Bachelor Degree Project

UNIVERSITY
OF SKÖVDE

Evaluating the effects of hyperparameter optimization in VizDoom

Bachelor Degree Project in Information Technology
Basic level 30 ECTS
Spring 2022

Markus Olsson, Simon Malm, Kasper Witt

Supervisor: Juhee Bae
Examiner: Joe Steinhauer

Abstract

Reinforcement learning is a machine learning technique in which an artificial intelligence agent is guided by positive and negative rewards to learn strategies. To guide the agent's behavior in addition to the reward are its hyperparameters. These values control how the agent learns. These hyperparameters are rarely disclosed in contemporary research, making it hard to estimate the value of optimizing these hyperparameters.

This study aims to partly compare three different popular reinforcement learning algorithms, Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C) and Deep Q Network (DQN), and partly investigate the effects of hyperparameter optimization of several hyperparameters for each algorithm.

All the included algorithms showed a significant difference after hyperparameter optimization, resulting in higher performance. A2C showed the largest performance increase after hyperparameter optimization, and PPO performed the best of the three algorithms both with default and optimized hyperparameters.

keywords: vizdoom, reinforcement learning, hyperparameter optimization

Acknowledgment

We thank Juhee Bae at the University of Skövde for her extended support during our thesis project. The authors also thank Robert Svensson for donating a GTX1070ti to the project.

Contents

1 | Introduction

In recent years, there has been a surge in research on deep learning and Reinforcement Learning (RL) applied to the area of video games. The first breakthrough that ignited this surge was when Mnih et al. (2013) presented their deep reinforcement learning model that successfully learned policies directly from raw video data. Today, research is ongoing on all types of video games, ranging from first-person shooters to real-time strategy games (Justesen et al. 2020). One application is that of game testing, where deep reinforcement learning has been shown to increase test coverage (Bergdahl et al. 2021). Another is to create competitive opponents for human players (Khan et al. 2020). The performance of such deep reinforcement learning algorithms is dependent on their associated hyperparameter settings, which control different aspects of such reinforcement learning algorithms. For example, the gamma hyperparameter dictates how much weight an algorithm gives to future rewards as opposed to rewards closer in time.

In the machine learning community, one big problem today is that the chosen hyperparameter settings are often omitted from the research articles. As Lynnerup et al. (2019) states in their research, this negatively affects the reproducibility of deep reinforcement learning research. In their research Vohra et al. (2022) compares how the hyperparameter settings of the underlying neural network architecture of the two different deep reinforcement learning algorithms, Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC), affect performance. The study shows that the hyperparameter settings greatly affect the performance of these algorithms; however, the study is limited to the neural network hyperparameter settings and does not include the hyperparameter settings of the algorithms themselves.

In this study, the reinforcement learning algorithms Proximal Policy Optimization (Henderson et al. 2018), Advantage Actor-Critic (A2C) (Mnih et al. 2016), and Deep Q Network (DQN) (Mnih et al. 2013) are compared with each other using the 3D environment VizDoom (Kempka et al. 2016) based on the classic Doom video game from 1993. This expands the research done by Vohra et al. (2022) by including more algorithms, and the focus will be on the hyperparameter settings of the algorithms themselves.

In their work Henderson et al. (2017) concludes that small implementation details greatly affect the performance of deep reinforcement learning algorithms. Due to this, the well-documented and tested Stable-Baselines3 framework (Raffin et al. 2021) was used for all algorithm implementations. To investigate the effect of hyperparameter optimization, the hyperparameter optimization framework Optuna (Akiba et al. 2019) was used. In a recent article by Shekhar et al. (2022), it was deemed one of the most efficient tools for hyperparameter optimization.

2 | Background

This section aims to cover the building stones used for the experimental part of the study. It will cover the fundamentals of reinforcement learning and associated neural networks and briefly discuss the three different reinforcement learning algorithms used in the experiment. In addition, it will discuss hyperparameter optimization and the proposed training environment. Finally, research work related to the study will be discussed.

2.1 Artificial Neural Network

Artificial Neural Networks (ANNs) are commonly used structures that have some of their properties inspired by neurons in the brain. Recent advances in deep learning refer to ANNs with many hidden layers; this includes reinforcement learning. Figure 2.1 shows a basic feed-forward ANN, where the signal only travels to neurons in the next layer. A recurrent Neural Network could have signals going backward or to the same layer. Each link has an associated real value that determines the neuron's activation level. The activation is determined by a semi-linear combination of the input values of the previous layer (Sutton & Barto 2018).

2.2 Convolutional Neural Network

A convolutional neural network (CNN) is a class of artificial neural networks consisting of multiple layers in deep learning, used mainly to visualize images. It is one of the most useful and strongest tools when it comes to handling large data sets for research areas such as image recognition (S. Albawi & Al-Zawi 2017). The convolutional neural network is one of the most popular for deep learning, and its name comes from a mathematical linear operation between matrices called convolution. A CNN consists of multiple layers such as; a convolutional layer, non-linearity layer, pooling layer, and a fully connected layer. Although the convolutional layer and fully connected layers have parameters, the pooling layer and the non-linearity layer do not. CNN is a great tool in machine learning because of the performance benefits it brings, especially for areas such as image recognition and processing languages, among many other areas of application that are possible for a CNN.

2.3 Markov Decision Process (MDP)

Markov Property: The future is independent of the past, given the present. That is, the present state contains all the relevant information needed to statistically represent the future.

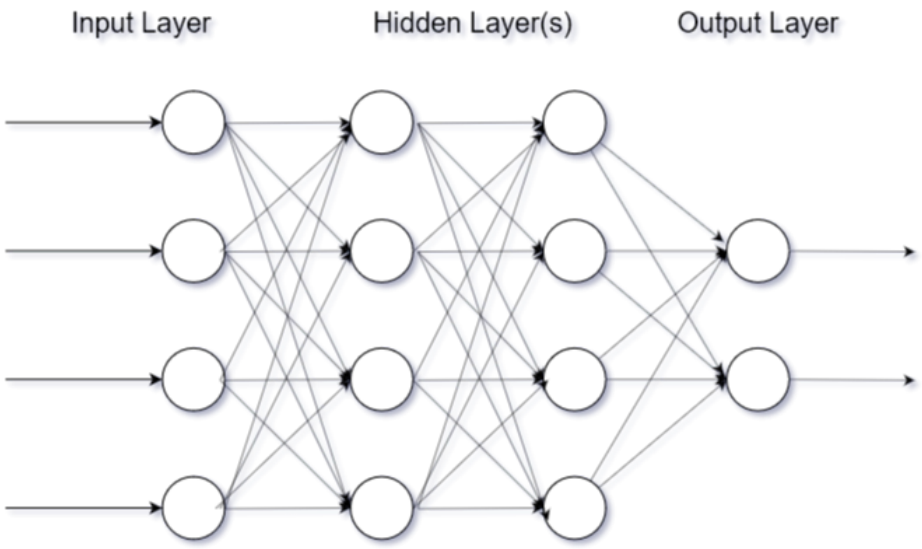

Figure 2.1: Figure showing basic ANN with 2 hidden layers.

Markov Chain: A finite set of states (S) and a state probability transition matrix (P) that define the probabilities of transition between the states.

A Markov Decision Process combines these elements together with the set of actions, rewards, and the discount factor γ. The discount factor $\gamma \in [0, 1]$ reduces the value of future rewards by a factor of $\gamma^k R$ where k denotes the time steps ahead of the current state. Using these elements, an MDP maps all action value pairs instead of the state value pairs. This creates a complete mapping of which action would return which value given every possible state in the MDP.

A policy is a mapping of which action to take given a state $\pi(a|s)$. When the following policy π takes actions based on expected returns, this is denoted as $q_\pi(s, a)$. Then, tie this together with the basic goal of reinforcement learning by describing the optimal actions to take given the state presented. $q_*(s, a) = maxq_\pi(s, a)$ q_* denoted the optimal reward given state s and action a.

Thus, an optimal policy can be found by maximizing the number of $q_*(a, s)$ actions in the policy (Silver 2015).

$$\pi_*(a|s) = \begin{cases} 1 if a = argmaxq_*(s, a) \\ 0 \text{ otherwise} \end{cases} \tag{2.1}$$

2.4 Deep Reinforcement Learning

For machine learning, you need a strategy for the approach you are going to use; an example of this would be with the help of deep reinforcement learning (François-Lavet et al. 2018), two other common approaches are supervised and unsupervised learning. The idea of reinforcement learning is to allow an agent to learn from interaction with its environment, where it learns depending on what actions are considered either bad or good, where the good actions are encouraged. From

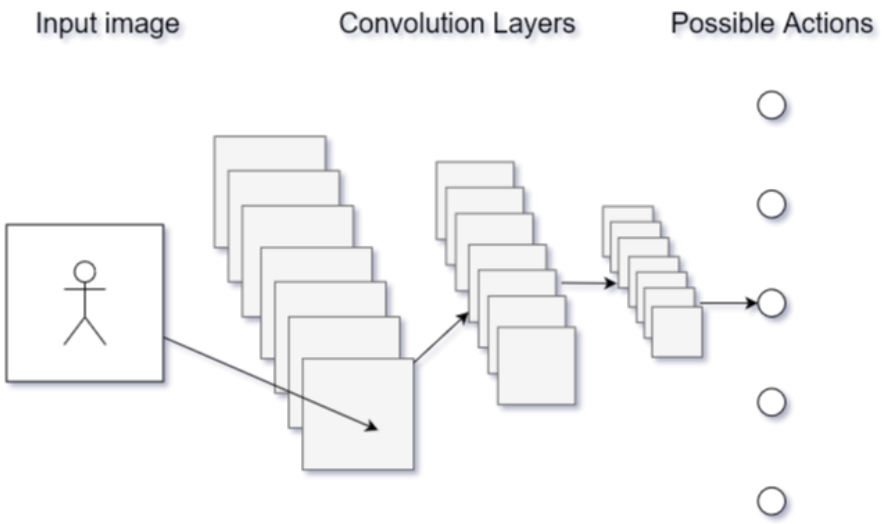

Figure 2.2: Figure showing the basic structure of CNN

$$\begin{bmatrix} S_{11} & ... & S_{1n} \\ ... & & ... \\ S_{n1} & ... & S_{nn} \end{bmatrix}$$

Figure 2.3: State transition matrix

learning by reinforcements, an agent accumulates experience and can use it to optimize some chosen objectives in its environment with the help of growing reward sizes. Reinforcement learning has become popular as a way to address challenging sequential decision-making problems, e.g. in environments such as chess or even more modern games like Starcraft 2 (Shao et al. 2019).

Machine learning provides ways in which automated methods can detect patterns that are fed to them and can subsequently perform tasks with the help of the detected patterns. The deep learning approach makes use of a neural network that consists of successive processing layers, each of which is a nonlinear transformation in nature, and each sequence of the layers leads to differing levels of abstraction. Most modern applications make use of many different types of neural network layer, where each different type has its specific advantages, as well as flaws or uses. The one you use would ultimately be based on what you are trying to achieve (Lapan 2018).

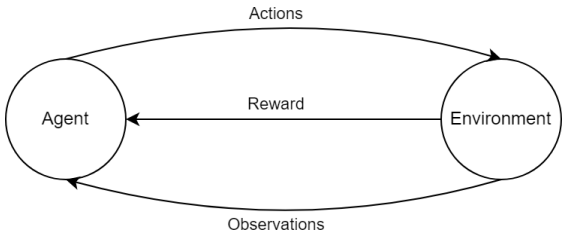

Figure 2.4: Figure showing how the feedback loop works in Deep Reinforcement Learning

2.4.1 On-policy vs Off-policy algorithms

For this study, research is done on two algorithms that are on-policy, A2C and PPO. In addition, an off-policy algorithm, DQN, is included in the study. The on-policy reinforcement algorithms compared to the off-policy algorithms have a high sampling complexity, while the off-policy algorithms are more difficult to optimize (Fakoor et al. 2019). With the help of reinforcement learning techniques, agents can learn a policy that is optimized based on the performance values given by interaction with the environment. Policies are encompassed by two types, on-policy and off-policy. With on-policy algorithms, the agent uses a batch of data for its current policy. However, off-policy algorithms reuse the data that remain from older policies to update the current policy.

2.4.2 Proximal Policy Optimization

The research area of machine learning is in constant flux and introduces new algorithms to address reinforcement learning issues; one of these relatively new algorithms is proximal policy optimization (PPO), whose article was first published in 2017 by a group of researchers at OpenAI (Schulman et al. 2017). PPO is a policy gradient algorithm, which means that it alternates between sampling data from the environment and optimizing an objective function with a stochastic gradient ascent. The algorithm is much simpler to implement compared to other policy gradient algorithms, such as trust region policy optimization.

PPO has the benefit of better sample complexity compared to, for example, A2C (Advantage Actor-Critic), where PPO outperforms in benchmarks in environments such as Atari games, with a good balance of sample complexity and ease of implementation (Schulman et al. 2017).

PPO is trained in an on-policy way by training a stochastic policy, meaning that the algorithm explores through the use of sampled actions dependent on its latest version of what its stochastic policy is (Schulman et al. 2017). How much randomness is introduced to a model when training with the help of the PPO algorithm is directly dependent on what the conditions are initially upon start and how the model is trained. Although initially random, PPO seeks to eliminate randomness progressively per update, so that it is encouraged to make use of rewards that have already been discovered.

The type of PPO used in this study uses a technique called clipping. This is to ensure that, whenever the policy is updated, it does not differ too much from its older policy, which means that the interval for the objective is clipped to a small range (Schulman et al. 2017). That is, the probability ratio is clipped, resulting in removing the incentive for the algorithm to move outside

of the objective goal whenever the objective is worsened, while the clipping is simply ignored if the algorithm instead has an improvement.

The pseudo-code found in Algorithm 1 is taken from and elaborated on in Achiam (2020).

Algorithm 1 PPO-Clip - pseudo code

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_K = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value of function $V_{\phi k}$.
6: Update the policy by maximizing the PPO-Clip Objective:

$$\theta_{k+1} = arg\ \max_{\theta} \frac{1}{|\mathcal{D}_K|T} \sum_{\tau \in \mathcal{D}_K} \sum_{t=0}^{T} min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta k}(a_t|s_t} A^{\pi_{\theta k}}(s_t, a_t), g\big(\epsilon, A^{\pi_{\theta k}}(s_t, a_t)\big) \right)$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = arg\ \min_{\phi} \frac{1}{|\mathcal{D}_K|T} \sum_{\tau \in \mathcal{D}_K} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

typically via some gradient descent algorithm.
8: **end for**

2.4.3 Advantage Actor-Critic

Advantage Actor-Critic (A2C) proposed by OpenAI (Wu et al. 2017) is an asynchronous variant of the Asynchronous Advantage Actor-Critic (A3C) algorithm (Mnih et al. 2016). It combines two types of reinforcement learning algorithms, policy-based and value-based. Like other actor-critic methods, it uses two neural networks, one for the actor and one for the critic. The critic estimates the value function, while the actor updates the policy distribution in the direction suggested by the critic.

The pseudo-code found in Algorithm 2 is taken from and elaborated on in Wang et al. (2018).

Algorithm 2 Advantage actor-critic - pseudo code

// Assume parameter vectors θ
Initialize step counter t $\leftarrow$ 1
Initialize episode counter E $\leftarrow$ 1
repeat
    Reset gradients: d$\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    t_{start} = t
    Get state s_t
    repeat
        Perform a_t according to policy $\pi(a_t|s_t;\theta)$
        Receive reward r_t and new state s_{t+1}
    until terminal s_t or t - t_{start} == t_{max}
    $R = \begin{cases} 0 \text{ for terminal } s_t \\ V(s_t\theta_v) \text{ for non-terminal } s_t \end{cases}$

    for i ϵ { t -1,...,t_start } **do**
        R $\leftarrow r_i + \gamma$R
        Accumulate gradients wrt θ: d$\theta \leftarrow$ dθ + ∇_θ log$\pi(a_i|s_i;\theta)$(R - V $(s_i;\theta_v)$) + $\beta_e\varphi$H($\pi(a_i|s_i\ ;\theta)$ / $\varphi\theta$
        Accumulate gradients wrt θ_v: dθ_v + β_v(R - V $(s_i;\theta_v)$)(φV($s_i\theta_v$) / $\varphi\theta_v$)
    end for
    Perform update of θ using dθ and of θ_v using dθ_v.
    E $\leftarrow$ E + 1
until E $> E_{max}$

2.4.4 Deep Q Network

Deep Q-Learning, which is the algorithm that creates the Deep Q Network, is one of the first real examples of creating a reinforcement learning agent that could solve complex problems that would be challenging for a human. The method was applied to emulated games from the Atari 2600 game console. The state in this environment is 128 RGB (red, green, and blue) over a 210x160 pixels frame. This was preprocessed in grayscale and reduced to 110x84 and cropped out irrelevant parts of the image, resulting in a final 84x84 image (Mnih et al. 2013).

DQN works by using a replay memory that saves state action pairs to approximate the expected return and selects an action based on $\epsilon - greedypolicy$. First, the algorithm is initialized by allocating the memory for the replay buffer, which is something that can be quite resource-demanding on the system's RAM (random access memory); the action value function is also initialized with random weights. Then it selects a random action or the best current action with the information. At first, this is randomized. The action is then executed, and the returned reward, along with the state, is stored in the replay memory. A sample is pulled from the replay memory. This batch is then used to adjust the weights by performing a gradient descent (Mnih et al. 2013). This scenario is very similar to the environment in this experiment, where the Doom agent only receives a frame buffer as its state information.

The pseudocode for DQN are found in Algorithm 3 and based off the pseudocode found in (Mnih et al. 2013)

Algorithm 3 DQN

Initialize replay memory buffer
Initialize action-value function with random weights
for $episode = 1, M$ **do**
    Initialize sequence s.
    for $timestep = 1, T$ **do**
        With probability ϵ select a random action
        otherwise select $max_a Q * (s, a)$
        Perform action a, receive reward and state
        Store transition $(s_t, a_t, r_t, r_{t+1}$ in replay memory
        $Sety_j = \begin{cases} \mathrm{r}_j \text{ for terminal state} \\ \mathrm{r}_j + \gamma max_{a'} Q(s, a) \text{ for non terminal state} \end{cases}$
        Perform gradient descent step
    end for
end for

2.5 Hyperparameter Optimization

When considering hyperparameter optimization. It used to be a cumbersome task and was done by humans, when the complexity of reinforcement learning and the number of parameters were smaller. Then only a few trials needed to be run, and humans were quite efficient at optimizing these. But the complexity of reinforcement learning has grown and the computational power available has increased. It has become more feasible to run a large number of trials to explore and optimize the hyperparameter values used by reinforcement learning models.

Grid search is a comprehensive method to explore all combinations of hyperparameters. This is exhaustive, and thus guarantees the best set of hyperparameters. However, this quickly becomes computationally infeasible with more complex models and environments that demand a significant amount of computational power to simulate for large time steps.

Although a random search is sufficient for several data sets in deep belief networks Bergstra et al. (2011), greedy sequential algorithms such as Tree-structured Parzen Estimators (TPE) were shown to yield a better result.

There are several different methods to reduce the number of parameters explored to find optimal values. Some common techniques are *samplers* and *pruners*. Samplers such as TPE create a more intelligent way of selecting hyperparameters to test on the basis of expected improvement models. There are also samplers like random search or grid search that select new hyperparameters based on chance or in an exhaustive way. Next, a pruner is used to monitor intermediate results in trials to monitor their progress compared to already achieved results and terminate them early.

Akiba et al. (2019) in the original Optuna paper tested the number of increased trials that could be performed using pruners. They found that at the same time and using the same hardware, it was possible to run roughly 35 times more trials with pruning than without pruning.

2.5.1 Optuna

Optuna is a software framework used for automatic tuning and optimization of hyperparameters, designed specifically to help with machine learning (Akiba et al. 2019). Optuna aims to balance the sampling and pruning of algorithms. The framework implements an API (application programming interface) to allow users to construct their own search space for parameters in a dynamic way, and it also allows for simple implementation of strategies when it comes to pruning and searching. This framework works to make it easier for developers and researchers to find the optimal hyperparameter settings for whatever they are working on within the domain of machine learning. By default, Optuna uses a Bayesian optimization algorithm and a tree-structured Parzen estimator as a sampler to find the most optimal hyperparameters.

2.6 VizDoom

VizDoom is a reinforcement learning research platform published in 2016 (Kempka et al. 2016) inspired by the recent success in research on the Atari 2600 (Mnih et al. 2013) platform, using the frame buffer as state information. It was the first first-person shooter environment to rely solely on visual input. The benefit of this approach is the elimination of the need for researchers and developers to create their own features for the agent to learn. Another major advantage of Doom is its lightweight and flexible nature, which requires only 40MB of disk space. Furthermore, Doom is software-rendered, which means that it can be run without a desktop environment inside a remote terminal window.

In 2016 and 2017, a multiplayer death-match competition was held to further evaluate and test how visual-based reinforcement learning is viable in complex environments that require many simultaneous tasks, such as navigating, collecting health and ammunition packs, aiming and avoiding opponents (Wydmuch et al. 2019). The competition concluded that while the bots were competent in tactical decisions, such as aiming and shooting, there were some major issues that a human player would be able to easily exploit. Such an example was the use of hard-coded crouching to exploit the lack of vertical aiming. Furthermore, it was observed that bots were not particularly proficient in strategic decisions such as navigation and positioning.

2.7 Related Works

Several environments exist for evaluating the performance of reinforcement learning algorithms. One of the most popular is the OpenAI Gym (Brockman et al. 2016). Among others, it provides the Atari environment used by one of the first successful reinforcement learning agents to visually learn (Mnih et al. 2013). There is a more recent platform, VizDoom (Kempka et al. 2016), which is based on the Doom video game from 1993. VizDoom provides a complex 3D environment in which the agent has to navigate and deal with enemies in different scenarios.

Since the release of VizDoom, it has been used in several research articles. Primarily, it is used to benchmark different variations of reinforcement learning algorithms. In 2016 and 2017 two competitions were held in which different reinforcement learning agents competed against each other (Wydmuch et al. 2019). The clear winner was the IntelAct bot built on a Direct Future Prediction algorithm, and the DQN-based bot took the second place.

The DQN algorithm, together with A2C and A3C, has been a frequent algorithm used to compare

the different variants of algorithms. For example, both are used in the article (Bakhanova & Makarov 2021), where A3C achieved much better performance. In this article, the researchers used a VizDoom scenario called Defend The Center (Appendix B), where an agent is placed in the middle of a room, with a limited amount of ammunition and health. The goal in this scenario is to survive as long as possible by shooting the enemies that spawn around the agent, as they are trying to kill the agent. The researchers of the paper suggested that future research should look at the Deadly Corridor scenario, which is explained in detail in Section 4.4 of this paper. This scenario, deadly corridor, is used in this study to broaden the comparison of the algorithms chosen.

A comparison of the PPO and DQN algorithms is made in article (Zakharenkov & Makarov 2021), where DQN outperforms PPO. This article raises the question of the importance of hyperparameter optimization due to the fact that the respective hyperparameter values for either algorithm were not disclosed. Furthermore, Green et al. (2019) contradicts this by stating that PPO outperforms DQN. This study further investigates how DQN and PPO perform to each other in a complex 3D environment.

As mentioned in both (Lynnerup et al. 2019) and (Vohra et al. 2022), optimizing hyperparameters is important for reinforcement learning. Newer algorithms such as PPO and Soft Actor-Critic (SAC) are said to be more robust to hyperparameter settings than older algorithms such as Deep Deterministic Policy Gradient (DDPG), which are infamous for being sensitive to hyperparameter settings. To automatically optimize these settings, several different tools are available. Some examples are HOOF, Optuna and Ray, (Paul et al. 2019), (Akiba et al. 2019), (Moritz et al. 2017). For this study, the Optuna framework was used due to its proven efficiency and native support for the Stable-Baselines3 framework (Raffin et al. 2021). Optuna was proved one of the most efficient hyperparameter optimization frameworks in an article by Shekhar et al. (2022).

3 | Problem

This chapter formulates the definition of the problem for the study. It starts with its aim and motivation. Going forward, the research questions, associated hypotheses and objectives performed to answer these research questions can be found.

3.1 Aim

This study aims to evaluate the effects of hyperparameter optimization on reinforcement learning algorithms and to determine how the three algorithms chosen differ from each other in the VizDoom environment.

3.2 Motivation

As stated by Paul et al. (2019) and (Henderson et al. 2018) policy gradient methods, such as PPO, are sensitive to hyperparameter configurations, such as learning rate settings. Despite this, there is a lack of documented guidelines for hyperparameter optimization (Vohra et al. 2022). In general, there is a lack of published hyperparameter settings within the machine learning community that affect reproducibility (Lynnerup et al. 2019). As a result of this general lack of hyperparameters enclosed in the research space, this study looks to show the significance and effects of hyperparameter optimization.

In their paper, (Zakharenkov & Makarov 2021) Deep Q-Learning (DQN) and Proximal Policy Optimization (PPO) were compared in the VizDoom environment. The authors found that the older algorithm, DQN, of the two outperformed the more recent and popular algorithm PPO. This was a surprise in that the research paper presented results that contradict the supposed efficiency and simplicity of PPO, which should in theory be the better algorithm, as stated by Green et al. (2019). Their paper (Bakhanova & Makarov 2021) also states that agent training in this kind of environment is an unstable process, strengthening the need to investigate what hyperparameter to optimize.

Because research is being conducted on these algorithms and in environments like VizDoom, an argument can be made for trying to improve upon the research and explain why one algorithm outperforms another when in reality it is expected not to, due to PPO being more sample efficient than DQN. The research paper by (Zakharenkov & Makarov 2021) lacks a showing of their hyperparameters for their respective algorithm. This is an issue with most of the research

literature that has been looked into during this study. Without any way of knowing the settings used, there is no way to completely reproduce their research.

3.3 Research Questions

RQ1: Is there a significant difference between PPO, DQN, and A2C in terms of accumulated reward when trained for fixed step size in the VizDoom environment for the deadly corridor scenario, with default hyperparameters?

RQ2: Is there a significant difference in terms of accumulated reward for each respective (PPO, DQN, and A2C) algorithm with and without optimized hyperparameters in the VizDoom environment for the deadly corridor scenario?

3.4 Hypotheses

As stated by (Vohra et al. 2022) currently there is no detailed investigation of the effect of hyperparameter optimization within reinforcement learning. The research articles (Zakharenkov & Makarov 2021) and (Shao et al. 2018) evaluate the different algorithms proposed in this study. However, these articles do not disclose hyperparameter settings and use different scenarios and possibly different reward functions. The result of this is that it is hard to draw any general conclusions upon which the proposed null hypothesises could be formulated.

3.4.1 RQ1 Hypotheses

1. Null hypothesis: There is no significant difference in the accumulated reward between PPO and DQN with default hyperparameters.

2. Null hypothesis: There is no significant difference in the accumulated reward between PPO and A2C with default hyperparameters.

3. Null hypothesis: There is no significant difference in the accumulated reward between DQN and A2C with default hyperparameters.

3.4.2 RQ2 Hypotheses

1. Null hypothesis: There is no significant difference in the accumulated reward for PPO when applying optimized hyperparameters compared to applying default hyperparameters.

2. Null hypothesis: There is no significant difference in the accumulated reward for DQN when applying optimized hyperparameters compared to applying default hyperparameters.

3. Null hypothesis: There is no significant difference in the accumulated reward for A2C when applying optimized hyperparameters compared to applying default hyperparameters.

3.5 Objectives

The following objectives aim to answer the research questions stated.

1. Find a suitable 3D environment, suitable reinforcement learning algorithms, and a hyperparameter optimization library. (All).

2. Implement each component.

    - Implement VizDoom (Simon Malm).
    - Create models based on the Proximal Policy Optimization algorithm (Markus Olsson).
    - Create models based on the Advantage Actor-Critic algorithm (Kasper Witt).
    - Create models based on the Deep Q Network algorithm (Simon Malm).
    - Apply the Optuna Library (Markus Olsson).

3. Train and validate the different artificial intelligence agents with and without hyperparameter optimization. Collection of data (Individual).

4. Explore, visualize, and perform statistical tests on the generated data (Kasper Witt).

5. Analyze the processed data (All).

4 | Methodology

4.1 Method

4.1.1 Experiment

In an experiment, one investigates how the independent variables affect the dependent variable. In this study, the aim is to investigate the efficiency of different reinforcement learning algorithms and the effect of hyperparameter optimization on these. The different hyperparameter settings as well as the different algorithms chosen in this study would in an experiment form the independent variables of the study. In this case, the dependent variable would be the total reward gained in an episode performed in the environment. The analogy fits well, as there is a clear cause-and-effect relationship, which supports a controlled experiment as the preferred method for this study. Furthermore; as Berndtsson et al. (2008) states; "Experiments are often done by implementing a model of some system and running simulations to see how the model is affected by different variables", as such, the experiment methodology is well suited for the purpose of the study.

1. **Independent Variables:** PPO, DQN, A2C, Hyperparameters
2. **Dependent Variable:** Reward

A complete list of the different hyperparameters used as independent variables in the experimental part of the study can be found in Chapter 4.8.

4.1.2 Alternative Methods

Case Study

A case study is conducted as an in-depth exploration of the phenomenon in its natural environment (Berndtsson et al. 2008). The major problem with this approach is to find an organization that is actively researching deep reinforcement learning in a more complex environment. Finding such an organization was deemed to take too much time. Furthermore, it would mean that control over the experiment would be lost to some extent; for example, all the chosen algorithms would not be available. For these reasons, this method was considered not suitable for the study.

Survey

A survey means that data are generated by giving different forms of questionnaires to a selected population. An approach to the study using this methodology would be to collect information from people who have experience working with the proposed algorithms and environment. As with the case study methodology, finding these people would be hard and take time. Since the study also aims to answer the questions asked with statistical certainty, this method was considered not suitable for the study.

Systematic Literature Review

A systematic literature review is a literature analysis that investigates a problem by analyzing published sources (Berndtsson et al. 2008). This method was deemed inappropriate because, as (Vohra et al. 2022) states, an evaluation of the effect of hyperparameter optimization for deep reinforcement learning algorithms is lacking in the current literature. Since hyperparameter settings are often omitted from the literature, it would also be difficult to answer the research questions using this methodology.

4.2 Data collection

To collect data from the experiment, various tools were used. Stable baselines use Pytorch by Paszke et al. (2019) and Tensorflow by Abadi et al. (2015) to write models and record training progress on the interactive log interface tensorboard, which can be used to monitor progress during training. Furthermore, the training data were saved in Comma Separated Value (CSV) files for later use in plots and statistical tests. The first step was to create three to five models per algorithm with default hyperparameters and visualize their training progress, the number of models varied due to time constraints. In Figures 5.1, 5.5, and 5.9, these models are visualized. The mean value of the different models is shown in a bold line, and the shaded area is the boundary between the reward that each model earns.

The second step in collecting data was to run the Optuna study for 50-52 trials, the 52 trials due to parallelization and the inability of Optuna to remove trials from a study and choosing the best performing agent; This agent was found by evaluating the model produced by each trial running 10 evaluation episodes of the scenario. The hyperparameters of this agent were then taken and three to five agents were trained using the same hyperparameters. Figures 5.4, 5.8, and 5.12 visualize the training progress of optimized agents.

After creating both default and optimized agents, each model trained for 1 million time steps was evaluated in 50 episodes. Data from this evaluation can be found in Figure 5.13. The data was then used to perform a statistical test using the Mann-Whitney U test in SciPy (Virtanen et al. 2020) to determine the statistical significance of the results and answer the research questions. The p-values of these tests can be found in Tables 5.14 and 5.15.

4.3 Experiment Validation

As opposed to traditional supervised and unsupervised machine learning, reinforcement learning typically does not use validation or test data. Instead, the performance of the trained agent is assessed by how well it performs the task at hand. In the case of this study, a demo script was developed to allow the viewer to see how an agent performs during training.

To answer the research questions with statistical significance, the Mann-Whitney U test was used.

4.3.1 Mann-Whitney U-test

To answer the research questions and answer the proposed hypotheses, statistical tests were performed with $\alpha = 0.05$. The Mann-Whitney U test was considered a good choice for a statistical test based on the research carried out by Colas et al. (2019).

The test was performed in Python using the Scipy (Virtanen et al. 2020) package's *Mann-Whitneyu* function. The test was carried out on a sample of 50 evaluation episodes generated by each model trained for 1 million time steps. The data from these evaluations are then compared with the U-test and the null hypothesis is accepted or rejected based on the resultant p-value.

4.4 Environment

The environment chosen is the deadly corridor scenario; it is considered a representative but simplified version of the challenges faced in a first-person shooter (FPS) game. The scenario contains a narrow corridor that requires one to navigate corners and shoot enemies before the agent is killed. At the end of the corridor there is an armor pick-up, which represents the goal of the agent, that the agent must navigate towards while avoiding enemy fire and taking them down quickly before taking damage or being killed.

The state of the environment in the deadly corridor is represented by default as a frame buffer of 320x240 pixels. Each pixel contains a triplet with values ranging from 0 to 255 representing RGB (red, green, and blue) values. There are several other game variables available as well, but this implementation focuses solely on visual input. However, game variables are used to shape the reward calculation.

This image is gray-scaled and reduced to the minimal frame size available with the built-in tools of VizDoom. This resulted in a frame buffer of size 160x120. Attempts were made in the earlier stages of the experiment by preprocessing the images of the game. However, this would have added computational complexity that was greater than the gain from the reduced pixel count. This meant that every state would has gone from a $3 \cdot 320 \cdot 240 = 230400$ to a $1 \cdot 160 \cdot 120 = 19200$-sized array.

Figure 4.1 is an overview of the deadly corridor scenario. The white dots represent different entities in the scenario. On the far left side is the agent character. On the far right is the health pack that the agent is supposed to take. The six remaining white dots in the middle are enemy entities. Figures 4.2 through 4.5 show images from different stages of the map.

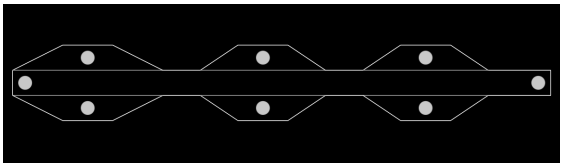

Figure 4.1: Map of deadly corridor scenario from the editor.

Figure 4.2: Starting position for the agent.

Figure 4.3: Agent moving around corner.

Figure 4.4: Agent taking damage.

Figure 4.5: End of the scenario.

4.5 Reward shaping

A major part of the performance of an agent in reinforcement learning is the reward associated with the actions an agent performs. In VizDoom, all scenarios have limited hard-coded reward functionality. For the deadly corridor scenario, this means that the agent receives an increasing reward as it gets closer to the armor, and if it manages to collect the armor, it gets a bonus. Since this in its original state is the only reward function, the agent has a very hard time understanding how to get to the armor, which is the ultimate goal, since the reward function only reinforces the behavior of moving forward.

Due to this limited reward function, an additional reward function was added to the source code for this study. This raises threats to external validity, as the added functionality goes outside the original VizDoom code. The reward functionality is primarily there to guide the agent's behavior; but it adds to the total accumulated reward for the agent, which is used as the data represented in the result section of this study. This makes it difficult to compare the results of this study with other studies carried out in the deadly corridor scenario, because there is no way to compare how the reward shaping was carried out by the other researchers with the same scenario, since their reward could be in the range of 0-1000 instead of 0-1, as in this study.

The complete reward function used in this study can be found in Table 4.1. This is heavily influenced by the work of Mehdi Boubnan (2019). The moving forward reward is the x-delta between the starting position and the armor. The VizDoom reward presented in the table, that is, moving forward, armor pickup, and death, is later divided by 5 to keep that reward relative to the other rewards, which are based on different game variables. The final sums of all the rewards are divided by 1000 to keep the reward close to the [0,1] space to allow a faster and more stable learning process.

Action	+	-
Moving forward	+ Δx (max 1300)	
Pickup armor	+ 1000	
Death		- 100
Enemy killed	+ 100	
Health loss		-5
Ammunition lost		- ammunition_lost * 0.5
Ammunition gained	+ ammunition_gained * 0.5	

Table 4.1: Table of the reward shaping.

4.6 Stable-Baselines3

Stable-Baselines3 (Raffin et al. 2021) is a fork of OpenAI-Baselines (Dhariwal et al. 2017) which is the source implementation of many algorithms used today as it is created and maintained by OpenAI. However, Baselines is no longer actively being developed and lacks many utility features that have been added to Stable-Baselines3. This includes methods that are used heavily in this study, such as function callbacks to save the model at frequent intervals to help with testing and more immediate feedback during the early testing phase such as; logging into the tensorboard and CSV file for creating graphs and data analysis. The complexity of RL algorithms is highly dependent on the specific implementations of an algorithm, which have a greater impact on the

final result than differences between the algorithms themselves (Engstrom et al. 2020). Therefore, this experiment required a single standardized library to compare the different algorithms to minimize implementation differences.

With the complexity of DRL algorithms and the difficulty with which to validate work due to the opaqueness of neural networks, this places even greater importance on the ability to reproduce previous studies. It has been difficult to validate the work (Henderson et al. 2017), and therefore a stable and fully common ground is necessary for the testing and comparing of algorithms. Stable-Baselines3 has a 95% code coverage of tests and aims to be thoroughly documented and easy to understand compared to other libraries (Raffin et al. 2021).

4.7 Choice of Algorithms

The specific algorithms chosen to compare in this experiment have a history of research in deep reinforcement learning. These are commonly used in experiments such as (Mnih et al. 2013) in their Atari experiment, as well as (Schulman et al. 2017) also in Atari and (Zakharenkov & Makarov 2021) VizDoom. These experiments share a common environment structure where the frame buffer of raw pixels is used, so they are relevant as comparisons.

Furthermore, according to the documentation of Stable-Baselines3, there is a limit to the shape of the observation and action space, these being continuous and discrete variables. DQN, PPO, and A2C are the three available algorithms that support continuous and discrete variables. The chosen scenario: Deadly Corridor only requires discrete actions, so the requirement could have been eased. However, to have wider compliance with the VizDoom environment, which supports both types of action spaces, for possible future work, it was decided that both types of action space must be supported, and thus the list of suitable algorithms was only three. There was also the possibility of adding more or other algorithms using the experimental version of Stable-Baselines3. But due to stability concerns and increased complexity, this was decided against.

4.8 Hyperparameters

Hyperparameters are the parameters set for an algorithm before the learning process takes place. Such parameters have a direct effect on how well a model can learn from its environment. To reach maximum potential for learning in an environment, a model must have its hyperparameters optimized, and there are several ways to do this. For this study, Optuna was chosen, which allows one to set an interval between certain values of hyperparameters and, in doing so, automatically find the best ones. Table 4.2 contains all hyperparameters used in the different experiments conducted in this study. Several more hyperparameters are available for each algorithm. The ones chosen are the ones most likely to affect the optimization outcome, mainly based on how they change between different environments in the Zoo documentation 5.15. Appendix A contains all hyperparameter values used in the experiments.

4.8.1 Learning rate

Learning rate is the rate at which the model learns and adapts to a problem or environment. It is the degree to which a new value is accepted versus an old value.

Hyperparameter	PPO	DQN	A2C
Learning rate	X	X	X
Epochs	X		
Entropy coefficient	X		
Gamma	X	X	X
Batch size		X	
Gradient steps		X	
Training frequency		X	
Max grad norm			X
General Advantage Estimate Lambda			X
Nstep			X

Table 4.2: Table of hyperparameters used in the study.

4.8.2 Epochs

Epochs are the number of times an algorithm will go through entire training data sets; that is, through one epoch, the training data set is passed forward and back through a neural network for it to learn.

4.8.3 Entropy coefficient

The entropy coefficient hyperparameter is used for regularization to help improve policy optimization in reinforcement learning. It helps to explore the models by encouraging the selection of stochastic policies (Ahmed et al. 2018). This means that the agent would explore states as it is learning and that the model would also be better equipped to deal with abnormal situations, which would make it more robust.

4.8.4 Gamma

The hyperparameter gamma is used as a discount factor for data points. It controls how single training points influence the algorithms. The more data points that are closer together, the higher the gamma; the opposite is also true, where the fewer data points there are, the lower the value of the gamma. With high gamma values, there is a risk of overfitting, which is why it is important to control that the data point values do not grow too high. Consequently, this dictates how much the algorithm values the future reward.

4.8.5 Batch size

Batch size defines the number of samples that will be propagated through the neural network prior to an update of the internal model parameters used. It impacts how well a model learns and has a direct effect on the stability of a learning process of an agent.

4.8.6 Nstep

The nstep hyperparameter, together with the n_env variable, forms the batch size for certain algorithms in Stable-Baselines3. The n_env variable is the number of environments that run in parallel. For A2C this means; $Nenv \cdot Nstep = batchsize$.

4.8.7 Gradient steps

Gradient steps are used to calculate the error for each iteration of the learning processes of an agent, so that the parameters of the model move as closely as possible to values that result in the minimum cost. The number of gradient steps must be finely optimized so that the training is robust; the smaller the rate of steps, the more reliable the training will be, but at the cost of more time. If the rate is too high, the training may simply not converge or diverge, which means that overshooting would occur. Therefore, a balance must be struck to find the minimum loss as quickly as possible without risking the robustness of the model.

4.8.8 Training frequency

This hyperparameter determines how often the model will be updated, depending on the number of steps of the set. This is used for Deep Q-Learning, where batch training is performed depending on the set training frequency and the parameters of the neural network are updated.

4.8.9 Max grad norm

Max gradient normalization is performed to find the maximum value of gradient clipping. Gradient clipping is when you force gradient values to conform to either a minimum or a maximum value if a gradient value goes above or below the interval of expected values.

4.8.10 General Advantage Estimate Lambda

Lambda is used for smoothing to reduce variance when a model trains, to make model training more stable.

4.9 Hyperparameter Importance

How much any particular hyperparameter affects the learning of a model in any given environment depends on how the individual hyperparameters are used by the algorithms. How much they have an effect on the variance of how a model learns can be quantified by a measure that shows how much they affect the training of a model (Hutter et al. 2014). This is done in this study with the help of the Optuna library, which provides a feature, where the functional analysis of variance (functional ANOVA) is taken into account to show how the hyperparameters of a study are evaluated, allowing for an overview of how important each hyperparameter has been for each Optuna study. The following figures show these 5.3, 5.7, and 5.11.

4.10 Training Hardware and Software

All code is written in Python and is available at `https://github.com/a19simma/BSc-thesis` which includes a complete list of packages used.

5 | Result

This section includes the results. The first three sections are for each individual algorithm. These chapters consist of three sub-chaps, each that presents the findings associated with default hyperparameters, optimized hyperparameters, and comparison between these. In the last section, the results of each algorithm are compared. For both PPO and A2C, the models were trained for 2 million time steps, while for DQN the models were trained for 1 million time steps. This was due to time constraints. In the experiments, the models were continuously saved, which means both A2C and PPO had models that were trained for only 1 million time steps, these were used for the comparison with DQN.

5.1 DQN

This section explores the experimental results of the DQN algorithm. It will present a default baseline result as given in Stable-Baselines3 Raffin et al. (2021) and their implementation of DQN. This implementation, in turn, uses the hyperparameters given by the original paper presenting DQN playing Atari by Mnih et al. (2013).

5.1.1 Default Values

First, a baseline is created for further comparisons. This set of default values is what will be used if no values for the parameters are given. The study uses fixed values of 1,000,000 for $buffer\_size$ and 50,000 for $learning\_starts$. This was changed to 1/40th and 1/20th, respectively, of the total time steps trained. This was done due to RAM constraints, as the replay buffer needs to be stored in RAM, or the simulation slows down by moving it to the virtual RAM on the physical storage. The original study trained in the Atari environment for 40,000,000 time steps (Mnih et al. 2013), while this experiment uses 1,000,000.

Hyperparameter	**Values**
Learning rate	$1 \cdot 10^{-4}$
Batch size	32
Gamma	0.99
Training Frequency (Time steps)	4
Gradient Steps	1

Table 5.1: DQN Default Hyperparameters.

Due to the specific way in which off-policy algorithms are implemented, the logging interval

cannot be set to a unit in time steps but must instead use episodes. Therefore, the horizontal axis uses episodes to ensure that each of the five runs has data points at the same places. However, each run is trained for 1 million time steps. The shadowed area in Figure 5.1 indicates the range of values of the five runs for each data point, the strongest line being the mean. A narrower band means less variance between the different values.

It can be seen that no improvements occur before 2,000 episodes, since this corresponds to the end of the *Learning starts* period, which is spent mapping the transitions with random actions, and no learning occurs.

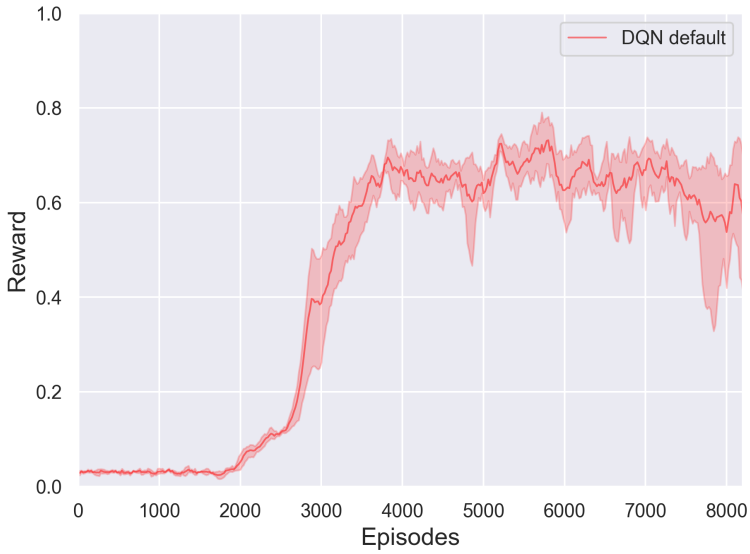

Figure 5.1: DQN Default learning curve.

5.1.2 Optuna Study

The Optuna study was conducted once for a total of 50 trials. Each trial was run for 1 million time steps and used hyperparameters suggested by the TPE (Tree-structured Parzen Estimator) sampler within a defined range. Values such as batch size and training frequency were restricted in their ranges, as they greatly impact training performance at higher values. The following are the ranges established for exploration in the study.

Hyperparameter	**Lower-bound**	**Upper-bound**
Learning rate	$1 \cdot 10^{-5}$	1
Batch size	1	128
Gamma	0	1
Training Frequency (Episodes)	1	10
Gradient Steps	1	10

Table 5.2: DQN Hyperparameter Ranges.

Figure 5.2 shows the progress of the study as it was carried out listing the target values (rewards)

achieved for each trial. Generally, the first values did not achieve high reward values, with a few exceptions. The higher the objective values, the further the study progresses.

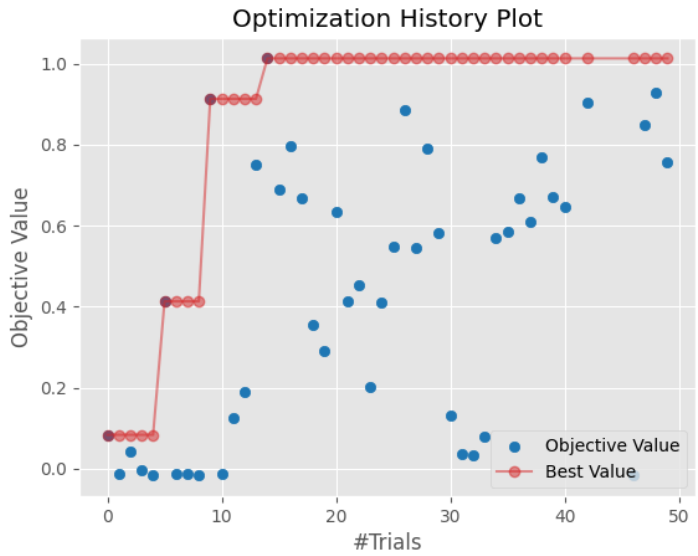

Figure 5.2: DQN Optimization History.

As shown in Figure 5.3, Optuna provides values for how much each hyperparameter contributes to the final objective value of the trial. This value is approximated from the study and is shown as a value between 0 and 1 and the sum of all values being 1.

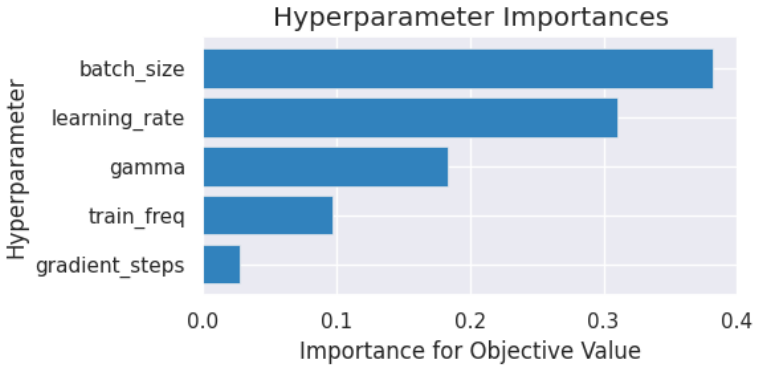

Figure 5.3: DQN Hyperparameter importance.

5.1.3 Result

Hyperparameter	Values
Learning rate	$1.9070714918537712 \cdot 10^{-5}$
Batch size	77
Gamma	0.6371027666566412
Training Frequency (Episodes)	3
Gradient Steps	10

Table 5.3: DQN Optimized Hyperparameters.

The procedure for the default value is repeated, five models are trained with the optimized hyperparameters, and the learning curves are combined. This yields the following plot. It can be observed that the model takes longer to converge to an effective policy, most likely due to the lower learning rate.

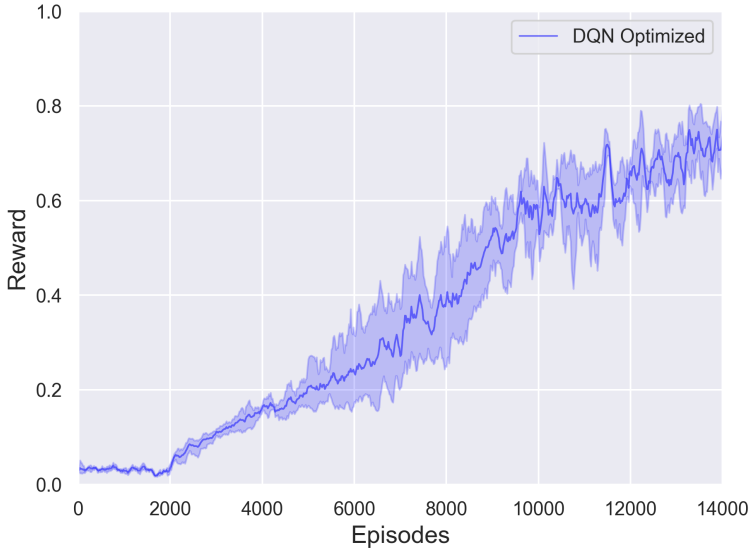

Figure 5.4: DQN Optimized learning curve.

Table 5.4 was created by sampling means from 50 episodes of the respective model. The reported p-value between the default hyperparameters and the optimized model was 0.00047695410128718537 which is less than $\alpha = 0.05$. Therefore, RQ2 $H2_0$ are rejected and it is concluded that there is a significant difference between the two models.

Measurement	Value
Mean Default	0.59396494
Mean Optimized	0.7214449
P-value	0.0004769

Table 5.4: DQN Evaluation results.

As shown in Table 5.4, there is a significant difference between the optimized and baseline values. Optimization increased the sampled mean reward.

$0.7214449/0.593964 = 1.2146 = 21\%$ difference in the mean reward sampled.

This is a pretty good improvement to the policy. The optimal trial was found in trial number 14, which is surprisingly early in the study. The policy also reached the maximum value much later in the training process. The baseline policy reached a stable 0.7 in 150,000 time steps, which in contrast was not surpassed by the optimized policy until the very end.

5.2 PPO

This section delves into the use of the Proximal Policy Optimization algorithm in the VizDoom environment in the deadly corridor scenario (Schulman et al. 2017). It will serve to present the results generated from the use of both optimized and default hyperparameters. The default hyperparameters are the values given by the implementation of the Stable-Baselines3 library by Raffin et al. (2021). Optimized hyperparameters are found using the Optuna library (Akiba et al. 2019).

5.2.1 Default Values

The default hyperparameter values for the PPO algorithm were derived from the implementation of the Stable-Baselines3 library, without adding changes to the hyperparameter settings Raffin et al. (2021). With those hyperparameters, there were three separate runs of the algorithm in the environment of VizDoom, a total of 2 million time steps for each run, to produce results that could reflect how well the algorithm works for a model without the help of hyperparameter optimization. It can be observed that even without optimization, the PPO algorithm is quick to learn the environment and is already able to achieve a relatively high reward around roughly 100,000 time steps.

Hyperparameter	**Values**
Learning rate	$1 \cdot 10^{-2}$
Epochs	10
Gamma	0.99
Entropy coefficient	0.0

Table 5.5: PPO Default Hyperparameters.

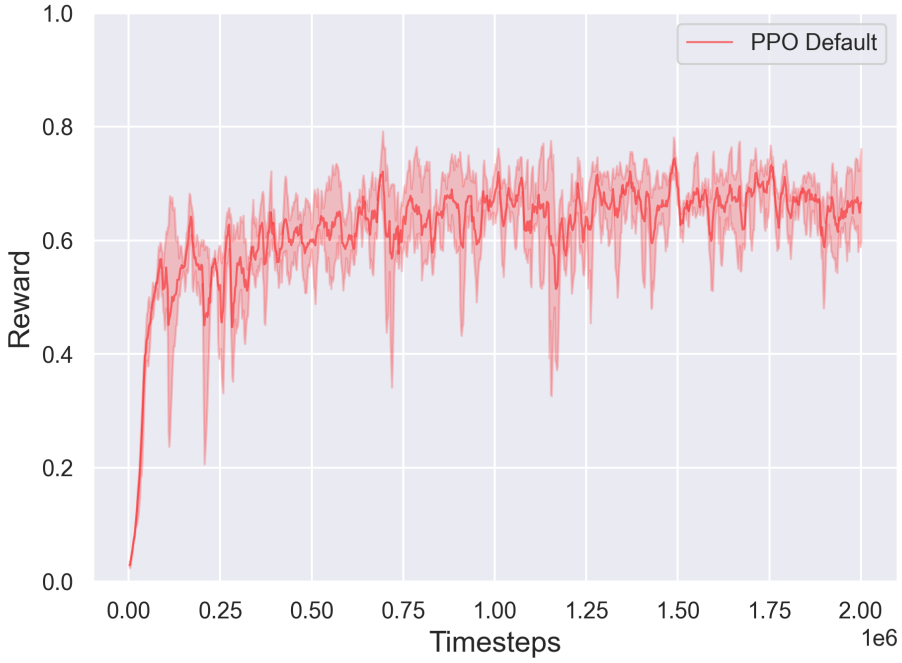

Figure 5.5: PPO Default learning curve.

5.2.2 Optuna Study

With the help of the Optuna library, 50 trials were completed with a total of 2 million time steps for each trial to allow the optimization library to find the best hyperparameters dependent on an interval of values for four particular hyperparameters. These are the learning rate, epochs, gamma, and the entropy coefficient.

Hyperparameter	**Lower-bound**	**Upper-bound**
Learning rate	$1 \cdot 10^{-4}$	$1 \cdot 10^{-2}$
Gamma	0.90	0.99
Entropy coefficient	$1 \cdot 10^{-8}$	$1 \cdot 10^{-1}$
Epochs	1	48

Table 5.6: PPO Hyperparameter Ranges.

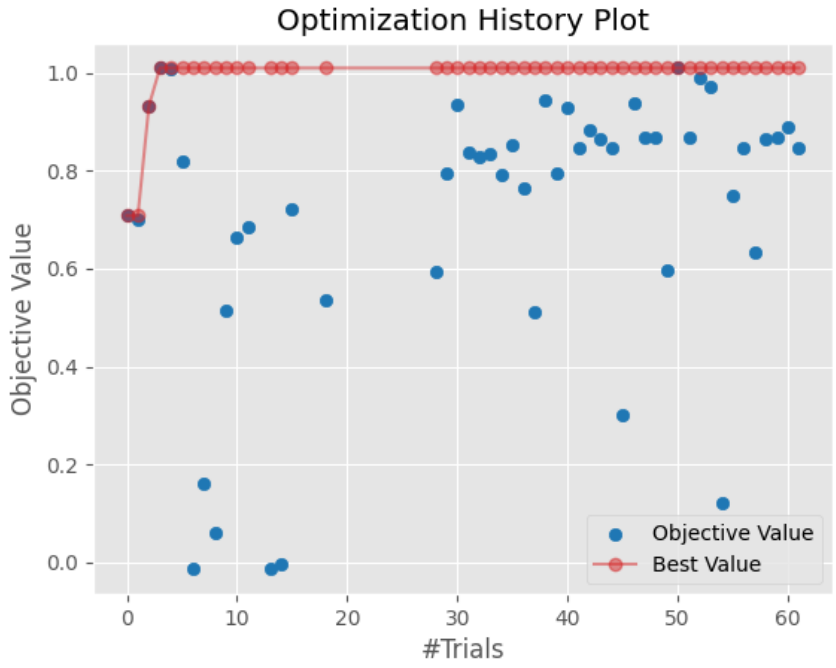

Figure 5.6: PPO Optimization History.

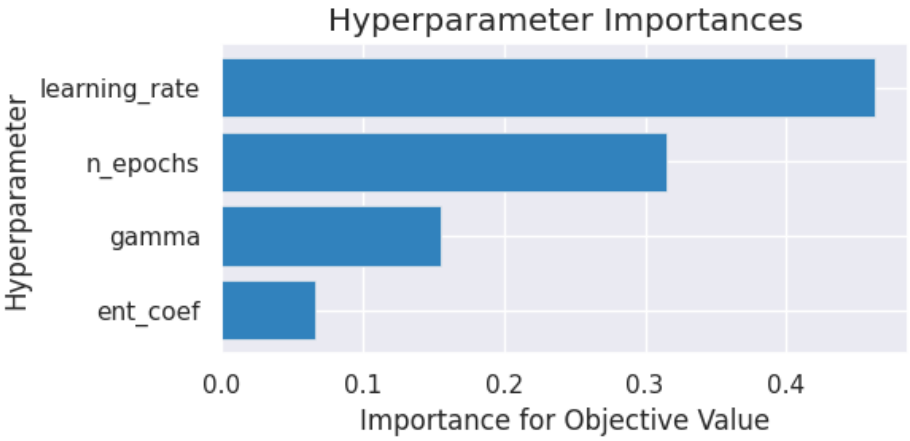

Figure 5.7: PPO Hyperparameter Importance.

5.2.3 Result

With the optimized hyperparameters of the 50 trials run with the help of Optuna, the procedure is repeated again with three training runs for a total of 2 million time steps to be compared directly with the graph data generated from the three models with default hyperparameters. It is clear from the very beginning that optimized parameters help the model learn with the PPO algorithm faster than without optimization.

Hyperparameter	**Values**
Learning rate	$2.59251 \cdot 10^{-4}$
Epochs	3.25784
Gamma	0.908591
Entropy Coefficient	0.00105699

Table 5.7: PPO Optimized Hyperparameters.

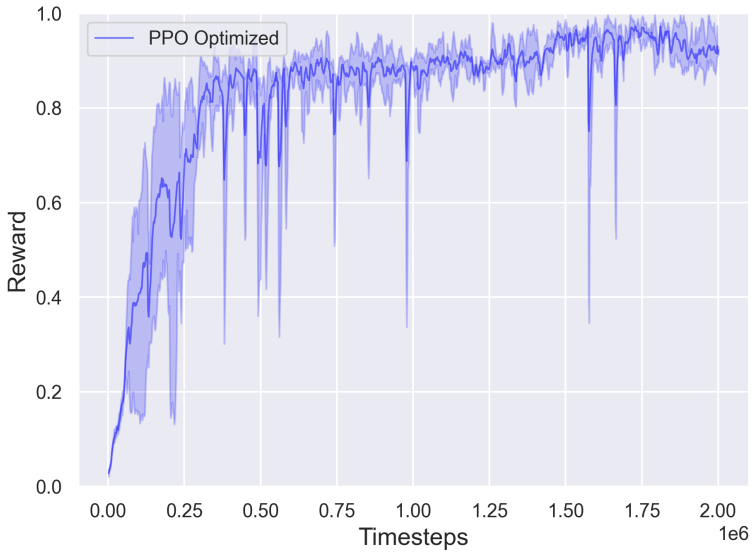

Figure 5.8: PPO Optimized learning curve.

The resulting P-value generated by comparing models with default and optimized hyperparameters with the PPO algorithm is $8.93418280372436 \cdot 10^{-15}$, which is less than $\alpha = 0.05$. This means that the null hypothesis is rejected for RQ_2, and there is a statistically significant difference between the two PPO models with default and optimized hyperparameters.

There is a stark difference between letting the algorithm train in the VizDoom environment with the default hyperparameters in contrast to the Optuna-optimized hyperparameters. Comparing the graph with the default values in Figure 5.5 against that of the graph with the optimized values of Figure 5.8. The agent reaches a much higher reward as they progress through the time steps, but also results in a much higher total reward comparatively.

The default hyperparameters achieve a reward mean of 0.70792408 while the optimized hyperparameters manage to reach a reward mean of 0.98212786, meaning that with the help of hy-

perparameter optimization there has been a significant difference in the result of the generated reward. For the reward mean, the total increase is 0.98212786/0.70792408 = 1.38733500914, which means that there has roughly been a 39% difference from the default hyperparameters to the optimized hyperparameters. The mean for the default hyperparameter agents is calculated by taking the final reward value they achieved, then adding the values together and dividing by the number of agents trained, that is, three. The same is done for agents that use optimized hyperparameters. Then finally, the mean reward achieved with the optimized hyperparameters is divided by the mean reward achieved by the default hyperparameters to see how much they differed.

Measurement	**Value**
Mean Default	0.70792408
Mean Optimized	0.9821278600000001
P-value	$8.93418280372436 \cdot 10^{-15}$

Table 5.8: PPO Evaluation results

5.3 A2C

In this section, the different results of the A2C experiments are shown. The section is divided into one default section, one Optuna section with optimized hyperparameters, and one comparison section named result.

5.3.1 Default Values

In this section are the different results from running an experiment with default hyperparameters. The default values for the hyperparameters used and changed later in the Optuna study can be found in Table 5.9. These default values are derived from the Stable-Baselines3 project (Raffin et al. 2021). With these hyperparameters, three A2C models were trained for 2 million time steps in the deadly corridor scenario. The timeline of agents trained with the default hyperparameters can be found in Figure 5.9. One can quickly observe, in contrast to the results of PPO and DQN, that A2C is unable to learn any good strategy.

Hyperparameter	**Value**
Learning rate	0.0007
Lambda	1.0
Gamma	0.99
Nstep	5
Max grad norm	0.5

Table 5.9: A2C Default Hyperparameters.

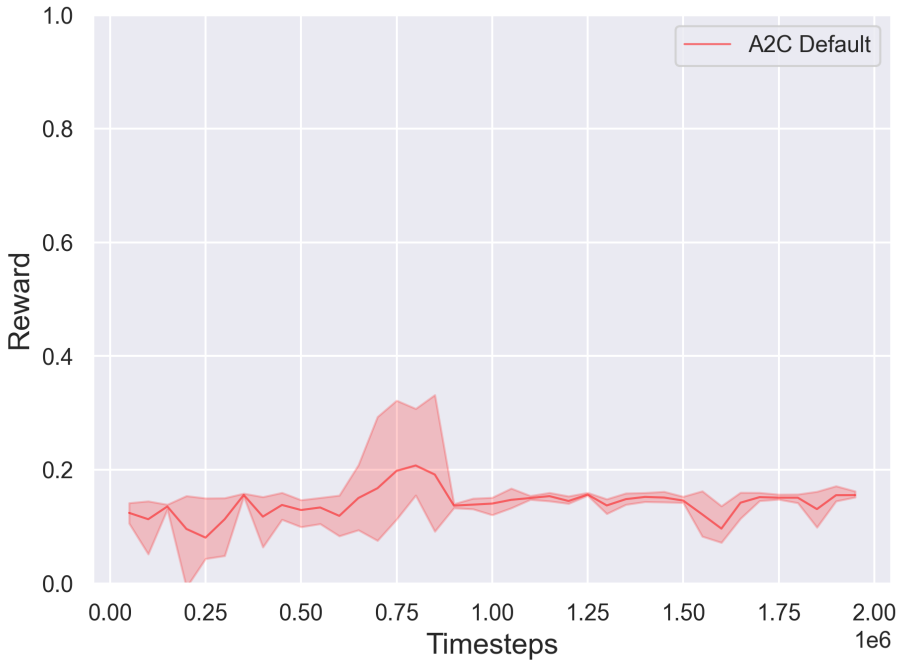

Figure 5.9: A2C Default learning curve.

5.3.2 Optuna study

In this section, the results of an experiment with the Optuna framework using the A2C algorithm are presented. The Optuna study consisted of 52 different trials, that is, each trial equals one model trained for 2 million time steps. In Figure 5.10 is an optimization history for the associated Optuna study. On the X-axis are the trial number, that is, for example, trial number 6 or 39, ranging up to 58 due to aborted trials. Here, there are a total of 52 completed trials and eight aborted ones, which are not shown in the diagram. On the Y-axis is the reward gained by the associated agent after the training is complete. The best trial was number 39; The hyperparameters of this trial can be found in Table 5.11.

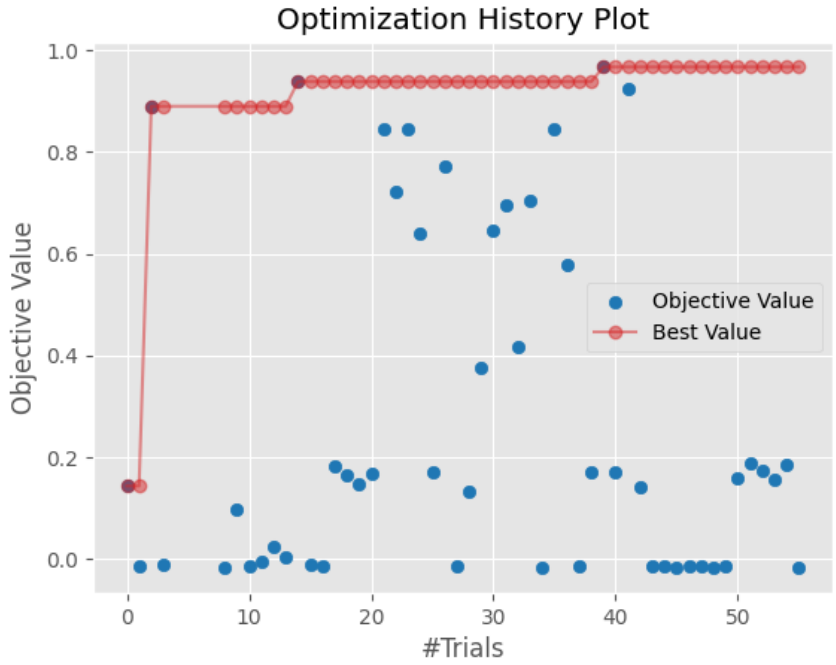

Figure 5.10: A2C Optimization History.

Each trial consists of a different mix of hyperparameters suggested by the Optuna engine. This combination of hyperparameters is taken from the ranges found in Table 5.10. For the learning rate, the engine suggests a float between the stated intervals. For the four other hyperparameters, the engine suggests a float or integer from each respective array of values. The ranges are taken from the Zoo project (Raffin 2020), which is a framework for Stable-Baselines3 that provides optimized hyperparameters for popular reinforcement learning environments, with VizDoom excluded.

In Figure 5.11 is the relative importance of each of the hyperparameters under investigation. These relative values are calculated using the FANOVA tool (Hutter et al. 2014). Similarly to PPO, the learning rate is the most important hyperparameter.

Using the optimized hyperparameters found by Optuna, three models were trained for 2 million

Hyperparameter	Lower-bound	Upper-bound	Categorical
Learning rate	$1 \cdot 10^{-5}$	1	
Gamma			[0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999]
Max Grad Norm			[0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 5]
Lambda			[0.8, 0.9, 0.92, 0.95, 0.98, 0.99, 1.0]
Nstep			[8, 16, 32, 64, 128, 256, 512, 1024, 2048]

Table 5.10: A2C Hyperparameter Ranges.

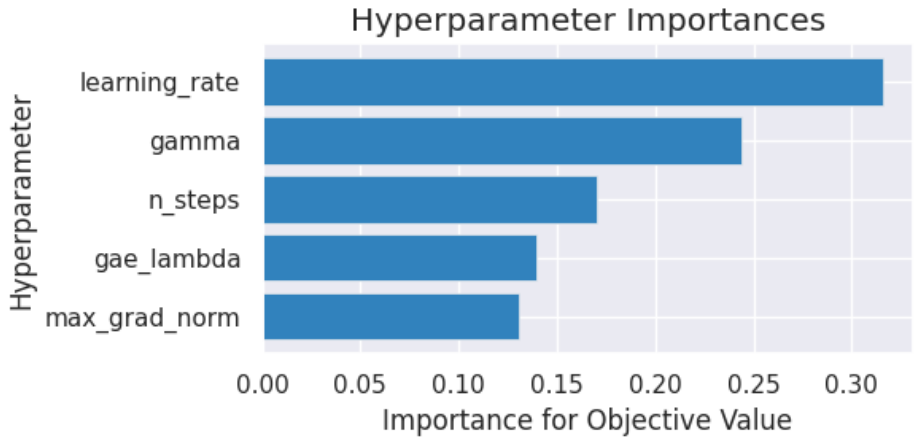

Figure 5.11: A2C Hyperparameter Importance.

time steps. The learning curve of these models can be found in Figure 5.12. In this diagram, the bold line represents the mean value between the three models. The shadowed area represents the variance between the models at each time step. Unlike the default learning curve found in Figure 5.9, it can be quickly seen that the agent has now learned a good strategy for the scenario, with a reward gained similar to that of PPO and DQN.

5.3.3 Result

A comparison of models trained with optimized and default hyperparameters was made. The results of this comparison consisting of 50 evaluation episodes for each agent can be found in Table 5.12.

Hyperparameter	Value
Learning rate	0.0007386
Lamda	0.98
Gamma	0.99
Nstep	256
Max grad norm	0.6

Table 5.11: A2C Optimized Hyperparameters.

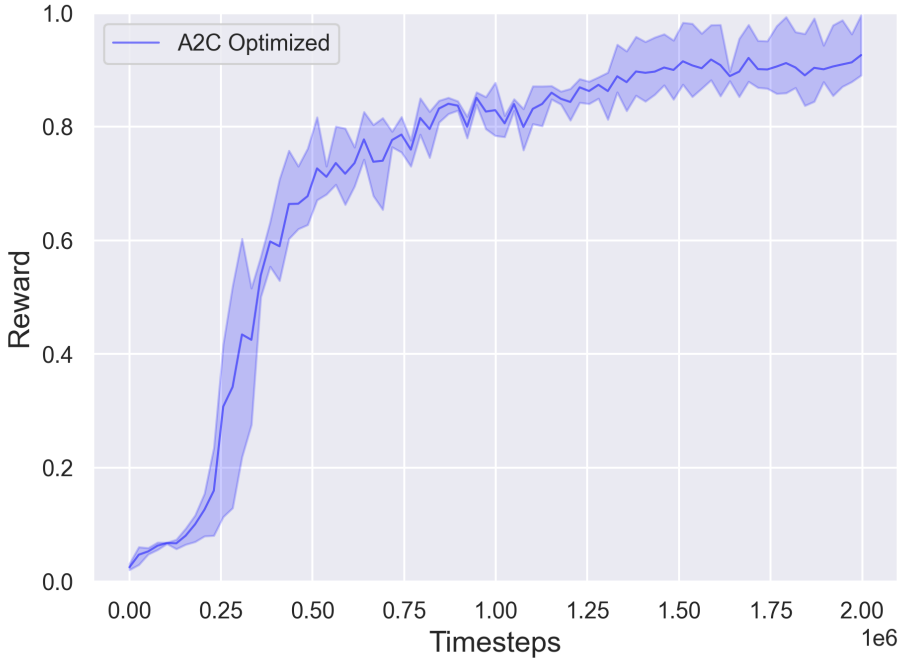

Figure 5.12: A2C Optimized learning curve.

Measurement	Value
Mean Default	0.15760
Mean Optimized	0.82636
P-value	$4.92 * 10^{-18}$

Table 5.12: A2C Evaluation Results.

5.4 Algorithm Comparison and Research Questions

5.4.1 Diagrams

To compare the different algorithms, each algorithm produced two models trained for 1 million time steps each in the deadly corridor scenario. One model with optimized hyperparameters and one without. These models were then evaluated in 50 episodes of the deadly corridor scenario. A box plot of these episodes for all models is shown in Figure 5.13, which compares the performance of all algorithms with and without hyperparameter optimization. In Table 5.13 the mean of the recorded episodes is found. Finally, the recorded episodes are used to statistically compare the performance of the different algorithms with each other without hyperparameter optimization and to themselves with and without hyperparameter optimization. The p-values of this statistical comparison performed using six different Mann-Whitney U tests are found in Table 5.14 and Table 5.15.

Algorithm	Default	Optimized
DQN	0.59396494	0.7214449
PPO	0.70792408	0.9821278
A2C	0.15760952	0.8263691

Table 5.13: Default evaluation means.

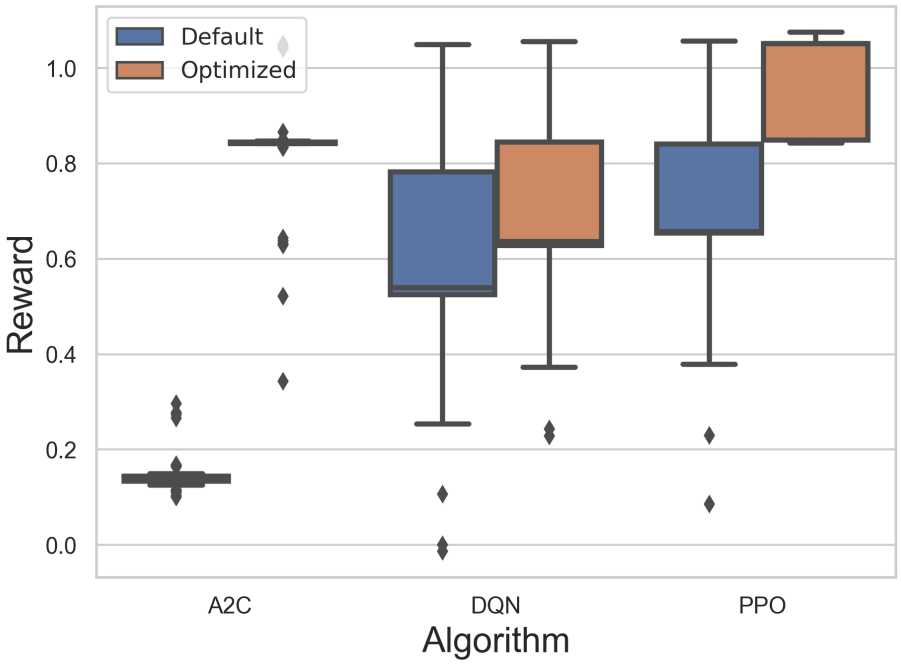

Figure 5.13: Algorithm comparison.

Hypotheses	**Algorithm 1**	**Algorithm 2**	**P-Value**
$RQ1H1$	PPO	DQN	0.000144756657222
$RQ1H2$	PPO	A2C	$5.68 * 10^{-18}$
$RQ1H3$	DQN	A2C	$7.15 * 10^{-18}$

Table 5.14: RQ1: P-values between algorithms.

Hypotheses	**Algorithm**	**P-Value**
$RQ2H1$	PPO	$8.93 * 10^{-15}$
$RQ2H2$	DQN	0.0004769
$RQ2H3$	A2C	$4.92 * 10^{-18}$

Table 5.15: RQ2: P-values for algorithms with and without hyperparameter optimization.

5.4.2 Conclusions

The first research question is the following.

> **RQ1:** Is there a significant difference between PPO, DQN, and A2C in terms of accumulated reward when trained for fixed step size in the VizDoom environment for the deadly corridor scenario, with default hyperparameters?

This question was answered by comparing, two at a time, the models of each algorithm trained with their default hyperparameters. This was done with the Mann-Whitney U test and the results are found in Table 5.14. Each test found a significant difference; therefore, all null hypotheses associated with RQ1 are rejected.

The second research question is the following.

> **RQ2:** Is there a significant difference in terms of accumulated reward for each respective (PPO, DQN, and A2C) algorithm with and without optimized hyperparameters in the VizDoom environment for the deadly corridor scenario?

This question was answered by comparing, two at a time, the models of each algorithm trained with and without optimized hyperparameters. The results of these tests are found in Table 5.15. Each test found a significant difference between the respective models. Due to this, all null hypotheses associated with RQ2 are rejected.

A diagram visualizing the performance of all trained models can be found in Figure 5.13. In addition to not being able to learn a successful strategy with default hyperparameters, A2C also shows a very low variance in its behavior with and without hyperparameter optimization. For the 50 episodes with optimized hyperparameters, 40 episodes were in the reward interval of 0.83 to 0.85. This low variance is well represented in both Figure 5.13 and Figure 5.12.

6 | Discussion

In this chapter, the different results will be discussed and a general discussion and generalization of the result will be made.

6.1 DQN

The results of using optimized hyperparameters for DQN, in ViZDoom's deadly-corridor scenario, were positive by a conclusive margin. This is in line with the expectation that hyperparameters are important for the result. Because of this, the conclusion is that optimizing for each scenario is important.

The study found the best set of hyperparameters early in the process, in 14/50 trials. Initially, the objective value was expected to gradually increase throughout the study rather than such a high objective value at the beginning of the study. Looking at Figure 5.2, this trend is generally true, as higher values are found later in the study with optimal trial 14 which appears to be an outlier.

Further evidence of this is the discrepancy in the objective value of Optuna, which being markedly higher than what was discovered during the sampling of means for the statistical test. The sampled mean was 0.77 and the reported objective value was 1.03. This indicates that the evaluation function that was implemented for the Optuna objective value did not sample enough data points for a reliable mean. Ten samples were drawn in the mean objective value of Optuna and 50 for the statistical test.

Some of the distinctive features of the set of hyperparameters found were the lower learning rate, which made learning slower to converge, and perhaps it could have reached higher values if more training time had been allowed.

Another interesting observation is that the number of gradient steps was on the upper boundary; this indicates that the variable's range should be expanded, and perhaps a faster convergence could have been achieved if the algorithm had performed more gradient steps at each training frequency interval.

6.2 PPO

Comparatively, between the default values and the optimized hyperparameters, there was a clear difference between the different runs that shows and highlights the importance of performing hyperparameter optimization when training models with the help of proximal policy optimization.

The best hyperparameters were found early in the third trial out of the fifty that were completed; this does not necessarily mean that future optimization done with the help of Optuna will result in the best hyperparameters early in the study, but that the possibility can not be ruled out. The more trials that are run for a study, the better the hyperparameters will eventually be. This is due to the use of the Bayesian optimization algorithm by Optuna, which explores the best regions to find the most optimal hyperparameters.

Not all potential hyperparameters were used when optimizing with Optuna and would need to be explored further. For example, the best value found for the gamma hyperparameter nearly bottomed out at the lower bound (Appendix C), and thus an even lower value should be explored in future work for the sake of thoroughness. Optuna does well in finding trials with hyperparameters that result in a good objective value for later trials, as seen in Figure 5.6, which end up mainly in a range between 0.8 and 1.0 for the objective value, with only a few outlying trials that result in a low objective value.

6.3 A2C

When comparing the optimized, Figure 5.12, and default, Figure 5.9, results of the A2C algorithm, there is a huge difference in the reward obtained. One big difference in the hyperparameters used in both experiments, found in Tables 5.9 and 5.11, is the nstep parameter. In the default settings, this parameter is set to 5 while the optimized value is 256. Furthermore; if one look at the optimized hyperparameters found in the Stable-Baselines3 Zoo project (Raffin 2020), the values for nstep are more or less 8 for all available environments. The big difference in Zoo is that the suggested settings includes an N_envs settings of 4-32, resulting in an effective nstep (batch size) of 32-256. In the experiments in this study a n_envs value of one was used. Looking at the logs of the A2C Optuna study in the experiment; no successful trial had an Nstep value below 32. This would suggest that, for a more complex environment, such as the deadly corridor scenario, the nstep, and as a result, the batch size needs to be higher. This is supported by the findings in the supplementary material of the research by Jin et al. (2017), where they found that nstep was the most important hyperparameter to get their A2C agent to converge in the scenario of my way home (Appendix B). Their optimal value for nstep was 40.

In their research Henderson et al. (2018), they conclude that for A2C and PPO, learning rate optimizers have a very narrow window of effective learning rates. Looking at Figure 5.11, the Optuna FANOVA tool suggests that the learning rate is the most important hyperparameter of the study. Looking at the study logs (Appendix C), one can see that all trials, except for two, out of 17, with a learning rate greater than 0.003, resulted in a negative reward after 2 million time steps. Due to these limited findings, it could be suggested that the property of Henderson et al. (2018) also holds for A2C when using a Bayesian optimization method to find an effective learning rate. A similar pattern can be seen for the gamma hyperparameter, where no trial with a gamma value lower than 0.99 succeeded in learning a successful strategy. The window of successful learning rates and other hyperparameters in this study could act as a guide to narrow

the search range of learning rates found in Figure 5.10, resulting in a faster finding of successful hyperparameters.

When comparing the default and optimized hyperparameters found in Tables 5.9 and 5.11, the values except for Nstep are more or less identical. The conclusion drawn from this is that the default hyperparameters suggested by Raffin et al. (2021) for the A2C algorithm are well optimized for general environments. As discussed, one has to be aware of the interaction between the nstep variable and the degree of parallelization one use through the n_env variable, and how this affects the total batch size while training an agent.

6.4 General Discussion

When comparing the resulting differences between the gain in optimizing PPO and DQN, they resulted in an improvement of 38% and 21%, respectively, by optimizing their hyperparameters. This is in contrast to the quality stated by Schulman et al. (2017) that PPO is less affected by hyperparameters and therefore easier to use with its baseline hyperparameters. The achievement of a better result than DQN by PPO was unexpected due to the Zakharenkov & Makarov (2021) result; however, they discuss that DQN appears to have been able to adapt to a more complex scenario in their maze experiment. It suggests that either they were able to optimize the hyperparameters of DQN more efficiently or that the Deadly corridor, which was used in this experiment, is not as complex. The study carried out in this thesis supports the claim of Green et al. (2019), that PPO is more efficient than DQN in contrast to the findings of Zakharenkov & Makarov (2021). In the work of Zakharenkov & Makarov (2021), there are no hyperparameter settings or details of the implementation of the algorithms themselves. As mentioned in the work of Engstrom et al. (2020), small details of implementation have a great impact on reinforcement learning algorithms. Because of this, no general conclusions can be drawn as to why the findings of this thesis and Zakharenkov & Makarov (2021) are conflicting.

For all three algorithms, there was an early trial that achieved a solid objective value. In the PPO study, the third trial was the best. In the DQN study, the 14th result achieved the higher objective value. What these results could suggest is that the TPE sampler is effective in suggesting meaningful hyperparameters, even with a small number of trials. Furthermore, it could be possible that the result was due to an outlier value in the evaluation function, as previously described in the difference between the Optuna objective value and the later tested mean value. This is due to a small evaluation sample that needs to be expanded and was a flaw in the design of the experiment; reliability should have been tested before. Value 10 was taken from a previous example code. It can be seen that PPO and A2C did not suffer from this kind of discrepancy. DQN had trials later in the study that had high values (around 0.9), which, if correct, could have changed the conclusion of the algorithm comparisons. Similarly, A2C found its best values in trial 39.

The optimal hyperparameters found for DQN was a significantly lower learning rate, which resulted in the time it took to converge to an effective policy within the 1 million time steps that occur quite late in the learning curve. This contrasts with PPO and A2C, which found that learning rates were similar to baseline values and converged quickly to an effective policy during the training period. But it could indicate that the scenario was suited to learning slower to have stable learning instead of higher learning rates. It is unclear whether this is the product of DQN being more sensitive, not featuring a clip function, or other similar stability measures that PPO

and A2C have. This could explain the possible difference in the learning rate found.

By a clear margin, PPO was the most effective of the three algorithms with and without hyperparameter optimization.

6.5 Future Work

For future work, the constraint on the number of time steps set for the trials could be greater to allow all algorithms the time they need to train properly. In the original Atari paper (Mnih et al. 2013) the agents were trained for 40 million time steps. It could be interesting to see if, for example, A2C were able to learn an effective strategy with default hyperparameters or if any of the three algorithms would find a whole new strategy or eventually find any bugs. Furthermore, it would be interesting to see how A2C would work with a higher degree of parallelization.

Although only strictly on-policy or off-policy algorithms were used to carry out the experiments, future work could seek to make use of ease-of-use on-policy algorithms combined with the efficiency of sampling that off-policy algorithms bring. This could potentially have been a more interesting way to approach the problem with an algorithm (Fakoor et al. 2019). Learning a complex environment such as Doom by combining off-policy and on-policy traits would prove quite challenging and interesting; however, such algorithms are very challenging and would require more expertise and require more time than a study such as this allows (Henderson et al. 2017).

Further studies in this area should also implement more complex environments for agents to learn, and even the creation of new scenarios would bring new and interesting challenges for future research. Creating a more varied and complex suite of scenarios such as Mnih et al. (2013) and Schulman et al. (2017) in their experiments would make it easier to create generalizable agents for ViZDoom.

The use of pruners was very effective when demonstrated by Akiba et al. (2019) in the original Optuna paper. Using pruners when optimizing hyperparameters might result in faster optimization. As our study used mostly default settings for the packages used, this feature was not included. However, it would be interesting to see more results on the use of pruners in optimizing hyperparameters.

6.6 Threats to Validity

6.6.1 Conclusion Validity

According to Wohlin et al. (2012), threats to conclusions are considerations that can hinder the ability to draw the correct conclusions between treatment and the result of an experiment.

As in previous research in the field of reinforcement learning, there was a clear lack of presentation when it comes to the hyperparameters used and the method used to acquire them. As Bergstra et al. (2011) explains, it is difficult to reproduce the findings of the studies due to the difficulty in optimizing machine learning models. Experimenters may not explore values for all hyperparameters, but may instead assign them some reasonable default. It usually makes the process of even the original method more of an art than a science (Bergstra et al. 2011). This

is due to the known problem of the complexity of machine learning models and their associated hyperparameters. Great efforts have been made to use open source, and recent efforts have been made to standardize the process, such as Optuna (Akiba et al. 2019) and publishing all the source code used on the code repository website GitHub. Furthermore, the study was designed with a reasonable but large number of trials (50) to ensure that the results are reliable. The code for the implementation of the environment, such as defined step functions and reward shaping, was the same. The library implementing the algorithms used was the same. Although this does not necessarily guarantee that the quality of the implementations is the same, this could affect the ability to compare results between algorithms.

Using the Mann-Whitney U test was shown according to Colas et al. (2019) to be one of the statistically stronger methods to compare reinforcement learning policies. Therefore, the results should not be likely to yield false results.

6.6.2 Internal Validity

Internal validity refers to the extent to which the cause-effect construct holds. This could be threatened by factors that are not under the control of the researcher or knowledge when experiments are carried out (Wohlin et al. 2012).

A discovered issue was the discrepancy between Optuna's reported objective value for a trial and the actual training data collected. The tensorboard logs sometimes showed quite different values compared to the Optuna objective value. This suggests that the evaluation method used, of 10 evaluation episodes, may not have been large enough to produce a reliable average reward value. There could also be a difference in how the tensorboard calculates the mean reward per episode batch. The effect of this could be that Optuna gave a faulty weight to some hyperparameter values, and as a result of this, focused on the wrong values, making it slower.

6.6.3 Construct Validity

Construct validity deals with potential design threats when it comes to the design of the experiment and how it can mirror the construct it is meant to portray (Wohlin et al. 2012).

One potential risk for the experiment lies in the coding proficiency of the authors of this study; this is because, although the authors have a fair bit of programming experience, this was the group's first dive into a deeper machine learning project where the language was Python, which none of the participants were especially familiar with. This has the added risk of the results of the experiment not being as optimized as they potentially could have been by someone else with more experience in the field. Therefore, this means that there is a potential validity threat with respect to confounding constructs, because the absence of programming experience may not explain the results generated from the study (Wohlin et al. 2012). Another threat to construct validity is the use of algorithms implemented. The small details of code optimization matter a lot, as stated by Henderson et al. (2017). This makes it difficult to make fair comparisons with other studies done. There is also the risk of different treatments, although Optuna was used to optimize the hyperparameters, the different algorithms used different hyperparameters. In addition, the importance of individual hyperparameters differs between the varying algorithms.

6.6.4 External Validity

External validity is related to the ability to generalize a population by drawing a sample. Through its use, the aim is to find and approximate the truth by inferences, conclusions, or propositions. This means that threats to external validity are things that limit the ability to generalize results to practice (Wohlin et al. 2012).

An issue with the results of the conducted study is that the results are not necessarily as generally applicable as one would like them to be. Only one environment and one scenario were used to test the different algorithms. To reduce the risk of external validity and allow the results to be more generalizable, more scenarios and environments should be studied. But for the sake of comparing all algorithms in a closed and complex environment such as the deadly corridor used in the VizDoom environment, there are some general conclusions that can be drawn from the results.

Within VizDoom, there are several simple scenarios, as described in the Appendix B. The purpose of these is to teach the agents what actions are beneficial. These scenarios could be combined in a form of curriculum learning where the agent is taught the beneficial actions of these scenarios step by step. Instead, this experiment used what was deemed the most complex scenario; Deadly Corridor. This scenario features navigation where the agent is moving in a corridor with enemies behind corners that it has to shoot, and a final pick-up at the end in order for it to complete the scenario. When comparing this scenario to a typical Doom map or a death-match scenario (Appendix B), it lacks some complexity, such as health and weapon pick-ups, moving enemies, and a more complex map to navigate. This lack of complexity would make it difficult to generalize the results to a more complex, typical Doom scenario. However, the basics of shooting enemies and moving around corners in a corridor should cover most of the simple actions needed.

Moving beyond results in VizDoom, the use of the frame buffer and some choice game variables such as ammunition and health as the state information makes the results not dependent on game-specific features such as hitboxes or object location coordinates. Most games will have a screen output and some user interface that displays information. Therefore, the results should be generalizable to shooting games.

6.7 Societal aspects

Training Machine Learning algorithms is a very computationally expensive endeavor, and the ability to increase the efficiency of optimizing the performance of your algorithm is an important problem. Both in terms of the amount of energy and top-of-the-line hardware typically required for today's ML models. A recent example is GPT-3 which is OpenAI's recent language model, which took roughly 50 petaflops/second - days of computing (Brown et al. 2020). A petaflop is 10^{15} floating point operations. The best Nvidia card, designed for ML, costing around $32,000 per piece, has a compute of 0.623 petaflops / second. This means that 80 units would need to run for 50 days to create a project like GPT-3. This was a 300,000x increase in computational demand from the early models in 2012 and is projected to increase in the following years (Amodei & Hernandez 2018). Although this may not have exactly come true since the prediction in 2018, this is a problem in the pipeline of machine learning research and product development.

The purpose of this study is to provide some general guidance on developing reinforcement learning models and to provide general answers to questions such as "Which algorithm should be used

for a reinforcement learning problem?" and "How important is hyperparameter optimization for my model?". These are important issues to address in development pipelines for creating products using reinforcement learning techniques and machine learning in general. As shown in the GPT-3 example, an enormous amount of time can be spent training and testing different algorithms and different combinations of hyperparameters. Our results show that choosing an effective algorithm for the task is important. Furthermore, the results of this study show guidelines for how to narrow the search space for optimal hyperparameters, further reducing the cost of developing a successful agent.

The results of the DQN and PPO experiments suggest that perhaps a smaller number of trials is sufficient, as they found their optimal values within the first 20 trials. This could help reduce the computational load on evaluating hyperparameters for these algorithms by designing smaller studies when evaluating hyperparameters. This could reduce the time to market for products using reinforcement learning models, save hardware costs, and reduce energy consumption.

6.8 Ethics

Artificial intelligence and machine learning are hot topics today. Research is currently investigating the use of reinforcement learning for drones Muñoz et al. (2019). The study investigates the use of drones for delivery. If it were successful, it would not be difficult to see military use. If so, it could be catastrophic if, for example, reward shaping came into the wrong hands. Another example is Microsoft's AI agent Tay, who a couple of years ago became racist after spending 24 hours on Twitter (Lee 2016). Similar to the case of Tay was a case in USA where an AI aided the criminal justice system. The AI agent in this case systematically overestimated the risk of recidivism in the black population (Lo Piano 2020). The possibilities of artificial intelligence are expected to change many parts of society and the use of technology, for example, car accidents are estimated to decrease as a result of autonomous driving (Đorđe Petrović, Mijailović & Pešić 2020).

Although one would hope that the changes would be for the betterment of humanity as a whole, there are no such assurances, and some of the worst tragedies have come from the best intentions. There are already examples of the current negative effects artificial intelligence (AI) has on society, allowing extensive surveillance and recognition systems (Andersen 2020). This makes it important to ensure that the AI systems created are designed to improve society and actively avoid applications that could be detrimental. Doom is a first-person shooter game whose mechanics may be likened to operating a real rifle. Given the small resolution and art style of the Doom game, it would be exceedingly difficult to transfer the agent to a real life scenario. Given previous discussions in this report on the difficulty of using the agent in different scenarios and game environments, it would be more difficult to apply these methodologies to applications even further away from a game. Regardless, the technology exists, and the decision rules that govern the action of an agent are up to its creators, for better or worse, as shown in the previous examples.

The research carried out in this report was carried out with research ethics in mind. As a product of this, all code developed, AI models produced, and general results are available online at `https://github.com/a19simma/BSc-thesis` as part of open science. This enables people to view, reproduce and validate the study.

7 | Conclusion

In this chapter, the research question is related to the results. Furthermore, a scientific contribution of the thesis and closing remarks are given.

7.1 Research Questions

Research Question one was answered in Table 5.14 where the default-valued algorithms were compared and showed a significant difference in terms of reward. When comparing the three algorithms with the default hyperparameter settings, PPO was the most effective. It was both the fastest to learn a good strategy for the scenario and its strategy was the best. A2C was unable to learn a successful strategy with the default hyperparameter settings due to the need for parallelization of the environment when using the default hyperparameters. DQN quickly learned a decent strategy after its initial learning start period. The conclusion here is that PPO is the most effective.

Research Question two was answered in Table 5.15 which showed a significant difference between the algorithms trained with and without hyperparameter optimization. Relatively A2C showed the largest difference when applying the optimized hyperparameters with an increase of 424% of accumulated reward. The algorithm also performed better than DQN after optimization. However, PPO produced the best model, both in training time and accumulated reward.

With both of these answered. Some conclusions can be drawn; both the choice of an algorithm affects both the outcome of an agent and how much they react to hyperparameter optimization. The results found suggest that there is a significant benefit in applying hyperparameter optimization. The authors recommend the use of PPO to the reader.

7.2 Research Contribution

Previous research such as Lynnerup et al. (2019) and Vohra et al. (2022) stated the need for more research on the process of optimizing hyperparameters, and how newer algorithms such as PPO and A2C respond to such optimization. The results of this thesis show that the hyperparameters chosen matter to the final result. In this thesis the authors have combined the frameworks of Optuna, VizDoom, and Stable-Baselines3 amongst others to investigate this. The conclusions are that there are benefits to draw from hyperparameter optimization. Hyperparameter optimization even when using a tool like Optuna is a matter of cost. In addition to the

above, this thesis provides a distribution of successful hyperparameter settings, which can be found in the Appendix C. The hope of the authors is that this range can narrow down the list of possible combinations of hyperparameters to try to find the right ones, resulting in a lower cost for developing reinforcement learning models.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2015), 'TensorFlow: Large-scale machine learning on heterogeneous systems'. Software available from tensorflow.org.
URL: *https://www.tensorflow.org/*

Achiam, J. (2020), 'Spinning up documentation'.
URL: *https://spinningup.openai.com/_downloads/en/latest/pdf/*

Ahmed, Z., Roux, N. L., Norouzi, M. & Schuurmans, D. (2018), 'Understanding the impact of entropy on policy optimization'.
URL: *https://arxiv.org/abs/1811.11214*

Akiba, T., Sano, S., Yanase, T., Ohta, T. & Koyama, M. (2019), Optuna: A next-generation hyperparameter optimization framework, *in* 'Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '19, Association for Computing Machinery, New York, NY, USA, p. 2623–2631.
URL: *https://doi.org/10.1145/3292500.3330701*

Amodei, D. & Hernandez, D. (2018), 'Ai and compute'.
URL: *https://openai.com/blog/ai-and-compute/*

Andersen, R. (2020), 'The panopticon is already here'.
URL: *https://www.theatlantic.com/magazine/archive/2020/09/china-ai-surveillance/614197/*

Bakhanova, M. & Makarov, I. (2021), Deep reinforcement learning in vizdoom via dqn and actor-critic agents, *in* I. Rojas, G. Joya & A. Català, eds, 'Advances in Computational Intelligence', Springer International Publishing, Cham, pp. 138–150.

Bergdahl, J., Gordillo, C., Tollmar, K. & Gisslén, L. (2021), 'Augmenting automated game testing with deep reinforcement learning'.
URL: *https://arxiv.org/abs/2103.15819*

Bergstra, J., Bardenet, R., Bengio, Y. & Kégl, B. (2011), Algorithms for hyper-parameter optimization, *in* J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira & K. Weinberger, eds,

'Advances in Neural Information Processing Systems', Vol. 24, Curran Associates, Inc.
URL: *https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf*

Bergstra, J., Yamins, D. & Cox, D. (2013), Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, *in* S. Dasgupta & D. McAllester, eds, 'Proceedings of the 30th International Conference on Machine Learning', Vol. 28 of *Proceedings of Machine Learning Research*, PMLR, Atlanta, Georgia, USA, pp. 115–123.
URL: *https://proceedings.mlr.press/v28/bergstra13.html*

Berndtsson, M., Hansson, J., Olsson, B. & Lundell, B. (2008), 'Thesis projects: A guide for students in computer science and information systems. 2nd edition'.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. & Zaremba, W. (2016), 'Openai gym'.
URL: *https://arxiv.org/abs/1606.01540*

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. & Amodei, D. (2020), 'Language models are few-shot learners'.
URL: *https://arxiv.org/abs/2005.14165*

Cobbe, K., Klimov, O., Hesse, C., Kim, T. & Schulman, J. (2018), 'Quantifying generalization in reinforcement learning'.
URL: *https://arxiv.org/abs/1812.02341*

Colas, C., Sigaud, O. & Oudeyer, P.-Y. (2019), 'A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms'.
URL: *https://arxiv.org/abs/1904.06979*

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y. & Zhokhov, P. (2017), 'Openai baselines'.
URL: *https://github.com/openai/baselines*

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L. & Madry, A. (2020), 'Implementation matters in deep policy gradients: A case study on ppo and trpo'.
URL: *https://arxiv.org/abs/2005.12729*

Fakoor, R., Chaudhari, P. & Smola, A. J. (2019), 'P3o: Policy-on policy-off policy optimization'.
URL: *https://arxiv.org/abs/1905.01756*

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G. & Pineau, J. (2018), 'An introduction to deep reinforcement learning'.
URL: *https://arxiv.org/pdf/1811.12560.pdf*

Green, S., Vineyard, C. M. & Koç, K. (2019), 'Distillation strategies for proximal policy optimization'.
URL: *https://arxiv.org/abs/1901.08128*

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. & Meger, D. (2017), 'Deep reinforcement learning that matters'.
URL: *https://arxiv.org/abs/1709.06560*

Henderson, P., Romoff, J. & Pineau, J. (2018), 'Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods'.
URL: *https://arxiv.org/abs/1810.02525*

Hutter, F., Hoos, H. & Leyton-Brown, K. (2014), An efficient approach for assessing hyperparameter importance, *in* E. P. Xing & T. Jebara, eds, 'Proceedings of the 31st International Conference on Machine Learning', Vol. 32 of *Proceedings of Machine Learning Research*, PMLR, Bejing, China, pp. 754–762.
URL: *https://proceedings.mlr.press/v32/hutter14.html*

Jin, P., Keutzer, K. & Levine, S. (2017), 'Regret minimization for partially observable deep reinforcement learning'.
URL: *https://arxiv.org/abs/1710.11424*

Justesen, N., Bontrager, P., Togelius, J. & Risi, S. (2020), 'Deep learning for video game playing', *IEEE Transactions on Games* **12**(1), 1–20.

Justesen, N. & Risi, S. (2018), 'Automated curriculum learning by rewarding temporally rare events'.
URL: *https://arxiv.org/abs/1803.07131*

Kempka, M., Wydmuch, M., Runc, G., Toczek, J. & Jaśkowski, W. (2016), ViZDoom: A doom-based AI research platform for visual reinforcement learning, *in* 'IEEE Conference on Computational Intelligence and Games', IEEE, Santorini, Greece, pp. 341–348. The best paper award.
URL: *http://arxiv.org/abs/1605.02097*

Khan, A., Naeem, M., Asghar, M. Z., Ud Din, A. & Khan, A. (2020), 'Playing first-person shooter games with machine learning techniques and methods using the vizdoom game-ai research platform', *Entertainment Computing* **34**, 100357.
URL: *https://www.sciencedirect.com/science/article/pii/S1875952119300497*

Lapan, M. (2018), *Deep Reinforcement Learning Hands-On*, Packt Publishing, Birmingham, UK.

Le, N., Rathour, V. S., Yamazaki, K., Luu, K. & Savvides, M. (2021), 'Deep reinforcement learning in computer vision: A comprehensive survey'.
URL: *https://arxiv.org/abs/2108.11510*

Lee, P. (2016), 'Learning from tay's introduction'.
URL: *https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/*

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. (2016), 'Hyperband: A novel bandit-based approach to hyperparameter optimization'.
URL: *https://arxiv.org/abs/1603.06560*

Lo Piano, S. (2020), 'Ethical principles in machine learning and artificial intelligence: cases from the field and possible ways forward', *Humanities and Social Sciences Communications* **7**(1), 9.
URL: *https://doi.org/10.1057/s41599-020-0501-9*

Lynnerup, N. A., Nolling, L., Hasle, R. & Hallam, J. (2019), 'A survey on reproducibility by evaluating deep reinforcement learning algorithms on real-world robots'.
URL: *https://arxiv.org/abs/1909.03772*

Mehdi Boubnan, A. C. (2019), 'Deep reinforcement learning applied to doom', https://github.com/boubnanm/Deep-Reinforcement-Learning-applied-to-DOOM.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D. & Kavukcuoglu, K. (2016), 'Asynchronous methods for deep reinforcement learning'.
URL: *https://arxiv.org/abs/1602.01783*

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013), 'Playing atari with deep reinforcement learning'.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I. & Stoica, I. (2017), 'Ray: A distributed framework for emerging ai applications'.
URL: *https://arxiv.org/abs/1712.05889*

Muñoz, G., Barrado, C., Çetin, E. & Salami, E. (2019), 'Deep reinforcement learning for drone delivery', *Drones* **3**(3).
URL: *https://www.mdpi.com/2504-446X/3/3/72*

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. & Chintala, S. (2019), Pytorch: An imperative style, high-performance deep learning library, *in* H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox & R. Garnett, eds, 'Advances in Neural Information Processing Systems 32', Curran Associates, Inc., pp. 8024–8035.
URL: *http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf*

Paul, S., Kurin, V. & Whiteson, S. (2019), 'Fast efficient hyperparameter tuning for policy gradients'.
URL: *https://arxiv.org/abs/1902.06583*

Raffin, A. (2020), 'Rl baselines3 zoo', `https://github.com/DLR-RM/rl-baselines3-zoo`.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M. & Dormann, N. (2021), 'Stable-baselines3: Reliable reinforcement learning implementations', *Journal of Machine Learning Research* **22**(268), 1–8.
URL: *http://jmlr.org/papers/v22/20-1364.html*

Renotte, N. (2022), 'Doomreinforcementlearning', https://github.com/nicknochnack/DoomReinforcementLearning.

S. Albawi, T. A. M. & Al-Zawi, S. (2017), 'Understanding of a convolutional neural network'.
URL: *doi: 10.1109/ICEngTechnol*

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017), 'Proximal policy optimization algorithms'.
URL: *https://arxiv.org/abs/1707.06347*

Shao, K., Zhao, D., Li, N. & Zhu, Y. (2018), Learning battles in vizdoom via deep reinforcement learning, *in* '2018 IEEE Conference on Computational Intelligence and Games (CIG)', pp. 1–4.

Shao, K., Zhu, Y. & Zhao, D. (2019), 'Starcraft micromanagement with reinforcement learning and curriculum transfer learning', *IEEE Transactions on Emerging Topics in Computational Intelligence* **3**(1), 73–84.

Shekhar, S., Bansode, A. & Salim, A. (2022), 'A comparative study of hyper-parameter optimization tools'.
URL: *https://arxiv.org/abs/2201.06433*

Silver, D. (2015), 'Lectures on reinforcement learning'.
URL: *https://www.davidsilver.uk/teaching/*

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement Learning: An Introduction*, second edn, The MIT Press.
URL: *http://incompleteideas.net/book/the-book-2nd.html*

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P. & SciPy 1.0 Contributors (2020), 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python', *Nature Methods* **17**, 261–272.

Vohra, I., Uttrani, S., Rao, A. K. & Dutt, V. (2022), Evaluating the efficacy of different neural network deep reinforcement algorithms in complex search-and-retrieve virtual simulations, *in* D. Garg, S. Jagannathan, A. Gupta, L. Garg & S. Gupta, eds, 'Advanced Computing', Springer International Publishing, Cham, pp. 348–361.

Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., Hassabis, D. & Botvinick, M. (2018), 'Prefrontal cortex as a meta-reinforcement learning system', *Nature Neuroscience* **21**(6), 860–868.
URL: *https://doi.org/10.1038/s41593-018-0147-8*

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. & Wesslén, A. (2012), *Experimentation in Software Engineering*, 1 edn, Springer.

Wu, Y., Mansimov, E., Liao, S., Radford, A. & Schulman, J. (2017), 'Openai baselines: Acktr & a2c'.
URL: *https://openai.com/blog/baselines-acktr-a2c/*

Wydmuch, M., Kempka, M. & Jaśkowski, W. (2019), 'Vizdoom competitions: Playing doom from pixels', *IEEE Transactions on Games* **11**(3), 248–259.

Zakharenkov, A. & Makarov, I. (2021), Deep reinforcement learning with dqn vs. ppo in vizdoom, *in* '2021 IEEE 21st International Symposium on Computational Intelligence and Informatics (CINTI)', pp. 000131–000136.
orđe Petrović et al.

Đorđe Petrović, Mijailović, R. & Pešić, D. (2020), 'Traffic accidents with autonomous vehicles: Type of collisions, manoeuvres and errors of conventional vehicles' drivers', *Transportation Research Procedia* **45**, 161–168. Transport Infrastructure and systems in a changing world. Towards a more sustainable, reliable and smarter mobility.TIS Roma 2019 Conference Proceedings.
URL: *https://www.sciencedirect.com/science/article/pii/S2352146520301654*

A | Appendix - Hyperparameter values

A.1 Hyperparameters

A.1.1 A2C

Table A.1: Hyperparameters for A2C.

Hyperparameter	**Default**	**Optimized**
Learning rate	0.0007	0.0007386
Lambda	1.0	0.98
Max grad norm	0.5	0.6
Gamma	0.99	0.99
Nstep	5	256

A.1.2 PPO

Table A.2: Hyperparameters for PPO.

Hyperparameter	**Default**	**Optimized**
Learning rate	3e-4	0.000259251
Epochs	10	3.25784
Entropy coefficient	0.0	0.00105699
Gamma	0.99	0.908591

A.1.3 DQN

Table A.3: Hyperparameters for DQN.

Hyperparameter	**Default**	**Optimized**
Learning rate	$1 \cdot 10^{-4}$	$1.9070714918537712 \cdot 10^{-5}$
Batch size	32	77
Gamma	0.99	0.6371027666566412
Training Frequency (Time steps)	4	3
Gradient Steps	1	10

B | Appendix - Environment

B.1 ViZDoom Scenarios

This section describes different scenarios available in the ViZDoom environment. The maps shown are represented by gray lines as walls and circles or dots as entities such as enemies, agent starting position, pick-ups and ammunition. The maps were taken from the ViZDoom GitHub account by Kempka et al. (2016).

B.1.1 Basic

The basic scenario is to check the basic functionality and viability of AI training in the environment. The setup is simply a stationary enemy, and the agent can move sideways and shoot. The scenario map is shown in Figure B.1.

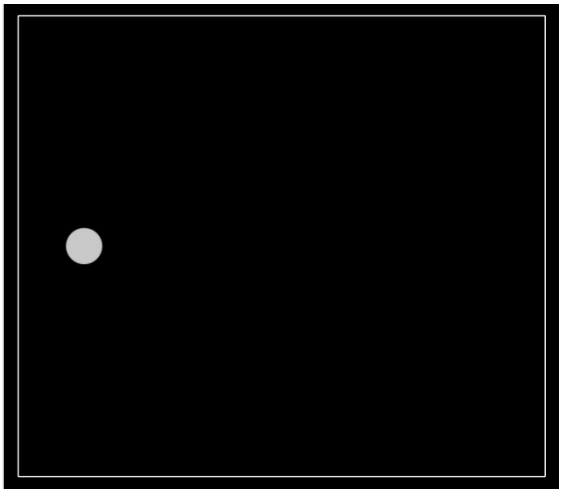

Figure B.1: Basic Scenario Map.

B.1.2 Deadly Corridor

Deadly corridor is the scenario used in this experiment and has 6 enemies behind the corners, with a green armor placed on the opposite side of the corridor. The map of the scenario is shown in Figure B.2.

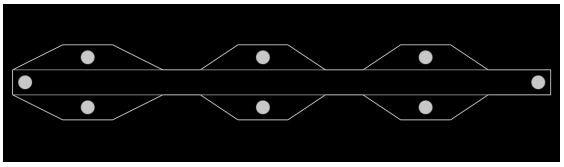

Figure B.2: Deadly Corridor Scenario Map.

B.1.3 Defend the Center

Defend the center is a circular room with enemies appearing at the perimeter moving towards the agent at the center. The agent can turn around and shoot to kill enemies before they reach the agent. The map of the scenario is shown in Figure B.3.

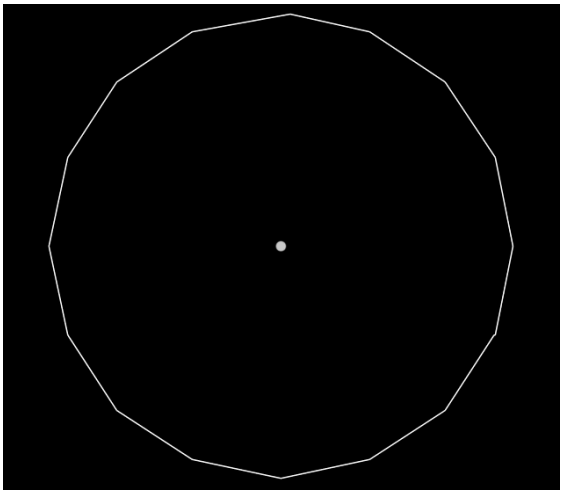

Figure B.3: Defend the Center Scenario Map.

B.1.4 Defend the Line

Defend the line is similar to defend the center, but instead of turning, the agent moves sideways like in the basic scenario with enemies approaching from the opposite side of the map. The map of the scenario is shown in Figure B.4.

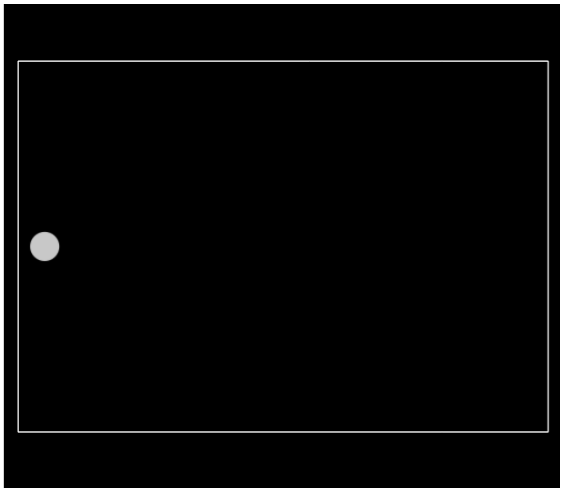

Figure B.4: Defend the Line Scenario Map.

B.1.5 Death Match

Death match is a scenario in which the agent faces enemies moving around. The objectives include killing enemies and picking up objectives such as ammunition, different weapons, and killing enemies. The map of the scenario is shown in Figure B.5.

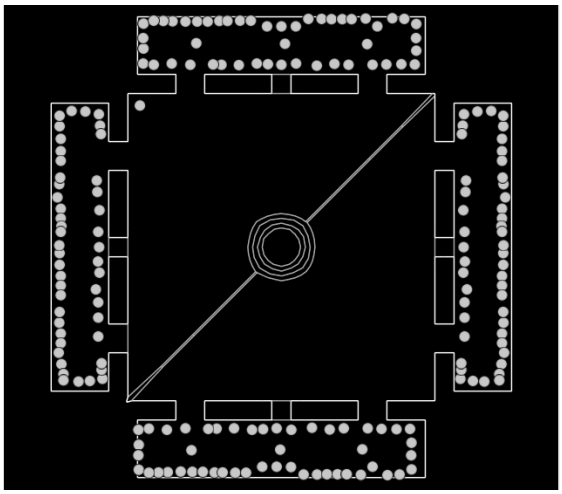

Figure B.5: Death-match Scenario Map.

B.1.6 Health Gathering

Health Gathering is a map covered in acid that damages the agent. There are medical kits on the map that recover the health of the agent when collected. The map of the scenario is shown in Figure B.6.

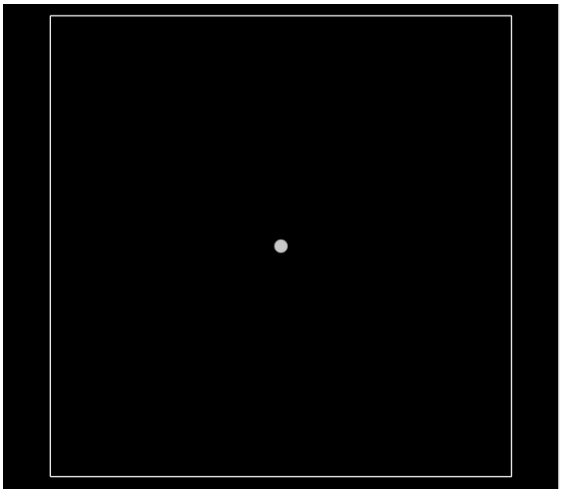

Figure B.6: Health Gathering Scenario Map.

B.1.7 My Way Home

My Way Home is a long series of corridors in which the agent is placed randomly and has to find its way to the green vest to complete it. The map of the scenario is shown in Figure B.7.

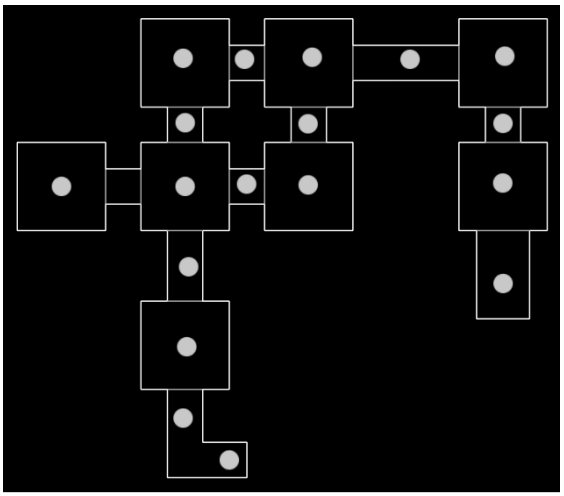

Figure B.7: My Way Home Scenario Map.

B.1.8 Predict Position

The agent is equipped with a rocket launcher that has a significant travel time to reach its target. The map is a rectangle with a monster walking back and forth on the other side of the map. The map of the scenario is shown in Figure B.8.

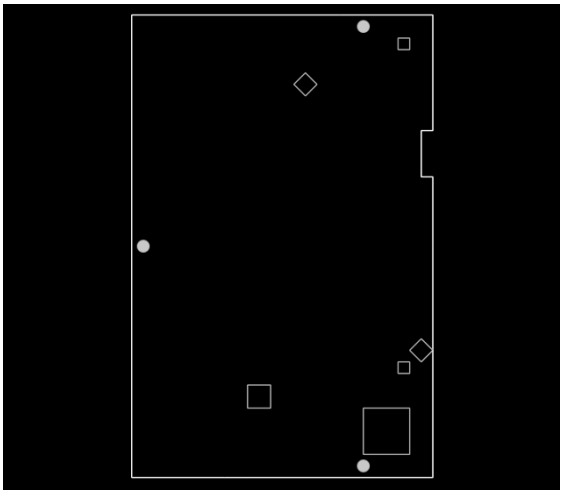

Figure B.8: Predict Position Scenario Map.

B.1.9 Take Cover

The take cover scenario is essentially the reverse of the predict position scenario with enemies firing slow-traveling fireballs that the agent must avoid. The map of the scenario is shown in Figure B.9.

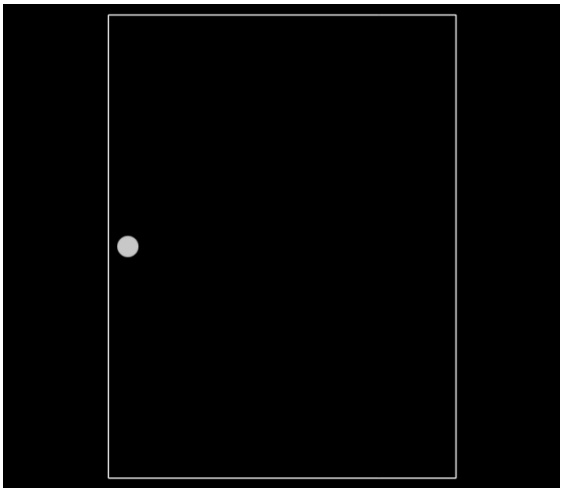

Figure B.9: Take Cover Scenario Map.

C | Appendix - Hyperparameter Distribution

C.0.1 A2C

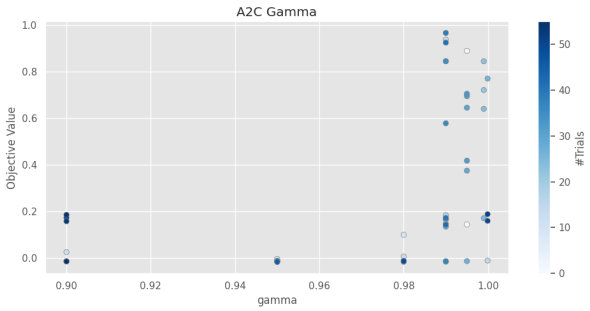

Figure C.1: A2C: Distribution of Objective value in relation to the Gamma Hyperparameter.

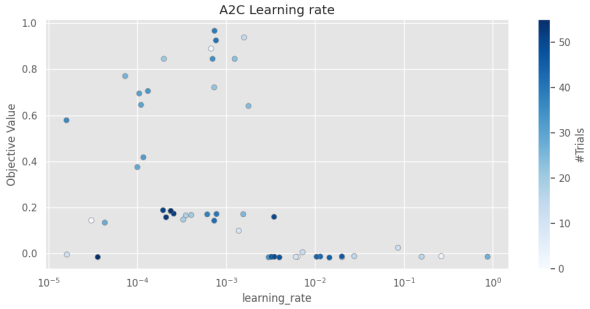

Figure C.2: A2C: Distribution of Objective value in relation to the Learning rate Hyperparameter.

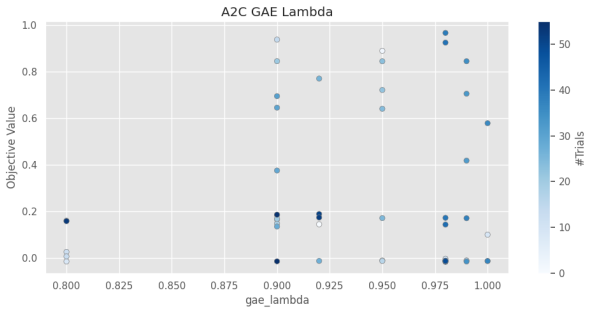

Figure C.3: A2C: Distribution of Objective value in relation to the GAE Lambda Hyperparameter.

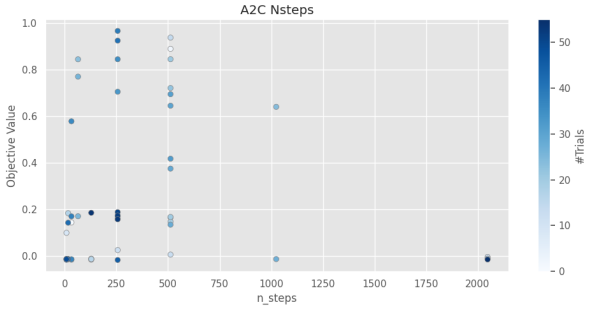

Figure C.4: A2C: Distribution of Objective value in relation to the Nstep Hyperparameter.

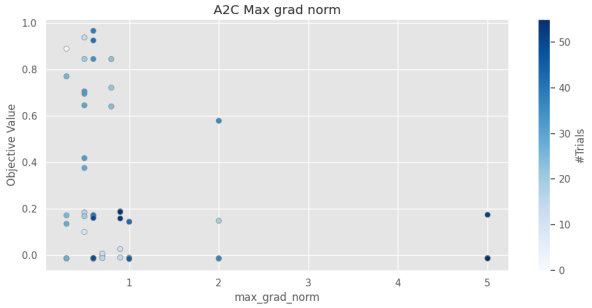

Figure C.5: A2C: Distribution of Objective value in relation to the Max grad norm Hyperparameter.

C.0.2 PPO

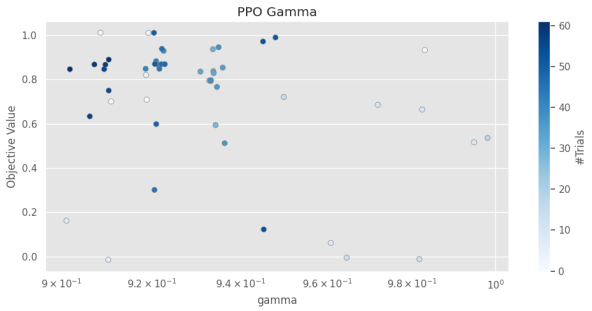

Figure C.6: PPO: Distribution of Objective value in relation to the Gamma Hyperparameter.

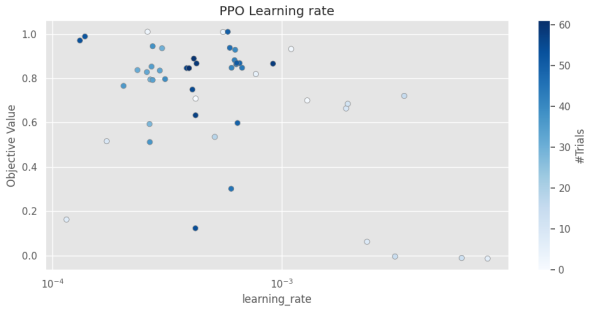

Figure C.7: PPO: Distribution of Objective value in relation to the Learning rate Hyperparameter.

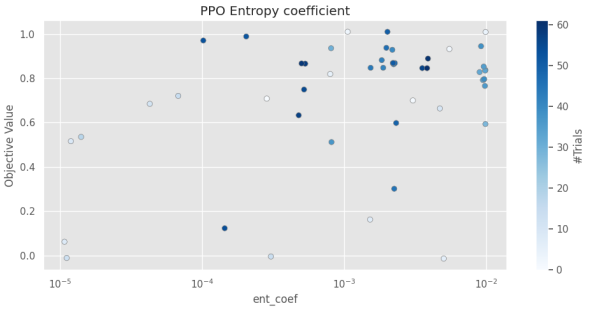

Figure C.8: PPO: Distribution of Objective value in relation to the Entropy coefficient Hyperparameter.

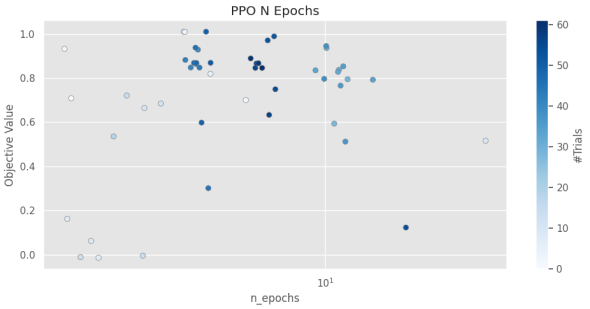

Figure C.9: PPO: Distribution of Objective value in relation to the N Epochs Hyperparameter.

C.0.3 DQN

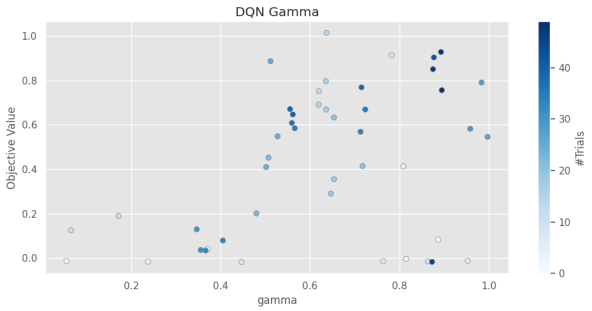

Figure C.10: DQN: Distribution of Objective value in relation to the Gamma Hyperparameter.

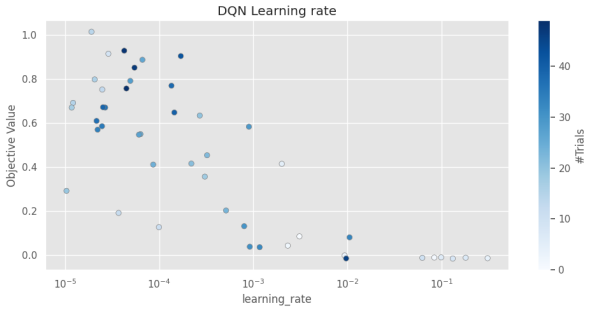

Figure C.11: DQN: Distribution of Objective value in relation to the Learning rate Hyperparameter.

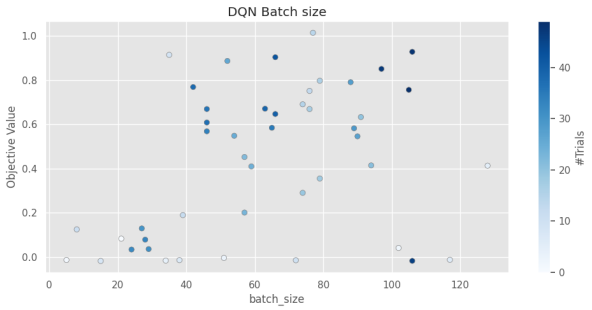

Figure C.12: DQN: Distribution of Objective value in relation to the Batch size Hyperparameter.

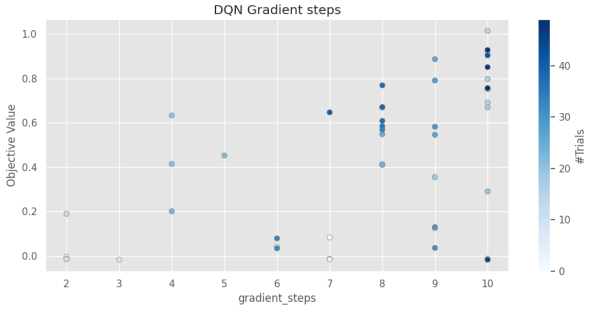

Figure C.13: DQN: Distribution of Objective value in relation to the Gradient steps Hyperparameter.

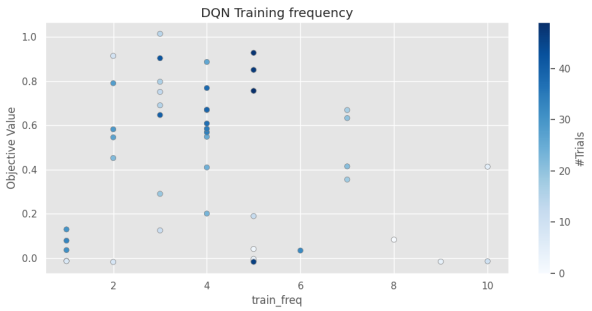

Figure C.14: DQN: Distribution of Objective value in relation to the Training frequency Hyperparameter.