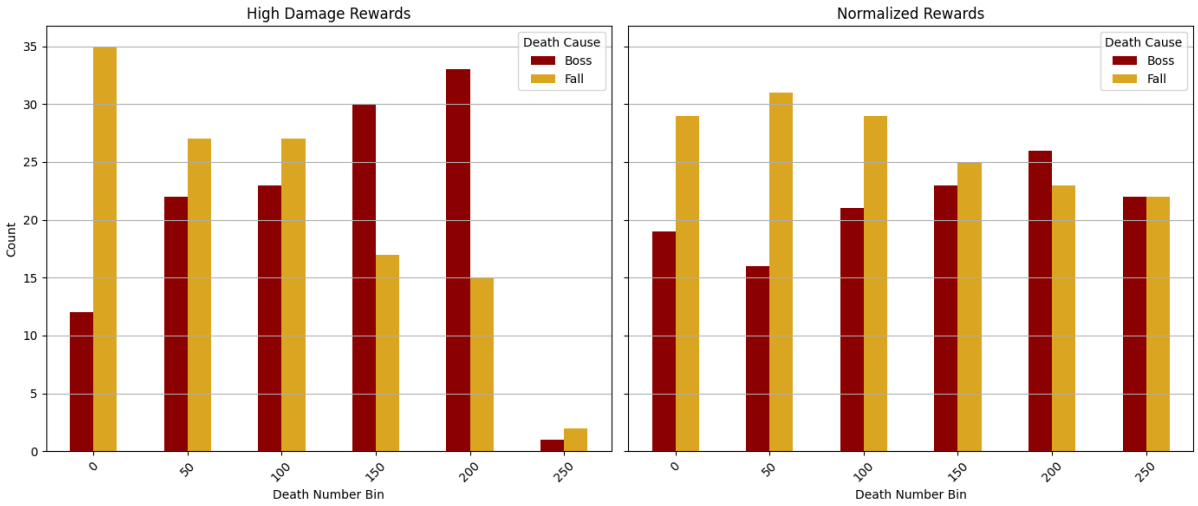

Figure 7: *Death cause distribution over time for High Damage (left) and Normalized Reward (right) agents.*

The plots in Figure 7) clearly show that at the start of the training, the agent tends to jump a lot off the map, with time this reduces and the High Damage Rewards Agent actually starts dying to falling quite rarely. I suppose this is due to its highly aggressive nature; it just goes towards the boss and starts attacking, resulting in more 'Boss' deaths and fewer 'Fall' deaths. The Normalized Rewards Agent is not as aggressive and tries to dodge much more than the previous version. This, despite being a good strategy, results in the agent falling off the edge of the map more often. Again, analyzing the death cause leads me to believe that the agent is in fact, learning some correct behavior and is not just randomly clicking buttons. Maybe the most important evaluation of the model is the damage it manages to do, since the goal of the game is to beat the boss by dealing damage.

In Figure 8, I have analyzed the average hits to the boss for every 25 death iterations.

For context, the last bar in High Damage Rewards is so high because it contains only 3 iterations, instead of 25. Looking at the first plot, the trend is very noticeable and it is clear to see that over time the agent starts landing more hits. In contrast, the Normalized version seems to stagnate and even drop significantly. This happens because the entropy is still high at -2.2 and the policy updates are very large according to the approximate KD divergence and clip fraction. In other words, the agent is still exploring a lot and, on top of that, is trying to pass big policy updates, resulting in drastic changes to the agent's strategy.

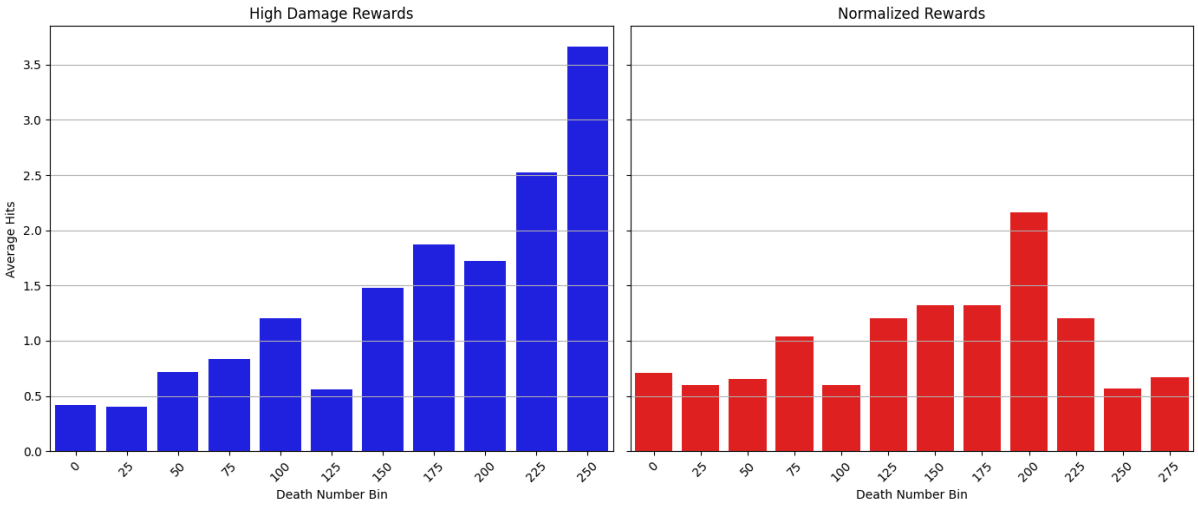

Figure 8: *Average hits per death over time for High Damage (left) and Normalized (right) reward strategies.*

Chapter 6 Limitations and Future Developments

The current implementation demonstrates underwhelming results, however, it is constrained by several technical and methodological limitations that highlight key opportunities for future improvement. The first step for future development is to let the algorithms train for more steps and evaluate the results further. This can be done by simply letting the agent train longer in the environment or using vectorized environments to be able to run several iterations at the same time. This will significantly boost the training process. The major issue with this project is the inconsistency of the reward detection. The computer vision techniques used in this project are rather unreliable and it is a complicated job to get them to be even usable. The OCR detection for boss damage is the best method that I tested for damage detection, but it sometimes doesn't even log the hit or logs the hit twice. That translates to the reward not being given instantly after the algorithm chose to attack and therefore confusing the algorithm as to which action yielded the reward and resulting in a very low or negative explainable variance. In order for this to be fixed a reliable and instant method is required for detecting boss damage. This can be done by giving the reward to the step with the last used attack, however, the agent tends to press the buttons much more often than required, resulting in doing actions while in attack or dodging animations, which results in those actions not happening. I suspect this also damages the learning procedure and in order to fix it, the timing for each animation should be recorded and the algorithm should be able to do another action only after the animation. Similar implementations should be done for the player health detection, but that is a bit more abstract since its changes don't come from the player's actions directly, but the boss's. Another technique that needs to be implemented is reading the boss's location and awarding the agent for staying close to the boss but not taking damage. This can be done by using a CNN to detect how much of the screen is taken by the boss, since he is pretty large. This will guide the algorithm

into surviving longer, via dodging enemy attacks, instead of keeping its distance.

In conclusion, the logic behind the project is sound, but the methods used are unreliable due to the noise in the environment. In order to take this project to the next step better computer vision techniques will be required, which may raise the complexity of the model and require better hardware to run.

Chapter 7 Social Aspect

While this thesis focuses primarily on the technical development and evaluation of reinforcement learning agents in a complex game environment, it also touches on broader social implications. As RL systems become more capable of navigating uncertain and high-stakes environments, their potential applications extend into areas like autonomous vehicles, healthcare, and robotics, domains where decision-making has real-world consequences. Developing these systems responsibly, requires not only improving performance, but also ensuring transparency, reliability, and safety. By experimenting in a game like *Elden Ring*, this project provides a safe and controllable environment for testing how RL agents might behave in less forgiving environments, raising important questions about how we design, train, and trust these systems as they become more integrated into society.

Also, using popular games as testbeds for reinforcement learning has the potential to make complex AI topics more approachable to a wider audience. Viral video games like *Elden Ring* already interest millions of teenagers and young adults, so when these games are used as platforms for research, they create a compelling entry point for younger generations to explore artificial intelligence, machine learning, and programming. This convergence of gaming and AI research could serve as a gateway for educational engagement, helping demonstrate technical subjects and inspire future innovation in the field.

Chapter 8 Conclusion

This thesis demonstrates the potential of reinforcement learning in complex environments, but it also discusses the challenges and limitations of its implementation. A big part of the thesis is focused on how exactly to set up the model to work for the specific environment and what issues might arise during setup. The work also discusses the different evaluation metrics and explains each one of them in detail, providing information that is needed to understand the training process of the agents. Using *Elden Ring* as a testbed, PPO agents were trained to learn, despite the imperfect feedback. Early results show few signs of adaptation and until better computer vision methods are implemented to deal with the noisy environment, an agent who will be able to defeat *Margit, the Fell Omen* seems infeasible.

This work is also important to research because it bridges the gap between controlled simulation environments and the unpredictability of real-world systems. Most reinforcement learning research is conducted in clean, abstract environments with perfect state information, which often fails to reflect the messiness of real-world perception and control. By applying RL in a game like *Elden Ring*, which features high-dimensional visual input, partial observability, and delayed, noisy feedback, this thesis pushes the boundaries of where RL can be tested meaningfully. The environment demands quick adaptation, strategic behavior and precise action under uncertainty, which are challenges that mirror those found in real-world robotics, autonomous navigation and decision-making systems. As such, this research contributes toward understanding how RL models behave under realistic constraints and what tools are needed to bring them closer to general-purpose intelligence.

Code Repository

The full source code used in this thesis is publicly available on GitHub:

`https://github.com/VlaadiAangelov/EldenRing_RL_Thesis`

This repository contains the environment setup, training scripts, data logs, and custom reward functions used for the experiments conducted in this thesis.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). *Human-level control through deep reinforcement learning*. Nature, 518(7540), 529–533. https://doi.org/10.1038/nature14236

[2] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. arXiv preprint arXiv:1707.06347

[3] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.

[4] Zheng, Y. (2019). *Reinforcement Learning and Video Games*. MSc Thesis, University of Sheffield.

[5] Hong, J., et al. (2025). *Autonomous Drone Navigation for Smoke Tracking Using PPO*. arXiv:2504.12664.

[6] Moni, R., & Gyires-Tóth, B. (2025). *Self-supervised domain transfer for driving with RL*. ScienceDirect. https://doi.org/10.1016/j.eswa.2025.123456

[7] Yin, J., & Xiang, T. (2025). *HEA-PPO for Multi-Vehicle Navigation*. Expert Systems with Applications. https://doi.org/10.1016/j.eswa.2025.456789

[8] Phillips, V. (2025). *Counterintuitive Approaches to Explainable AI in Healthcare*. AI & Society, 40(2), 267–289.

[9] Hammad, A., et al. (2025). *Adaptive Cyber Defense with PPO*. FutureTechSci, 12560.

[10] Wang, X., et al. (2025). *Edge Intelligence with RL for IoT Systems*. Journal of Electronics & Information Technology, 47(6).

[11] Ocram444. (2023). *EldenRL: Reinforcement Learning in Elden Ring*. GitHub repository. https://github.com/ocram444/EldenRL