Final Year Project Report

Full Unit - Final Report

Effective Exploration in Markov Decision Processes

Cougar Tasker

A report submitted in part fulfilment of the degree of

MSci (Hons) in Computer Science (Artificial Intelligence)

Supervisor: Dr. Anand Subramoney

Department of Computer Science
Royal Holloway, University of London

April 1, 2024

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:

Student Name: Cougar Tasker

Date of Submission: 08/12/2023

Signature: *Cougar Tasker*

Dedicated to my partner, friends and family:
Thank you for your continued support and encouragement

Table of Contents

Abstract

0.1 TODO - (Rework Rationale) Resourceful robots

To complete many objectives robotic agents need to adapt and learn new environments. This presents a problem for traditional supervised learning approaches that operate under the (IID) assumption [1], training data may not apply to new environments. This report investigates reinforcement learning (RL) techniques as an alternative for this class of robotics. Reinforcement Learning is a machine-learning paradigm where the learning component receives external judgment like supervised learning. However, in reinforcement learning this judgment is provided after the agent decides on the consequences of their actions. This is unlike supervised learning; there is no need for training in RL the agent will continuously learn from their successes and failures. Although RL does not need training data it is also unlike unsupervised learning, since RL agents learn to achieve more reward from within the system rather than observing underlying patterns.

These unique benefits of reinforcement learning make this field applicable to many real-world problems. RL is built upon a strong mathematical foundation that allows us to make strong guarantees about solutions in the reinforcement learning domain. Furthermore, RL spans widely uniting together many important and diverse areas of study such as Economics, Optimisation, Game Theory, robotics and Cognitive Science.

It follows that the primary goal of this project is to understand and implement reinforcement learning. As this project develops the secondary aim is to frame these solutions from the perspective of autonomous robotic agents to aid in decision-making. This robotics background fits well with this project's other requirements: such as implementing a grid world environment and Q-learning agents. Resource-gathering robots from autonomous vacuum cleaners to industrial warehouse robots can be modelled in these grid world environments and this project aims to create reinforcement learning agents that are successful in these environments. Although the project is not limited to these technologies we will also evaluate more advanced techniques such as deep reinforcement learning agents and diverse RL environments from the gymnasium library.

Chapter 1: **TODO - Introduction**

1.1 TODO (rework + extend) - Aims and Objectives

- Efficient Navigation: The agent must autonomously navigate a grid world while making decisions to optimise rewards. The robot should learn to prioritise efficient paths.
- Balancing Constraints: The agent must balance collecting resources with constraints such as energy expenditure. The robot should adapt its behaviour based on the availability of resources and its current energy levels.
- General Approach: The agent must not use any environment-specific techniques. It should learn from and function in any Markov environment.
- Deep Reinforcement Learning: The program should integrate deep neural networks (DNN) with Q-learning.
- Visualisation: The program should provide a GUI that visualises the operation of these agents interactively.

1.2 Motivation

My inspiration for studying for this degree, specialising in artificial intelligence, comes in large part from my belief that AI is becoming increasingly pivotal in shaping the future of technology and industry. For this purpose, this project presents an invaluable opportunity for my personal and professional growth. It is a fantastic platform to improve my comprehension of reinforcement learning while offering hands-on experience.

What is unique about this resource-gathering robot project is its structured progression of complexity, Starting from fundamental concepts and culminating in advanced techniques. This gradient makes the complex nature of reinforcement learning more approachable than it may be in industry.

This project interests me because of its generality and applicability to many different scenarios. Resource-gathering has the potential to incorporate many real-world constraints like energy, visibility and obstacles. I would like to see how this impacts different exploration strategies.

Last year, I completed my year-long internship at Zing Dev (Zing), a digital communications company that is progressively incorporating AI systems for its customers. This experience has demonstrated to me the value of understanding the internals of these AI systems. It is clear that AI is a clear focus for most companies, Ransbotham et al. said:

> Almost 85% believe AI will allow their companies to obtain or sustain a competitive advantage [2]

Through this project, I aim to improve my understanding of autonomous agents' benefits, biases, and limitations. This knowledge will be desirable for many companies like Zing working with artificial agents.

1.3 TODO - Literature Review

Full critical review of literature relevant to the study.

Chapter 2: **Professional Issues In Machine Learning**

Machine learning programs and techniques, as explored in this project, possess immense potential for social benefit when managed judiciously, but they also risk significant social and ethical consequences [3]. One concern is the ability of ML systems to reinforce and perpetuate societal biases in their decisions [3]. One prominent example of this dichotomy of potential was "Rekognition", Amazon's facial recognition tool (FRT) [4]. Rekognition is a cloud-based deep-learning application which can identify people, activities, objects, and more [5]. This powerful tool can bring societal benefit one example is an application that notifies professionals who are in high-risk sectors such as healthcare or construction [5]. Rekognition can make this possible by detecting Personal Protective Equipment.

Recent developments in AI have spurred many companies such as IBM, Microsoft, and Amazon to release FRTs, making identification more convenient than ever [6]. While there have been many commercial applications, the most notable example from a moral, ethical and social standpoint was the use of Rekognition by US law enforcement agencies. From 2018-2020 the FBI trailed Rekognition to assist in searching video footage collected during investigations [7]. Other agencies such as the DEA use FRTs to help identify criminals captured on surveillance cameras [7].

So why is this significant? Unfortunately, Rekognition's social impact has not been wholly positive; it has tended to falsely identify women and people of colour as criminals to a disproportionate degree [8]. Moreover, it falsely identified twenty-eight NFL players as criminals, of which thirteen were persons of colour [7]. While human oversight may be used, it is not enough. These errors have already led to serious harm, and the significant bias to race and sex is discriminatory and unacceptable.

While the intentions of operators and machine learning professionals may be for social good it is evident that without careful consideration machine learning systems may not be aligned with our goals [9]. In the context of FRT, it is clear there had been several compounding failures. Firstly it can be easy to believe that your ML system is unbiased and avoid responsibility when your algorithms are generalised and independent of any particular protected characteristic. However, vendors such as Amazon provide these ML systems pre-trained as a service. For ML models their training data can be the largest determinant of their functionality.

> Bias in, Bias out [7]

> For Microsoft, almost ninety-four percent of the faces misgendered were individuals who were considered darker individuals [7]

Biases in training datasets propagate to the decisions made by the model trained on it. These biases in the error rate are believed to be a consequence of an over-representation of some social identities in the training data [10]. Gathering more data has been long considered the goal in ML for creating better models, particularly large language models which have been directly trained on large corpora of text from the internet [10]. However, it's my belief that fair and equal representation of all protected characteristics should be a top priority even at the expense of discarding data.

While reducing biases in training is incredibly important and achievable, with datasets at scale it may be impractical or impossible to truly remove all bias. We must acknowledge and account for the shortcomings of ML systems. This is especially important where AI systems make decisions that impact people. The FBI's system involved human oversight and intervention and yet people of colour were still disproportionately harmed. In fact, in another example, even a truly unbiased technology, automatic number plate recognition (ANPR) has been found to cause a disparate impact on people of colour [11]. The implicit biases of the operators resulted in this technology being used in higher frequency in Black and Latinx communities [11, 7].

While the human factor of implicit biases makes them harder to tackle systematically, I believe several steps can be taken in machine learning to reduce the aforementioned issues:

- Unbiased systems as default, where using protected characteristic data is not necessary or avoidable, then this information should be removed where possible [12]. This may be possible to achieve with redaction or masking in some contexts.
- Active training management, ensuring equality in training datasets that include people's protected characteristics. This may be achieved by sampling and further models to categorise data [12].
- Algorithmic bias awareness, trained ML systems must have their bias quantified before their use. Stakeholders must be made aware of the system's limitations before it is approved [12].
- Implicit bias awareness, operators of ML systems should receive continuous training and feedback to ensure that they are aware of their own implicit biases and the impact of their decisions [7].
- Transparency and accountability: decision-makers at every stage of ML systems should be transparent about the decisions that they have made and accountable to the people that these decisions affect [12].

AI must be regulated in a fashion where we no longer perpetuate existing or create new social harms while not discouraging the potential societal benefits that it may bring [12].

This case study is one of many that may highlight similar or different challenges and benefits that AI may bring. Unlike the case study, the application in this project is exploratory and operates in simulated environments. In this sense, I believe it is insulated from bias and causes this type of social harm. However, this is just one facet of how these systems incorporate with society. Like machine learning as a whole RL, specifically deep reinforcement learning (DRL), increasing use to solve practical problems [13]. One place DRL is being applied is in militaries across the world [14, 15]. Of these military applications, their use in UAVs has seen impressive success. In DARPA's AlphaDogfght competition, an autonomous DRL agent beat a human F-16 pilot 5:0 and won the championship [14]. These UAVs have grown in their autonomous capabilities [15] and it's clear that in conflicts these agents may operate weapons systems. When a reinforcement learning agent's action space includes taking a human life, is this morally and ethically right?

These systems may have been trained in a simulation of military operations and rewarded for their actions that achieved military goals [14]. These experiences will shape the agent's interpretation of value. But, these experiences are unlike ours as humans. As humans, we have a vast variety of experiences, including companionship and kindness. In this respect, the value a human and DRL combatant will assign to killing their opponents may differ vastly. I believe there may be situations where an AI may pick actions that may lead to a huge loss of life only for a miniscule increase in the chance of success.

Chapter 3: **Fundamental Concepts**

3.1 Markov Decision Processes

Markov Decision Processes (MDP) provide a mathematical formalisation of decision-making problems. Markov Decision Processes provide the foundation for reinforcement learning (RL). This is because MDPs distil the fundamental parts of decision-making, allowing RL techniques built upon MDPs to generalise to learning in the real world and across different domains such as finance and robotics.

As a formal mathematical framework, MDPs allow us to derive and prove statements about our RL methods built upon them. An important example of this is that we can prove that Q-learning (an RL technique explained in chapter 3.4) will converge to the true Q-values as long as each Action-State pair is visited infinitely often. [16]. Furthermore, MDPs allow us to reason about problems with uncertainty, allowing RL agents to account for randomness in their environment.

The standardisation of decision-making problems as MDPs allows for a uniform definition of optimality with the value functions. MDPs give a basis for assessing the performance of RL algorithms, facilitating like-for-like comparisons for different RL approaches.

3.1.1 Markov Property

The Markov property is that the future state of a Markov system only depends on the current state of the system. In other words, if we have a system that follows the Markov property, then the history preceding the current configuration of the system will not influence the following state.

To put the Markov property formally, S_t represents the state at some time as t. S_t represents the outcome of some random variable. Then the Markov property would hold if and only if:

$$\Pr(S_{c+1} \mid S_c, S_{c-1}, \ldots, S_0) = \Pr(S_{c+1} \mid S_c) \tag{3.1}$$

This definition demonstrates how the Markov property can hold in non-deterministic, stochastic processes. It also shows that predictions that are only based on the current state are just as good as those that record the history in a Markov process. The sequence of events in this definition, S_t, is called a Markov Chain [17].

3.1.2 Extending Markov Chains

Markov Decision Processes extend Markov Chains in two important ways. Firstly, MDPs introduce decision-making through actions. Each state in an MDP has a set of actions available to it. In each state, an action is required to transition to the next state; this action with the current state can affect what the following state will be. Secondly, MDPs introduce a reward value. The reward is determined from the current state and action; it is produced simultaneously with the following state.

A formal definition of a Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}_s, p)$ where:

- $\mathcal{S}$ defines the set of all states
- $\mathcal{A}_s$ defines the set of available actions in state s
- p defines the relationship between states, actions and rewards:
  $p(s', r \mid s, a) \doteq \Pr(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$ [18]
  - $s, s' \in \mathcal{S}$, $a \in \mathcal{A}_s$ and $r \in \mathbb{R}$
  - $p : \mathcal{S} \times \mathbb{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$

The function p is an integral part of this definition; it fully describes how the system will evolve. We call this function the dynamics of the MDP. What this definition does not describe is how actions are chosen. This decision-making is done by an entity called an agent. For our purposes, the agent will have complete visibility as to the current state of the MDP. However, like most real-world situations, our agent will not have any a priori knowledge of the dynamics.

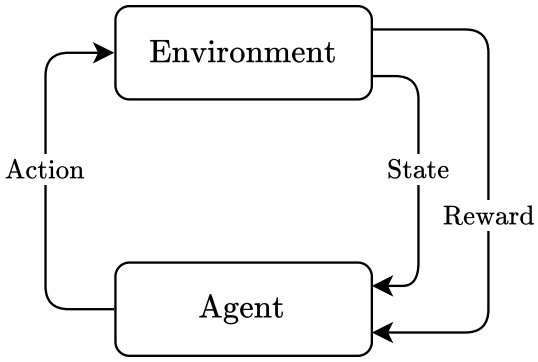


Figure 3.1: The agent-environment interface

The agent comprises the entire decision-making entity in an MDP; anything unknown or not wholly under the agent's control is called the agent's environment. In the context of reinforcement learning, the environment is essentially the dynamics of the MDP. Figure 3.1 demonstrates how the agent and environments affect each other in an MDP.

For learning agents, we wish to improve the agent's behaviour over time. For this purpose, we introduce a policy π. This policy defines the action chosen by an agent under a particular state. The policy can be represented with a lookup table like in Q-learning 3.4 or a more complex process such as deep Q-learning. A policy like this is not hard-coded, allowing the agent to update the policy based on the information the agent learns from the environment.

3.2 Policy and Value functions

After each action, a reward is received. It follows that the goal of an agent should be to choose the actions to maximise these reward signals received. Following the Markov principle and the definition of an MDP, this reward only depends on the current state and the action chosen. Consequently, being in some states and performing some actions are more valuable to the agent than other states and actions. We can define value functions:

- $v(s)$ function determines the value of being in a given state

- $q(s, a)$ function determines the value of being in a given state and performing a specific action

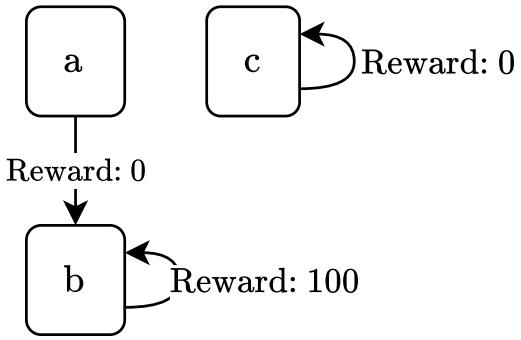


Figure 3.2: An example of transitive value of states $v(b) > v(a) > v(c)$

Intuitively, the value of being in a state is more than only its immediate reward that might be found from performing actions in that state. It is also related to the potential future reward that might be achieved in the reachable subsequent states. This can be demonstrated with two states a, and b, where there is a large reward at b and the only way to reach b is through a. It is clear that despite the identical immediate rewards, the state a is more valuable than c due to future potential at b available at a. However, a can be considered less valuable than b, because it always requires more steps to achieve the reward from a than at b. To account for our preference for more immediate rewards, the value function should also discount future value with a parameter γ.

These value functions go hand in hand with an agent's policy; a good policy maximises being in valuable states and performing valuable actions. On the other side of the coin, the value is determined by the subsequent states and rewards, which are in part determined by the actions the policy selects. Basing the policy on the value function gives the value function's definition impact over that agent's decision-making, in particular, the discount rate (γ). With high discount rates $\gamma \approx 1$, the agent can be far-sighted and ignore short-term high-reward actions available to it and take longer to learn. With low discount rates $\gamma \approx 0$, the agent can be short-sighted, ignoring the potential long-term benefits of specific actions.

While the policy is informally described at the end of chapter 3.1.2, a formal definition of a policy (π) is the probability distribution of an agent picking a given action in a given state:

$$\pi(a \mid s) \doteq \Pr(A_t = a | S_t = s) \tag{3.2}$$

Where $s \in \mathcal{S}$ and $a \in \mathcal{A}_s$. This definition shows how the policy can be stochastic. A stochastic policy can be beneficial in many ways, such as breaking ties where multiple actions are equally good and choosing between when to explore more or seek rewards.

3.2.1 optimal policy/value function via the Bellman equation

With a known policy and dynamics, the future state can be wholly determined, allowing for a complete mathematical definition of the value functions under a given policy that describes our above intuitions. for the state-value function (v) and action-value function (q) under a policy π we have the formulas:

$$q_\pi(s,a) \doteq \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \tag{3.3}$$

$$v_\pi(s) \doteq \sum_a \pi(a \mid s) q_\pi(s,a) \tag{3.4}$$

These value functions are defined recursively in terms of each other; these definitions can be unrolled to only be in terms of themselves. The unrolled form of the state-value function is known as the Bellman equation. These equations are named Richard Bellman, who, in the process of developing the field of dynamic programming, created them [19].

These functions demonstrate the intertwined relationships between the policy chosen and the value of that state if there is a particularly valuable action $a^\star$ such that $q_\pi(s, a^\star)$ is far better than for all other actions. A policy $\pi(a^\star \mid s) = 0$ would hamper the potential value of s. Therefore, the value function can be used to compare how well different policies perform. If for policy π_a there does not exist another policy π_b such that π_b has a better value $v_{\pi_b}(s) > v_{\pi_a}(s)$ for all states $s \in \mathcal{S}$ then we can consider this policy π_a an optimal policy. There may be many optimal policies; however, we often do not need to distinguish them, so we often denote any optimal policy with $\pi_\star$. This is because all optimal policies share the same state-value, and by definition action-value, function, we denote this $v_\star$ and $q_\star$. The optimal value function $v_\star$ is known as the Bellman optimality equation. These optimal equations can be written formally as:

$$q_\star(s,a) \doteq \max_\pi q_\pi(s,a) \tag{3.5}$$

$$v_\star(s) \doteq \max_\pi v_\pi(s) \tag{3.6}$$

3.2.2 Finding optimal policies by iteration

Although optimal policies exist, finding them is another matter. The policy search space is potentially infinite so an intelligent method is required. An optimal policy can be extracted from an optimal value function and the dynamics of the MDP; the optimal policy would only select actions that result in the highest value. Finding the optimal value function with the optimal policy is straightforward, but this is a catch-22. The optimal value function must be self-consistent with the Bellman equations. One approach to solving these equations is iteration, for each step moving slightly closer to the optimal solution from an initial guess.

Policy Iteration

In policy iteration, we improve a policy over time until it is optimal. Updating a policy like this is only possible because of the policy improvement theorem. This theorem considers if we have a policy π_{old}. We are at some state s, π_{old} will pick the action a under this state $\pi_{\text{old}}(a \mid s) = 1$ what happens if we consider some other action a' but then continue to follow the original policy. Because we continue to follow the original policy, we can use the existing value function $v_{\pi_{\text{old}}}(S_{t+1})$ for the subsequent states. We call this slightly adjusted policy π_{new}. By applying the Bellman equations and the existing value function we can recalculate the value at s of π_{new} if this new value is better than the original policy then we know that π_{new} must be as good if not better for all states $s \in \mathcal{S}$ than π_{old} thus π_{new} would be a better policy.

$$\sum_a \pi_{\text{new}}(a \mid s) q_{\pi_{\text{old}}}(s,a) \geq v_{\pi_{\text{old}}}(s) \Rightarrow \pi_{\text{new}} \geq \pi_{\text{old}} \tag{3.7}$$

For some policy π you can apply this policy improvement theorem for every state and action in the MDP. This approach of comparing all actions over all states is called policy improvement. This policy improvement step can be applied iteratively until the policy stops improving. If the policy does not improve over this policy improvement step, then all of the actions are optimal, and this policy is optimal

Although this policy improvement sounds computationally expensive, each state can be considered simultaneously and with a shared base policy; in each iteration, the state-value function is the same; caching this removes redundant calculations. Calculating the value function is improved by using an iterative approach and utilising the previous value function as a launching point.

Value Iteration

Policy iteration is a practical approach, for a finite MDP is guaranteed to finish in finite time. In practice, policy improvement does better and only takes a few iterations. However, in each iteration, multiple full sweeps of the state space are required. The idea of Value Iteration is to improve the policy within the Value Iteration step. This Value Iteration approach only requires one iteration.

The Bellman equation 3.4 can be used as an update rule to compute the value function iteratively. A table of values is maintained for each state, initialised randomly. The value of each state can be updated based on the immediate reward and the current estimates of the subsequent states; this is guaranteed to reduce the error at that state because the γ discount rate discounts the error at the subsequent state. This process is called bootstrapping; the smaller γ, the quicker the error rate will decrease, and the faster the process will converge. When the inconsistency at each state is suitable, the process will stop.

This standard approach uses the traditional value Bellman equation 3.4 to find the value for a given policy. In value iteration, there is an implicit policy that exclusively picks the action that has the maximum value. This policy is optimal for the optimal value function; when the Value Iteration converges, it must be optimal because of these conditions. This augmented update rule can be defined as:

$$q_{\max}(s,a) \doteq \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{\max}(s')] \qquad \text{(from 3.3)}$$

$$v_{\max}(s) \leftarrow \max_a q_{\max}(s,a) \tag{3.8}$$

$$\therefore v_{\max}(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{\max}(s')] \tag{3.9}$$

As this new update rule involves no explicit policy, a final step is required in Value Iteration to extract an explicit policy. In this step, the action that leads to the best value for each step, according to the $q_\star$ function.$q_\star$, can be derived from the $v_\star$ with the dynamics p function.

3.3 Learning as incrementally optimising policy in a MDP

Learning is the process of acquiring new information and skills. There are many different ways organisms can learn. In psychology, one of the methods is operant conditioning. In Operant Conditioning, the learner receives feedback from the environment as either a reward or punishment for completing different actions [20]. In an experiment, this could be some food for opening a door, and this feedback influences the learner's future actions. This mechanism has equivalences to reinforcement learning. For this reason, RL is considered the dominant theoretical framework for operant learning [21].

To demonstrate learning as policy optimisation, consider an agent that is introduced to a new Markov environment. This agent has no prior knowledge; therefore, its initial policy is independent of the environment's features. Eventually, the agent will pick some action in a state and receive some reward from the environment. This information can be used to update the policy and prioritise one successful action. Since the environment obeys the Markov property, repeating this successful action will continue to reward. If the agent continues to optimise its policy as it finds rewards, then it can improve its average reward. Through optimising its policy incrementally, this agent has learnt which actions are more successful and has improved its behaviour.

3.4 Q-learning

Q-learning is like value and policy iteration; all search for an optimal value function. However, Q-learning operates under extra constraints. The value and policy iteration extensively use the MDP's dynamics (p). In most real-world problems, the dynamics are unknown or too complex to be represented accurately. Iteration approaches can be adapted using samples to work without p. Samples are captured from the environment when the agent performs actions, which can be chosen randomly or by following another policy. Monti-Carlo is another technique that uses samples to emulate the value functions more directly, but these approaches are inherently offline. While offline methods have many advantages, their shortcomings, such as the inability to adapt to changing environments, make them unsuitable for many applications.

3.4.1 Temporal difference

Temporal difference (TD) algorithms are another class of RL algorithms. TD is an online process that improves the policy as new data becomes available. The goal of TD is to minimise the δ parameter that represents the difference (error) between the observed and predicted rewards. This difference δ is used to update the model like the iterative approaches we bootstrap our model over time. The magnitude of each update is controlled with the learning rate parameter, α. α helps avoid overfitting to samples since observations made in the real world may be noisy and change over time. The learning rate and discount rate can affect the convergence rate for TD; however, these are distinct variables and have different purposes. When α is low, the process will take longer to converge; however, if α is too large, the process may diverge

Reinforcement learning algorithms typically can learn in two fashions: on-policy and off-policy. On-policy algorithms learn while the agent uses the policy being improved; an example

is SARSA. Off-policy algorithms typically learn the value functions while the agent follows a different "behaviour" policy. While on-policy techniques can start exploiting their knowledge for reward quickly, they can get stuck in local minima. Off-policy techniques can provide more control over the exploration and exploitation. Q-learning is an off-policy TD reinforcement learning algorithm implemented in this project.

3.4.2 Definition

As the name suggests, Q-learning learns the optimal action-value function q to find the optimal policy. TD techniques must learn the q function directly. The v function requires knowledge of the dynamics to derive the optimal policy; however, with the q alone, the optimal policy can be determined. For this purpose, Q-learning needs to maintain a table entry for each action in each state so these entries can be updated after each observed action. We will represent this estimate of q with Q. There are five parts to each transition:

- S_{t-1} the previous state before the transition
- A_{t-1} the action that was performed
- R_t the reward received
- S_t the new and now current state

Q-learning uses these observations to update its estimates with this formula:

$$Q(S_{t-1}, A_{t-1}) \leftarrow Q(S_{t-1}, A_{t-1}) + \alpha[R_t + \gamma \max_a(Q(S_t, a)) - Q(S_{t-1}, A_{t-1})] \tag{3.10}$$

This formula can be thought of as interpolating the old estimate at the old state $Q(S_{t-1}, A_{t-1})$ with this new observed Q-value $R_t + \gamma \max_a(Q(S_t, a))$. When $\alpha = 1$, this formula replaces the existing value with the new observed Q-value. When $\alpha = 0$, the observation is ignored like it never happened. The formula $R_t + \gamma \max_a(Q(S_t, a))$ calculates the new observed Q-value based upon the same principle as Value Iteration; the implicit policy is to pick the best possible action.

3.4.3 Implementation conditions

Q-learning is guaranteed to eventually converge the Q estimates to the optimal q values $q_\star$ provided that the behaviour function visits all state-action pairs infinitely often. In practice, these q-values do not need to be perfect to derive an optimal policy. However, it can still take many visits to converge enough. Many observations may be necessary to build up a picture of the probability distribution and isolate noise. However, another reason for repeated observations is that the behaviour policy moves the agent forward in time, but the Q-learning table updates the last state. Therefore, the updates are in conflict: Information propagates in the opposite direction of the updates that spread it. For example, suppose a sequence of n states-actions have no reward but lead to some large reward at the end. as the behaviour completes these actions. In that case, only the last action will be updated to reflect the potential value, and every time the sequence is repeated, some of the value will propagate back one step. Many Q-learning implementations like ours will replay recent observations in reverse order to improve this performance. This is called the action-replay process (ARP). Replaying observations can be particularly effective when getting new observations is costly or slow, allowing for quicker convergence.

In 3.10, we can see how α controls the influence of each observation. But how do we tune this hyper-parameter? One option is to treat it like other hyperparameters where possible and use previous experimentation to find a practical value. However, one of the main reasons for using RL is that it does not require prior knowledge, so this is not always suitable. A fixed learning rate may not also be suitable. Some observations may be more significant than others; It has been observed that a decaying learning rate has been more effective. A decaying learning rate is believed to allow learning algorithms to avoid local minima at the beginning with the large updates and then settle on a global minima as the learning rate decays [22]. The paper "Learning Rates for Q-learning" [23] derives how polynomial learning rates such as $\alpha = 1/t^{\omega}$ converge much better than linear ($\alpha = 1/t$) rates.

Picking the behavioural policy is important; it must balance exploring and gaining rewards (exploitation). For some policies, the ε parameter determines the ratio of exploration and exploitation. A high ε would result in more exploration. There are two common behaviour policies for this:

- ε-greedy: this policy randomly picks between (A) selecting the best action based on the current Q-table or (B) selecting another action. A or B is random with the ratio determined by ε
- ε-soft: this policy assigns probabilities to all actions based upon their q-values, biased towards the higher Q-values by ε. Then, random actions are chosen according to this probability distribution.

Both ε-greedy and ε-soft policies utilise the current Q value estimates, which can lead to bias. Incorrect over-optimistic and over-pessimistic estimates can lead to a poor distribution of observations, compounding these effects. One approach to limit bias is called double Q-learning. This is where two Q-learning tables are maintained, and actions are chosen based on alternating tables. This helps average out the bias and improves the accuracy of estimates.

3.5 Measuring Performance

Accurately measuring the effectiveness of machine learning algorithms is essential for guiding research and assessing the state-of-the-art. For supervised learning tasks such as classification, this can be relatively straightforward. The model can be assessed new data against its goal. However, in RL, several important differences make assessment difficult. Firstly, the agent's decisions influence the training data. This influence creates a unique challenge: evaluating the performance of different algorithms becomes an apples-to-oranges comparison [18].

The second key challenge is RL's sensitivity to hyperparameters and environmental factors. RL algorithm's behaviour can vary wildly, and getting the suitable parameters can be make-or-break. This situation can lead to the "tune-and-report" method, where researchers tweak hyperparameters to achieve good performance [24]. This method can be misleading because it is hard to know if an algorithm is genuinely better or benefited from extensive tuning for a specific environment.

> Recent reproducibility analyses have shown that reported performance results are often inconsistent and difficult to replicate [24]

A third challenge is what metric to assess the model; unlike supervised learning, the agent receives its performance signal from the environment, which means the objective and amount of reward can vary wildly between different environments. This dependency can complicate comparisons between different environments. However, these comparisons are necessary to measure how well the algorithm generalises. One solution is creating a standard testbed of varied environments and a protocol for exploration [25, 24]. In this approach, the agent only receives meta information such as the size of the state space. So the agent must be able to adjust to the environment automatically. In my opinion, this integrates naturally with the core philosophy of refinement learning. However, this is incompatible with the traditional algorithms explored in this project as they have hyperparameters.

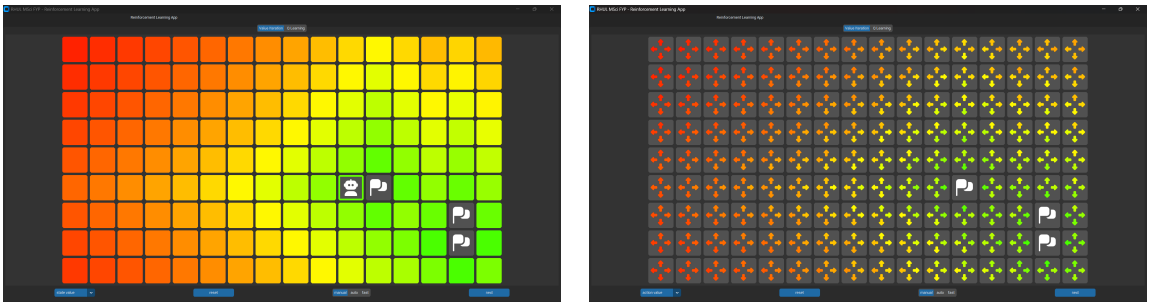
Another approach to address this third challenge is to use the optimal policy as a benchmark. That is, the performance measures should be relative to the optimal policy for that environment. There are two main measures of this kind. The first measure is called regret. It is the difference in cumulative reward between an optimal agent and the exploitation strategy during learning. Regret is a commonly used metric [26]; however, it requires the environment to have a cut-off for this calculation. While there may be a natural choice in episodic tasks, this cut-off is yet another tunable parameter that contributes to the second problem. Regret is relevant when you are interested in the reward gathered during learning. You may be interested in regret with non-stationary environments or otherwise require continuous learning.

The second measure is called the characteristic time T^*. This measure comes from the best policy identification (BPI) problem. There are two phases of the BPI problem: training and testing. During the training phase, the objective is to learn the best policy to maximise the reward in the testing phase. This two phase approach bears some similarity to supervised learning and can be applied to learning games such as Go or Chess. [27]. If regret is to measure effective exploitation, T^* is a measure of efficient exploration. The characteristic time is a complex formula that essentially measures the minimal number of samples required to identify a nearly optimal policy in an MDP [28]. A useful approximation of T^* is the number of time-steps t needed to maximise the following formula [26]:

$$1 - \frac{\|V^\star - V^{\pi_t}\|_\infty}{\|V^\star\|_\infty} \tag{3.11}$$

The characteristic time measure was developed on the multi-arm bandit problem [28]. It uses the Kullback-Leibler divergence between an allocation vector and the true distribution. Each MDP has a corresponding optimal allocation vector, which represents the best allocation of samples (action-taking) to determine the underlying distribution. However, this problem is non-convex [29] and requires extensive knowledge of the MDP's underlying distribution.

Chapter 4: **Software Engineering**



(a) The V (state-value) table (b) The Q (action-value) table

Figure 4.1: Two of the application's visualisations

This chapter details the software engineering of the project and demonstrates how this allowed the project to succeed. Software engineering consists of methods and techniques to develop large and robust applications [30]. Software engineering techniques are principally aimed at managing complexity and evolving requirements. As functionality and complexity have grown, the software engineering consideration of this chapter has become crucial. At the time of writing, the application has over 5,300 lines of Python code across 150 files.

4.1 System Design

This application is written in Python with the model-view-controller architecture. Python was chosen for many reasons, such as its robust ecosystem of machine learning libraries, which are widespread and crucial for performant deep reinforcement learning in Python. Python is also a high-level language that is clear and readable. However, python has notable constraints. The predominant Python interpreter, CPython, is substantially slower than other implementations [31]. For compatibility, I have chosen this interpreter. To mitigate the effects in the most performance-sensitive areas of the application, I have used libraries such as Numpy, Numba, and Pillow, which allow me to move the bulk of the computation out of Python.

The application has a complex GUI with controls that visualise the state of more complex reinforcement models. The model-view-controller architecture is well suited for this application's requirements, as it helps to reduce the coupling between these two different but connected modules. This reduced coupling has proved invaluable as I needed to change the GUI library several times to meet my requirements. Due to the nature of modern GUI frameworks, the code traditionally considered the view receives update signals from the user. For this reason, this project requires the use of the layered MVC pattern [32, 33], where the controller acts as a bridge between the model and view. This is similar to the MVVM and MVP architectures [34].

Multiprocessing is a method used to enable parallelisation. In the application this allows the model to perform expensive computations without blocking the view. To the user, this means the application remains responsive at all times. Furthermore, this parallelisation allows the program to use the hardware available more effectively, therefore completing tasks sooner. Unfortunately, multithreading could not be used for this task since Python implements a global interpreter lock (GIL). This lock prevents an interpreter from executing Python code

in parallel.

In this multiprocessing model, the program operates across multiple processes, each with its own address space. This separation presents an important challenge; for the program to work as one it must communicate between these processes. This is called inter-process communication 4.2. Managing communication effectively is vital to avoid race conditions. The MVC architecture provided a clean interface where the controller is responsible for transferring communications. At this boundary, to avoid race conditions, I use a request/response model where requests are queued and validated before being applied. On the response side, the model is considered a source of truth and the view represents the most recent snapshot available.

The application has been written in an object-oriented fashion with modern software engineering principles in mind [30]. Object-oriented code is a programming paradigm built around the idea of combining state and functionality into one entity called an object. The goal of object-oriented code in this project and software engineering is to reduce coupling and increase cohesion. Coupling describes how interrelated different objects are, while cohesion describes how related code is within the same object. This application achieves low coupling and high cohesion by adhering to software engineering principles [35] and using software design patterns where applicable.

Figure 4.2 shows the overall structure of the application. While this diagram omits some detail it includes all the essential components. Starting from the top, the core of the model is the top-level entities, these are the agents, environments and statistics. These core components together form a simulation. The learning system takes in the configuration provided by the view and sets up the primary simulation with these options. The learning system also performs other operations, such as compiling the main simulation's current state together into a format that is useful to the view. The hyperparameter system runs multiple simulations with different hyperparameter configurations and records their results. Like the learning system this provides feedback and results to the view in its own format.

In the middle of the figure 4.2, the controllers connect the view and model with two bridges for bidirectional communication. The controllers also validate the requests and transform them into the actions performed on the model. Finally, at the bottom is the view; packages such as the state display create a view from the current snapshot of the model's state for the user to see. The UI controls package not only represents the current state, with the user's action, can trigger new requests. The statistics package contains the functionality for displaying simulation results, such as the reward history and hyperparameter tuning. The graphing package contains a framework for embedding graphs in the application in a way that allows them to fit in with the theme and enables exporting to pdf with a file picker.

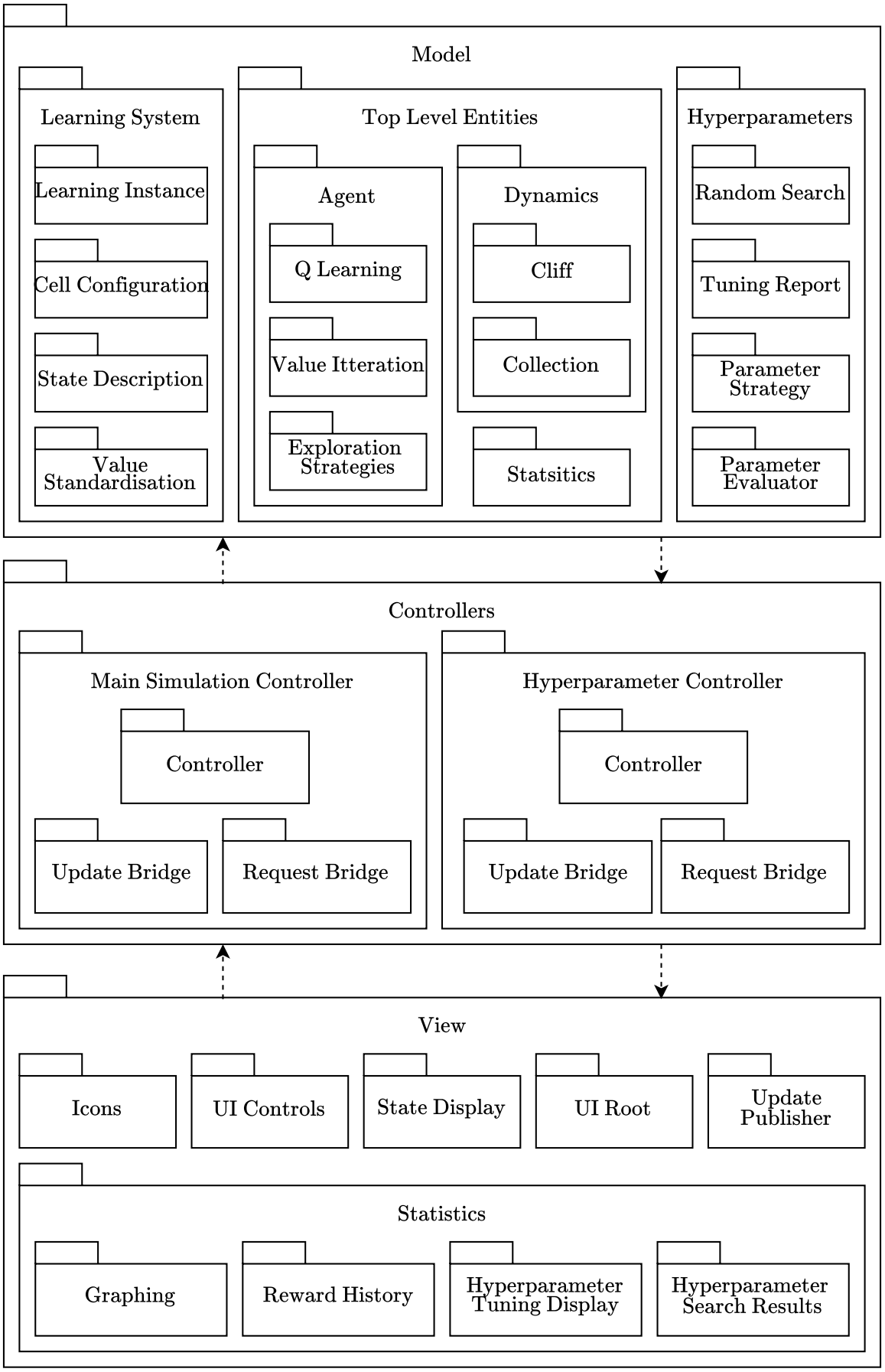


Figure 4.2: UML package digram of core packages from the application

4.2 Technical Decisions

Software development is made up of a large number of technical decisions. Most decisions are small and have a limited scope and consequences. In this section, I would like to highlight some of the key decisions and their impact on the project. Which algorithms to analyse is central to the project and sets direction as a whole since it is the foundation upon which the rest of this project is based. Within reinforcement learning, there is a wide variety of available algorithms. These can widely be grouped into two main categories: tabular and function approximation. Of the latter category, there is deep refinement learning, one of the most cutting-edge areas of reinforcement learning research and where some of RL's biggest success stories and practical applications lie.

Deep reinforcement learning is the application of deep neural networks to reinforcement learning, which can be applied in many ways, such as state representation or function approximation of value functions. While running these models can be straightforward, training them can be challenging. The cutting-edge models are trained using custom hardware at an immense cost [36]. While this project would not operate at that scale, even a modest network requires substantially more computation than a tabular method. Secondly, hyperparameter tuning is a significant challenge to these models [37]. When slow iterations are combined with a difficult search, it can be severely time-consuming. This high startup cost can be mitigated partly by building upon others work but this is not the direction I wanted to take this project. With the potential risks of DRL, I chose tabular methods, allowing for an expansion of the scope in other directions.

Of the tabular methods, Value-Iteration and Q-learning have been implemented. Value-Iteration is a dynamic programming technique that efficiently solves the Bellman equations. Value-Iteration is an ideal starting point since it provides a baseline to measure other algorithms. Implementing Q-learning is essential because it is a practical algorithm and it is the basis of a lot of work in RL. Secondly, Q-learning is off-policy, allowing for the implementation and analysis of different exploration strategies.

As discussed in the system design section 4.1, the program requires a multiprocessing architecture to maintain a responsive user interface. Consequently, the application needs inter-process communication (IPC). This IPC forms the backbone of the application, so managing what and how to communicate has wide-reaching implications. Many IPC mechanisms are available to Python applications, such as signals, message queues, and shared memory. Shared memory allows two different processes to access the same memory at the same time. This can be performant but if not managed effectively, can easily lead to several different kinds of race conditions. Signals can be used to avoid these race conditions by blocking processes from accessing the same resources at the same time. I decided not to directly use shared memory in my application due to these concerns. However, I do use shared values where appropriate. Shared values are used to build up the hyperparameter reports between the worker processes. Shared values are a managed alternative to shared memory provided by Python. These managed objects, like shared values and queues help avoid some of the pitfalls of multiprocessing; but, this has a performance and latency overhead.

The link between the primary model and view processes is the most critical across the application; any latency will be directly experienced by the user, and if it were to break, the program might not be able to recover. With these concerns, I found the built-in solutions were not appropriate due to their relatively large overhead and latency. Instead, I chose the ZeroMQ messaging library for its focus on low latency communication with a fantastic reputation. ZeroMQ is used by many large companies, it has a large community and is very stable. With this stability I fell confident I can use it to build the backbone of the application. Unlike the inbuilt queue, ZeroMQ provides IPC with a message queue without a message broker, this is

what the zero stands for.

There are many IPC styles. Between the model and view I use two publish-subscribe channels. At initiation each channel's details are negotiated with traditional IPC. I chose to use this dual pub-sub architecture as it allows unsolicited communication, unlike the Request-reply (Remote procedure call) pattern. These unsolicited messages allow the model to provide updates continuously as the model changes, such as providing constant feedback on the progress of the hyperparameter processes.

When communicating messages between processes, the data must be communicated concisely while the receiver must interpret in the same way the sender intended. This process is called serialisation and it is crucial since it directly influences the work necessary in IPC. One interchange format is textual, with XML and JSON being common choices. However, textual formats are not particularly efficient, especially for numerical data. Since the application predominantly transfers boolean and numerical values, I chose a binary serialisation format. Since both endpoints are python based and trusted, I could use the standard pickle module. This module is ideal for our application since it is flexible, allowing it to be extensible rather than some fixed or custom serialisation.

When serialising messages, compression should be a key consideration. Compression allows data to be reduced in size, this can be done in two fashions, lossy and lossless. In lossy compression uninformative information is removed to reduce in size. While the numbers transferred may not use the complete precision or range available, I have chosen to stick with a standard format for simplicity. Otherwise, due to the nature of the data of this application, there isn't any real opportunity for lossy compression. Lossless compression involves identifying patterns in data and representing these more concisely. This process could be used with this application however, I have chosen not to since the size of messages is so small the lossless compression overheads are not worth the gains.

4.3 Workflow

All aspects of this project are stored under the Git version control system, and changes made on this project are grouped into small units called commits. Each commit belongs to a branch and has a message summarising the changes. Conventional commits [38] is a standard for commit messages. In this project, I have followed this standard because it keeps the commit message consistent and enables the use of tools such as automated change log generation and versioning. As this project has a single author, I have chosen to use branches to group related work together instead of following a specific branching strategy. However, like most branching strategies, the main branch represents the project after each stage of work is finalised. Therefore this branch is always in a complete and functional state without blocking partial work.

One essential part of this project and its workflow is documentation. The documentation is written alongside the code in what are called docstrings. This serves many purposes. Firstly, the co-location helps keep the documentation in sync with the application. Special linting rules ensure that these docstrings are written consistently for every piece of code, guaranteeing 100% documentation coverage. Docstrings are also integrated into the Python language with the 'help' command, and most editors integrate docstrings so they are accessible during development. Importantly, this documentation is made accessible as a documentation site. This site is generated automatically for these docstrings and other aspects of the code. This process results in a searchable, linked documentation page that is up-to-date [39] and covers all of the code.

Testing is integral to this project. As part of my workflow, I write unit tests. Unit tests are pieces of code that perform validation and verification parts of the main application. A unit testing framework has been configured to run these tests and collate the results. All tests must pass before each commit can be made, which means that regressions are spotted quickly. These tests are also integrated into the IDE to provide feedback and ease debugging. While testing can be valuable, it can be costly, Daka et al. found developers spent a substantial time writing tests [40]. The returns from testing can be especially low for trivial code that can take several times longer to test than create. For this reason, I have limited tests to all important areas or complex areas of the program.

The development environment is the system of programs where this workflow takes place. This project is configured to work well with a particular Integrated development environment (IDE); for instance, the IDE will be able to start the application with a debugger attached, allowing the user to step through the code in the editor. Code quality is maintained in this repository with several automated tools. Static code analysis tools enforce a consistent code style and check for typing errors. I have configured these tools to integrate with the IDE, giving immediate feedback so issues can be addressed quickly. Furthermore, Git is configured with pre-commit hooks, so these tools must run and validate the code before every commit. Furthermore, these tools can be integrated with CI/CD pipelines, which is particularly useful for collaborative projects.

While all of the tooling for this project improves many aspects of development, this may be difficult to set up and provides many opportunities for discrepancies to emerge between different setups. For this reason, I have created a development container. This container fully specifies every tool and the version to be installed. This containerised environment makes it easy to set up the environment and consistent each time. Containers for development are rapidly growing in interest [41].

For the sake of consistency, the application itself has been configured with a dependency management and packaging tool. All of the libraries the application relies upon are specified in the standard ‘pyproject.toml‘ and the tool manages their installation in an isolated virtual environment. This process makes it easy to distribute this application and avoids inconsistencies on different computers.

4.4 Functionality and Usage

The application's functionality revolves around running reinforcement learning simulations and allowing the user to interact with and analyse the effects of different options. These simulations encompass two main units: agent and dynamics. The agent unit encompasses the functionality for learning and decision making while the dynamics unit encompasses a grid world Markov decision process. These units are interchangeable and configurable by the user dynamically through controls in the user interface and through a configuration file `config.toml`. The results of these learning experiments are expressed by the application in several ways, such as a state-by-state view, information displays, and graphs.

4.4.1 Getting Started

The project is set up as a standard distribution package, and poetry is the build backend. While you may also use poetry as the build front-end if you have it installed, it may be excessive if you are only interested in starting the application.

Pipx is a simple build frontend that should be familiar to you if you have experience with pip. Instructions on how to install Pipx for your platform can be found here. Once you have installed pipx the application can be started with the following command:

```
python3 -m pipx run --spec . start
```

For the first run this command will take longer than usual as it sets up the application. You can find more information about this application and how to start it in the "readme" file in the code folder.

4.4.2 User Interface

The user interface is not only a significant component of the project's functionality but also the user's window into the entire application. When starting the application, you will be greeted by a window that is composed of into three section. At the top, there is a section which contains the main controls for selecting the most significant options of the primary simulation; such as the agent and dynamics. If the agent is off-policy, you can select its own exploration strategy.

In the middle, there is a tabbed area. The first tab is the primary display this display visualises the current state of the main simulation. The second tab displays the reward history of the simulation as a graph. The third and fourth tabs can run auxiliary simulations in the background to examine the effect of different hyperparameter choices. The third tab shows graphs of the effects of varying each hyperparameter and the reward achieved with 95% confidence bounds. Finally, the fourth tab controls a random search procedure. Starting this procedure searches for the best parameter choices for minimising regret. All of the graphs allow you to save their contents to a file of your choosing.

The main display on the first tab visualises the current state of the primary simulation, each dynamics are based around a grid world and the display visualises this with a grid of cells, each cell representing a possible position the agent can occupy. The function of this grid will vary based on the display mode, In the initial default display mode a robot icon represents the agent's location, flags represent the agent's goals and stop icons represent obstacles. In other Display modes arrows represent the agent's actions in each state and a colour scale is used to represent the agent's interpretation of value. On this colour scale hues vary from red to green, where red is the least valuable state and green is the most valuable state or state-action.

At the bottom are specific controls specific to the main simulation and its visualisation. For example, there are controls for selecting the display mode and allowing the user to step through the simulation manually or automatically.

Figure 4.3: "Current State" tab with optimal state-value visualisation of the cliff environment

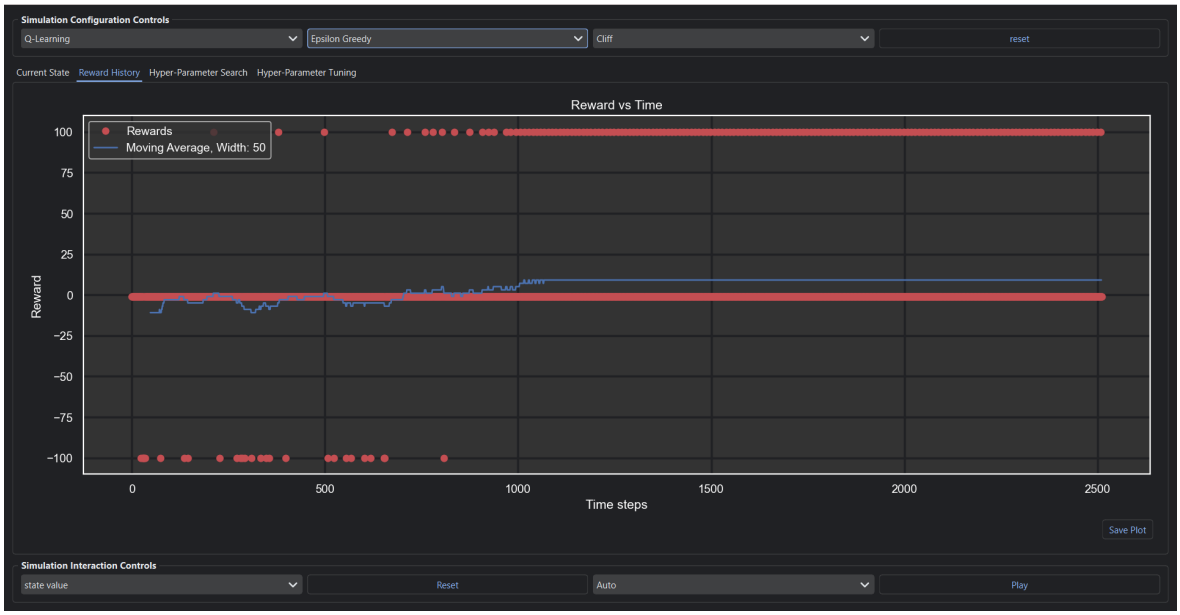


Figure 4.4: "Reward history" tab with Q-learning agent in the cliff environment

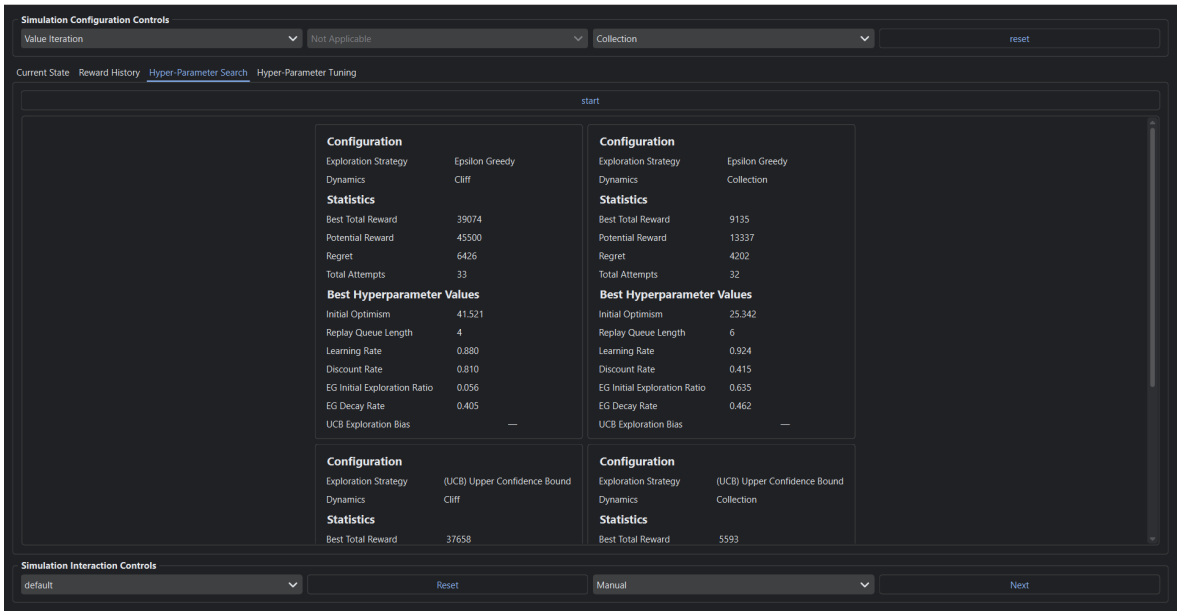


Figure 4.5: "Hyper-parameter Search" tab showing search results

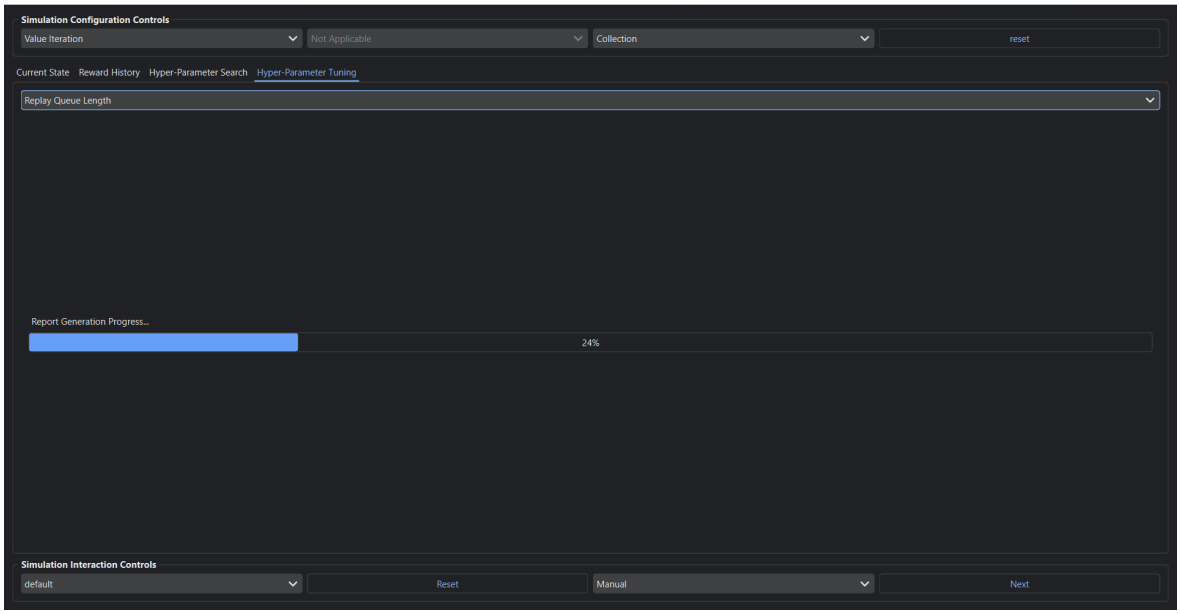


Figure 4.6: "Hyper-parameter Tuning" tab loading screen

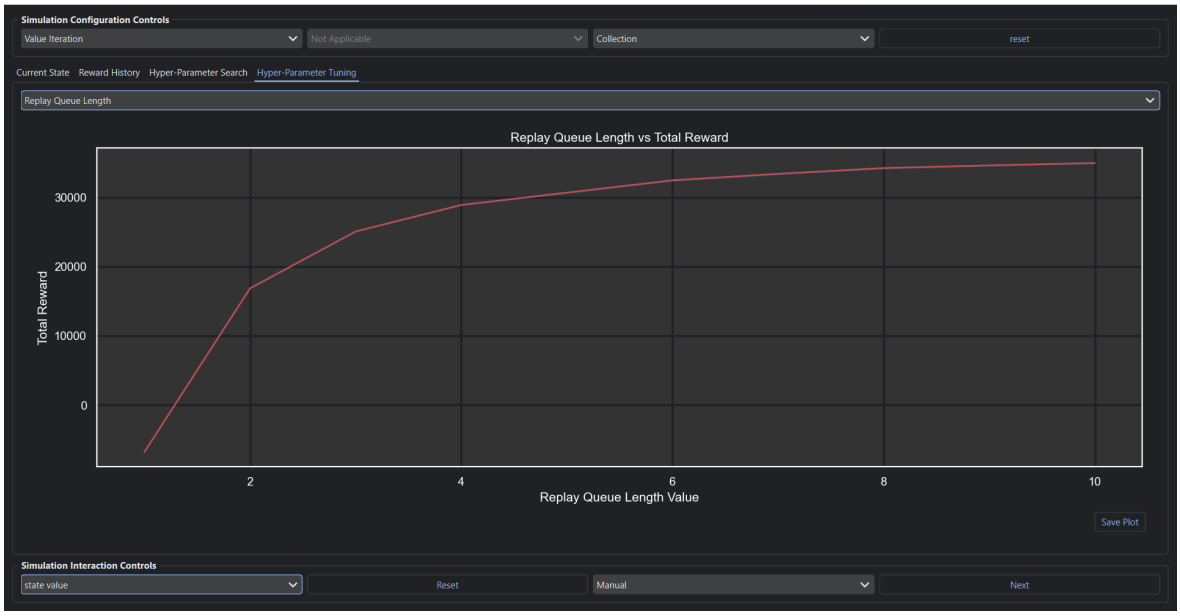


Figure 4.7: "Hyper-parameter Tuning" tab showing the results of tuning the reward queue length

Chapter 5: **Exploration Strategy Effectiveness**

Q-learning is an off-policy process, which means that the policy that the policy that it is learning and improving is not necessarily the same as the policy it is following during exploration. By this, the actions chosen by a Q-learning agent may be independent of its current beliefs of the values of each action. Off-policy learning has several advantages, it may explore more fairly and completely than an on-policy approach. An on-policy approach may be incentivised to explore thoroughly if its initial estimates are particularly optimistic. However, in a stochastic environment where rewards are random with a large degree of spread, it may still be difficult for an on-policy technique to properly assess the rewards before falling into an in-optimal pattern.

On-policy reinforcement learning algorithms such as SARSA do have many advantages in deep reinforcement learning (DRL). In DRL a deep artificial neural network is used to estimate the Q-values [42]. DRL can help generalise reinforcement learning to larger state spaces than is possible with tabular methods. Typically in this context, fully off-policy methods do not perform as well as on-policy or hybrid methods [43]. since off-policy methods suffer from instability.

In this project, I have focused on the off-policy Q-learning method. This chapter details the performance and effects of different exploration strategies. An exploration strategy is the policy that the agent follows during training as opposed to the learnt policy. This exploration strategy may be any method to select an action, however, to maintain the guarantees of convergence this policy must reach all states infinitely often 3.4.3. While a random action section, a random walk, does meet the convergence criteria it is not efficient in terms of rewards received or information gained. As an improvement in these measures, more sophisticated strategies have been developed that take into account many factors.

5.1 ε-Greedy

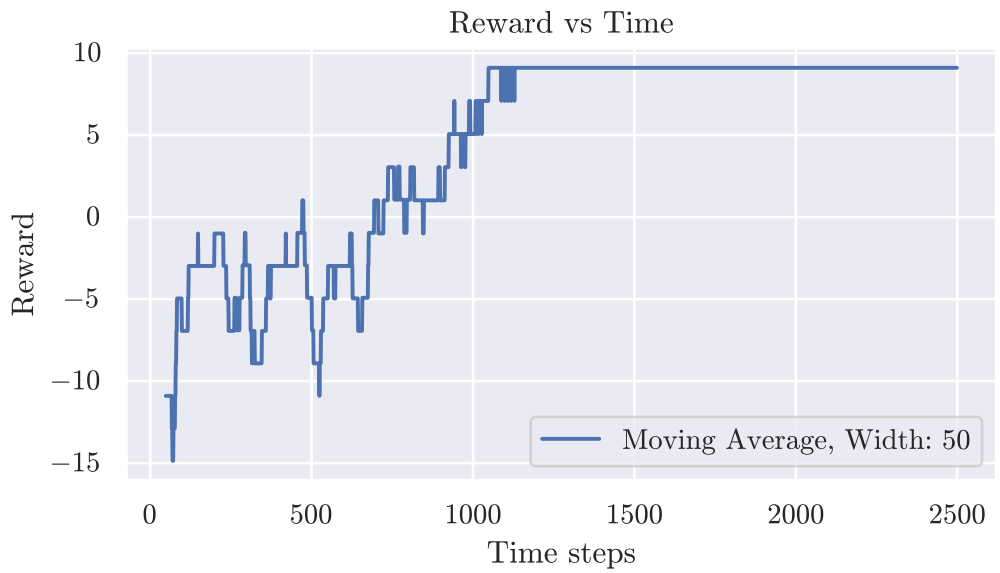


Figure 5.1: Reward history for an ε-Greedy exploration strategy instance

The ε-Greedy is the most straightforward strategy implemented in this project and it acts as a good baseline. In its basic form, there is a parameter ε that controls the probability of the agent acting in two modes. The first and typically most common mode is where the agent selects the best action based on the current Q-estimates. In this first mode, the agent is essentially operating in an on-policy manner. In the second mode, the agent selects an action at random 3.4.3.

This epsilon parameter controls the ratio between the agent's exploration and exploitation. Having high exploitation can garner more rewards in the short term but this has a large opportunity cost of not finding potentially lucrative strategies. On the other hand, if you are already following an optimal policy an exploratory action would not be optimal, and in the case of the cliff environment, it can be potentially devastating. In the beginning, higher exploration may be necessary to avoid local minima, but once the policy has started to converge higher exploitation can be more rewarding. To address this issue the ε parameter can be dynamically adjusted during training. This project implements the traditional discounting with a fixed hyperparameter however more advanced reward-based techniques have been found more successful [44].

Despite ε-Greedy's simplicity figure 5.1 shows how competitive this strategy can be when tuned to the environment. A combination of optimistic initial estimates and forced exploration can be extremely efficient. Unfortunately, to achieve these extraordinary results, each hyperparameter needs to be "just right" for the environment; this doesn't generalise easily and can be thought of as a form of over-fitting. I cover the effects of the exploration decay rate parameter in more detail in section 5.4.3.

5.2 Upper Confidence Bound

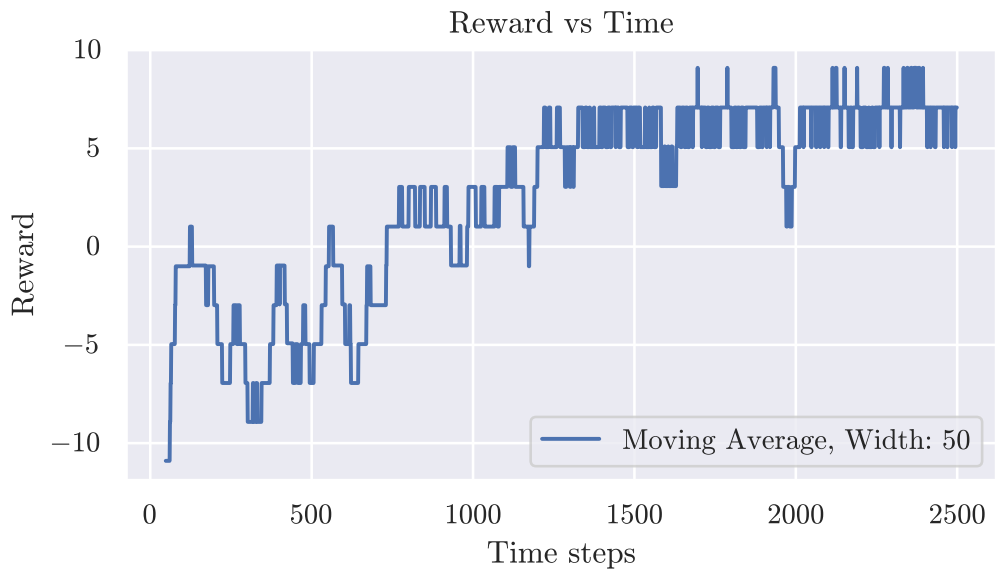


Figure 5.2: Reward history for an UCB exploration strategy instance

The upper confidence bound (UCB) exploration strategy uses the current Q value estimates to direct its decisions however instead of using randomness to dictate exploration, it uses an informed method. UCB prioritises actions that have the highest potential for reward. The potential of value is determined from two factors the current value estimate and a measure of confidence in the value estimate. The key idea behind this confidence measure is that it relates to the number of samples that have been made of the state action [18]. This relationship is defined in the following rule:

$$A_t \doteq \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{5.1}$$

[18]

where A_t represents the action chosen at time t, and $N_t(a)$ represents the number of samples of action a have been taken at the current time.

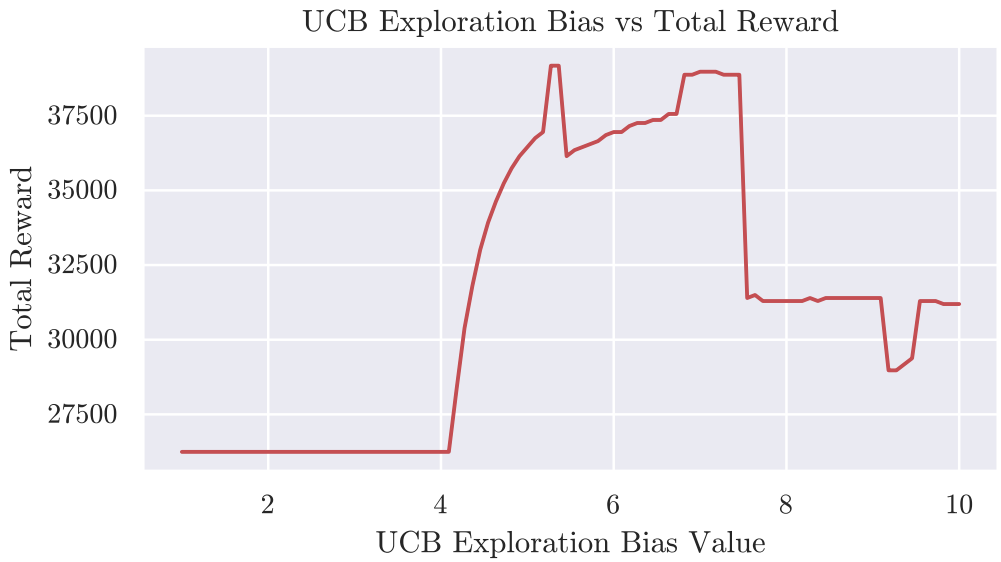


Figure 5.3

In essence, this equation is optimistic as it will never underestimate or fail to explore an action. It also provides a natural balance between exploration and exploitation; Furthermore, Without a discount factor, this strategy naturally tapers off exploration over time. Instead of even exploration like ε-Greedy; UCB avoids redundant exploration in areas that have already been covered. However, UCB is not without hyper-parameters. c controls the exploration balance; higher values encourage more exploration. Like ε from ε-greedy, this parameter can have a profound impact on the strategy's performance. Figure 5.3 shows how a change in this balance can almost double the cumulative reward.

Figure 5.3 also illustrate how the effects of c can be both gradual and abrupt. This stems from the relationship between the agent and environment and how they are both deterministic. Generally, the gradual improvement can be found in slight changes in the amount of exploration whereas abrupt changes happen if the agent discovers shortcuts or not. From my experience, values $c < 1$ operate in an on-policy fashion and $c > 10$ over explore.

ϵ greedy is able to get extraordinary results at the cost of under-exploration. UCB risks the opposite issue; it may over-explore. This is a consequence of its uncertainty term 5.1, it's optimistic and always growing, This guarantees the strategy will always explore enough to fully converge. Unfortunately, it is never truly done exploring. The $\ln t$ operator helps to mitigate this issue over time. However, at the beginning, the UCB is prone to re-exploring areas over new ones. I believe this behaviour explains some of the oscillations in figure 5.2. In UCB's favour, it considers the estimated value in all of its exploration and has received the least penalty of any strategy.

5.3 Model-Free Best Policy Identification

Figure 5.4: Reward history for an MF-BPI exploration strategy instance

Model-Free Best Policy Identification (MF-BPI) is a state-of-the-art method that has demonstrated an improvement in exploration over its contemporaries such as UCB [26]. This exploration strategy optimises for the best policy identification that was introduced in section 3.5. Unfortunately, the characteristic time T^* metric is non-convex [29] and requires extensive knowledge of the MDP's underlying distribution. Model-Free Best Policy Identification addresses both of these issues to make this approach practical.

This exploration strategy uses an allocation vector ω that distributes the search effort across the state-action space, however instead of optimising this with respect to the characteristic time T^* metric they derive a new tight convex upper bound $\tilde{U}$.

$$\tilde{U}(\omega) \doteq \max_{s,a \neq \pi^\star(s)} \frac{H(s,a)}{\omega(s,a)} + \frac{H}{\min_{s'} \omega(s', \pi^\star(s'))} \text{ [26]} \tag{5.2}$$

This new metric overcomes the first challenge however in the equation 5.2 it is evident that it is still dependent on the optimal policy, and the optimal value function; something an agent would not have access to. Therefore to make this practical the MF-BPI strategy estimates these values by maintaining an ensemble of value tables, from which it can estimate their variance. MF-BPI also uses a bootstrapping technique to ensure that the uncertainty decreases over time. Due to this bootstrapping, they claim that MF-BPI does not require forced exploration to ensure all state-action pairs are visited infinitely often however their implementations provided did include forced exploration, Garivier et al. said "the forced exploration step are rarely useful, but in some cases really necessary and not only for the theorems" [28]

MF-BPI strategy requires three hyperparameters: the ensemble size, the exploration parameter and what I call error sensitivity. however, the strategy does not seem to require extensive tuning and performs well with the provided values. While this strategy has not performed quite as well as a perfectly tuned ε-greedy it is nearly as good without any tuning. Notably, this strategy's regret is competitive despite not being the objective. Overall MF-BPI's

incredible generalisation impresses me the most.

5.4 Hyperparameter Selection

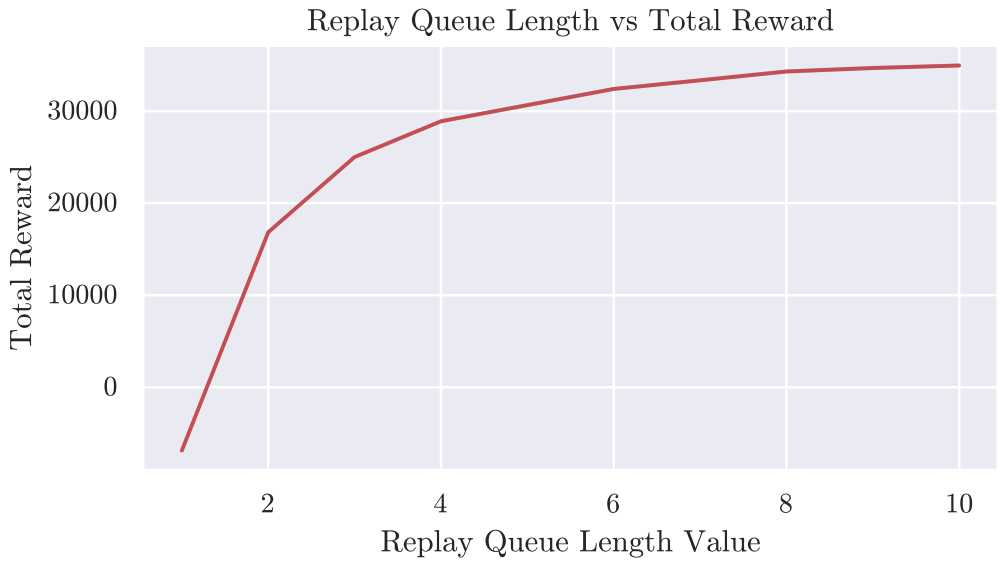


Figure 5.5

Q-learning itself and each exploration strategy require some parameters which are set outside of the learning process. Figure 5.5 Demonstrates how hyper-parameter values can have a profound impact on the effectiveness of the agent. However, selecting the best parameters is not straightforward. The first difficulty is that the best parameter choice depends on the agent and the environment that it operates on so they are not transferable and must be found through experimentation. The second difficulty is that the hyper-parameters are not independent different choices may compound or interfere with each other. Finally building on the second issue is the number of parameters, for Q-learning with ε-greedy there are 6 tunable parameters, This makes visualising the whole hyper-parameter space impossible.

For this reason, the application contains multiple systems for finding the appropriate values for these hyper-parameters. Firstly the hyper-parameters are configurable in the application's main configuration file. This configuration drives the parameters used by the main simulation and the defaults used in the report system. The hyper-parameter report system runs multiple simulations in the background and collates their results. This system allows the user to visualise and compare the effects of varying each parameter individually.

This report system overcomes the dimensionality visualisation issue by displaying a single axis-aligned slice of the space at a time. This approach is great for isolating the effects of a single parameter and fine-tuning parameters. Unfortunately, this approach cannot display the effects of any parameter combinations that do not coincide along one of the hyperparameter axes. Due to this limitation, the visualisations may display a local minimum and completely miss a global minimum.

A hyper-parameter search system is implemented to find the best baseline for fair comparisons. There are a few popular search strategies for hyperparameter optimization such as grid search, random search, and genetic algorithms [45]. Grid-search is a very popular approach for its advantages such as simplicity, uniformity and parallelisation [45]. however, it suffers from the curse of dimensionality [45] so it was not suitable for this application. Instead, the random search strategy has been implemented. This strategy is highly amenable to parallelisation, A number of workers are used and they pick the parameters at random and run simulations independently. In this manner, the random search strategy does not require orchestration of the parameters and is resumable, synchronisation is only necessary when collating results to avoid race conditions.

Since the performance is highly dependent on the environment; the same cliff dynamics have been used in all the graphs provided in this chapter. Furthermore, all of the episodes are equal in length so the cumulative reward achieved is comparable. An agent that directly follows the optimal policy without learning can achieve a cumulative reward of 45500 under the same conditions. This provides the upper bound on the agent's performance and the bar from which to determine regret 3.5. In the following sections, a Q-learning agent with the ε-greedy exploitation strategy is used for the evaluation. This strategy is stochastic and its variability affects the performance. To account for this influence many simulations are run for each configuration. Each graph plots the mean reward in red and a 95% confidence interval in blue. The following sections will cover an interesting subset of the hyper-parameters and discuss my findings.

5.4.1 Initial Optimism

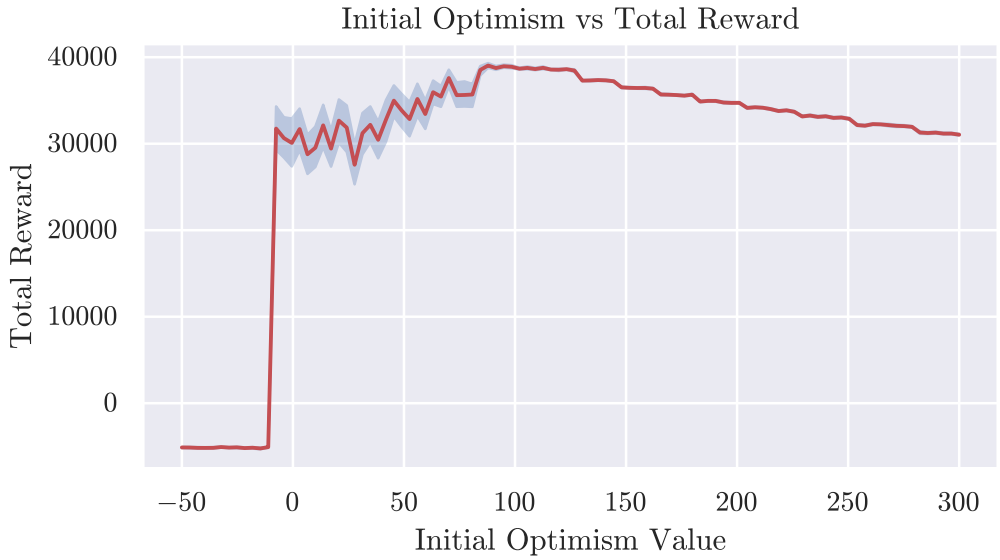


Figure 5.6

The initial optimism hyper-parameter determines the initial Q-value estimates. In essence, this is the agent's interpretation of the value of state actions before the agent has gained any experience of that state action. Since Q-learning is a bootstrapping method depending on parameters such as the learning rate, the discount factor and the rewards provided initial biases may persist through more or less updates. This can be an opportunity to provide information a-priori, value tables from previous agents can provide a head-start [46]. While realistic initial estimates are beneficial, figure 5.6 shows how excessive values can be cost performance.

When the initial estimate is larger than the actual value of each state this is optimistic. Optimism encourages the agent to try unknown states over known states since the optimistic initial values will be strictly better than the more realistic derived value. On the other hand, pessimistic values act oppositely, discouraging exploration as known states seem better than unknown states. You can see this demonstrated in figure 5.6, in the pessimistic region (left of 100) the variability is higher because the agent relies entirely on the random forced exploration.

Increasing optimism gradually decreases performance as the agent over-explores and neglects exploitation. The effect is similar for pessimism however the performance hits a cliff after the initial estimates become negative. At this point, the agent gets stuck repeatedly choosing the same state or states. I believe this may be a consequence of several factors. Firstly the other parameters aren't configured for this degree of pessimism, so the agent can't reach the goal based on forced exploration alone, and the pessimism stops it from exploring enough to find it. Secondly, the discount factor works in reverse when all values are negative it encourages taking longer. Finally, Q-learning's maximisation bias may be inhibiting its ability to propagate these negative updates correctly [47, 18].

5.4.2 Learning Rate α

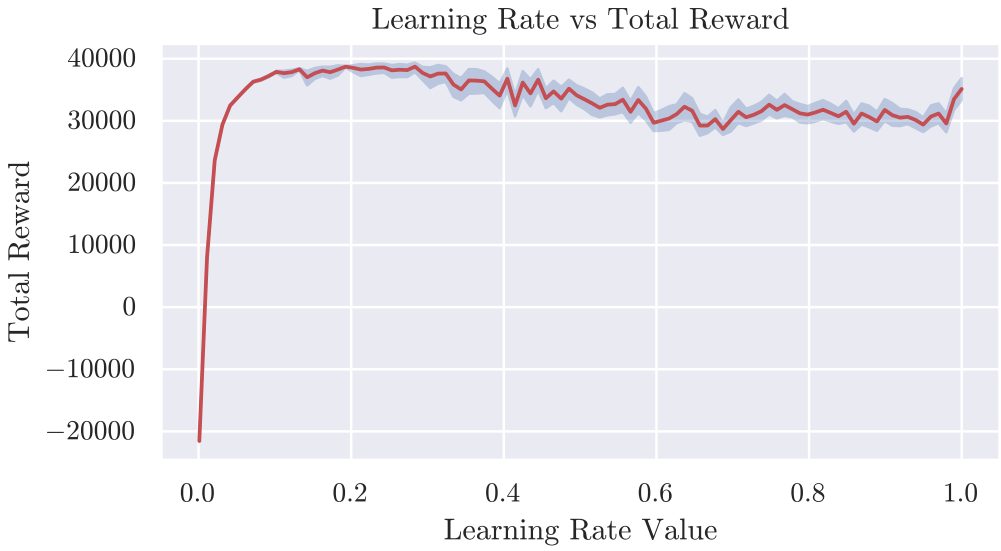


Figure 5.7

The learning rate α is another hyperparameter of Q-learning 3.10. This parameter controls how much each new TD estimate will update the current estimates. For stochastic environments getting this value right is crucial, If set too high the variability from the individual observations could lead to instability and the Q-table may not even converge at all. The cliff dynamics that are used in this demonstration are completely deterministic. The larger the learning rate the fewer updates are necessary for the values to converge. However, this can be a double-edged sword because, in the current configuration, optimistic initial estimates drive the majority of the exploration. Therefore smaller updates encourage more thorough but slower exploration whereas larger updates are more rapid but unstable exploration.

Figure 5.7 shows the balance of learning rate and how this impacts performance. In the range of 0.2 to 1, there is a plateau where the stability and speed of convergence are in balance. However, as the learning rate approaches zero the performance diminishes rapidly as the time taken to learn anything extends. In this region, the exploration period inhibits the agent's ability to exploit the environment.

5.4.3 ε-Greedy Decay Rate

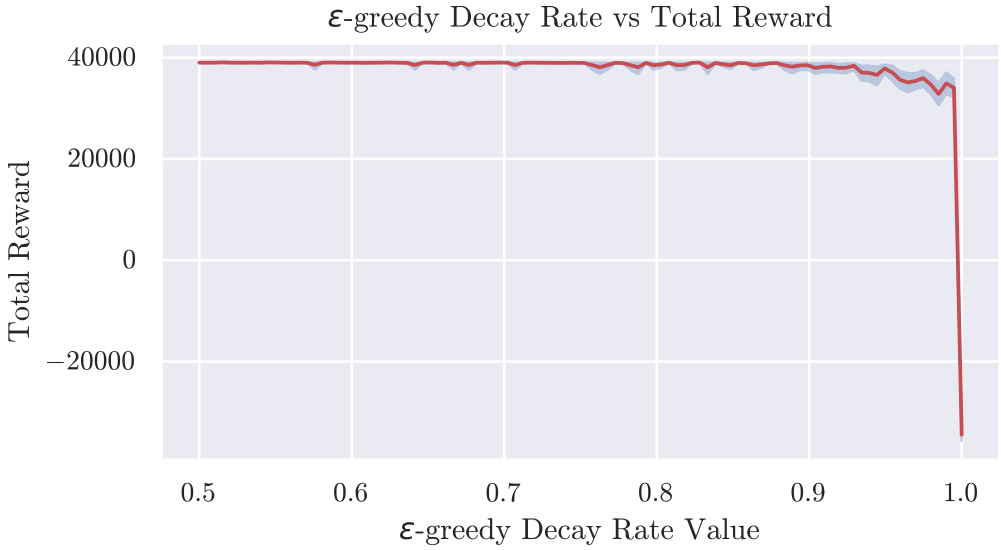


Figure 5.8

The ε-greedy decay rate (EGDR) parameter is an important and unique parameter. Figure 5.8 demonstrates how just a minuscule amount of decay has a substantial improvement on the agent's regret completely changing the agents' performance. Without EGDR the cumulative reward was negative. This follows from how ε-greedy acts in a partially off-policy, in the cliff dynamics the optimal policy minimises the distance travelled and therefore travels right next to the cliff face. This becomes problematic when ε-greedy introduces its off-policy exploration actions, which randomly cause the agent to select the action to jump off the cliff incurring a large penalty.

An on-policy method may account for the risks of selecting exploratory actions in its value estimates encouraging the agent to give the cliff a wide berth or choose to limit the exploratory actions. However, Q-learning is an off-policy technique so Another approach is necessary. An alternative is to reduce the exploration rate once the agent has had the opportunity to learn. This application has implemented exponential decay for the exploration ratio ε. This exponential decay is controlled via two hyperparameters the initial value and the decay rate. In this procedure, the exploration ratio ε can be computed as:

$$\varepsilon_t = \lambda^t \cdot \varepsilon_0 \tag{5.3}$$

Where ε_t is the exploration ratio at the time step t and λ represents the EGDR.

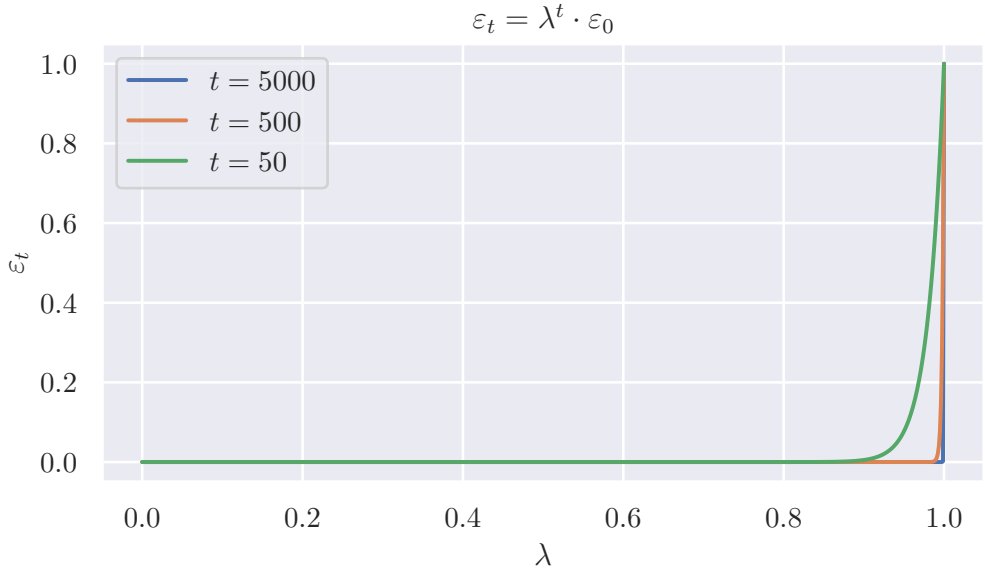


Figure 5.9

EGDR has a very striking form as there is almost no variability in performance until approximately > 0.99 where the performance almost immediately plummets. This sharp corner follows from EGDR's compounding effects on the exploration probability. Each simulation has 5000 time steps, figure 5.9 demonstrates how with this number of time steps all but the most extreme values will not be reduced to near zero. The figure 5.9 also shows how the EGDR parameter can control how long the agent has forced exploration, on this graph a decay rate of $\lambda = 0.99$ would have a 60% exploration probability by time step 50 whereas $\lambda < 0.6$ would have a probability if less than $1 \cdot 10^{-12}$, in other words, the figure 5.9 shows that depending on the EGDR the exploration phase may take more or less than 50 time-steps.

I derive this formula to predict the length of exploration:

$$t_{\text{exploration}} = \log_\lambda(\frac{\varepsilon_{\min}}{\varepsilon_0}) \tag{5.4}$$

$$\varepsilon_{\min} = 1 - (1 - \alpha)^{\frac{1}{d}} \tag{5.5}$$

Where $\varepsilon_{\min}$ is the minimum chance of exploration to consider the agent is still exploring in each time step. This probability also compounds so you can use the equation 5.5 to compensate for the remaining duration of the episode, where α is the probability of an exploratory action in the duration d (number of time-steps).

5.4.4 Discount Rate γ

Figure 5.10: Full range discount rate graph.

The discount rate hyperparameter affects how much future value propagates backwards. The larger the discount rate the more future rewards are considered and the more far-sighted the agent becomes, conversely the smaller the discount rate the more short-sighted the agent's actions become 3.2. When the discount rate is $\gamma \approx 0$ the agent's actions have no foresight. Unfortunately, the cliff environment's reward is sparse and figure 5.10 shows how having some foresight is necessary. without it, the agent is essentially lost.

On the other end of the spectrum when the discount factor is large $\gamma \approx 1$ this can encourage far-sighted behaviour such as waiting. However, for the cliff environment, far-sighted behaviour was not a concern, instead, the regret observed around $\gamma > 0.8$ in figure 5.10 is explained by another factor. The bellman equations form a contraction mapping where γ is the contraction factor [48]. The result is that the discount factor is inversely proportional to the rate of convergence. Figure 3.2 shows how at the extreme the slowed convergence noticeably increases the agent's regret as the time the agent takes to learn increases.

5.5 Conclusions

In this chapter, we have discovered that each of these three model-free exploration strategies is capable of extremely efficient exploration and exploitation. It is clear that under the right conditions, regardless of a strategy's complexity or sophistication it can perform similarly. Furthermore, each exploration strategy's performance can be incredibly sensitive to its conditions such the the environment and hyperparameters. This sensitivity can make it easy to fall into the "tune-and-report" trap. Instead of focusing on concrete results, reporting on general trends may be more informative.

The parameter space has a high dimensionality and is heterogeneous so where possible a systematic search over the parameter space is advisable. Where this is not possible it seems a small, relative to rewards, amount of optimism is beneficial. A small but not negligible learning rate like 0.1 is a good place to start. And finally, it seems the discount rate should be roughly proportional to how sparse rewards are.

The three strategies explored in this report vary from the simple to the state-of-the-art. On this scale the more advanced models demonstrated better generalisation, they perform better out-of-box and have less tunable parameters. However, these more sophisticated models can exhibit unusual behaviour and actions can be difficult to explain. It seems where generalisation to varied environments is important; advanced strategies are superior. However, if your environment is fixed a simpler strategy with tuned parameters can have similar performance and improved explainability.

Chapter 6: **TODO - Project Analysis**

Critical analysis and Discussion (10%): A discussion of actual project achievements and how successful the project was. Clear evidence of reflection on the project process, its difficulties, successes and future enhancements. Any conclusions or results analysed or discussed appropriately;

would have been great to implement a wider range of environments including stochastic to challenge the strategies further

Bibliography

[1] J. A. Bagnell, "Robust supervised learning," in *AAAI*, 2005, pp. 714–719.

[2] S. Ransbotham, D. Kiron, P. Gerbert, and M. Reeves, "Reshaping business with artificial intelligence: Closing the gap between ambition and action," *MIT Sloan Management Review*, vol. 59, no. 1, 2017.

[3] R. Tiwari, "Ethical and societal implications of ai and machine learning," 2023.

[4] C. Godfrey, "Legislating big tech: The effects amazon rekognition technology has on privacy rights," *Intell. Prop. & Tech. LJ*, vol. 25, p. 163, 2020.

[5] R. K. Indla, "An overview on amazon rekognition technology," 2021.

[6] V. Srivastava, "Ethics of facial recognition within businesses."

[7] R. S. Fleischer, "Bias in, bias out: Why legislation placing requirements on the procurement of commercialized facial recognition technology must be passed to protect people of color," *Public Contract Law Journal*, vol. 50, no. 1, pp. 63–89, 2020.

[8] V. Koniakou, "From the "rush to ethics" to the "race for governance" in artificial intelligence," *Information Systems Frontiers*, vol. 25, no. 1, pp. 71–102, 2023.

[9] I. Gabriel, "Artificial intelligence, values, and alignment," *Minds and machines*, vol. 30, no. 3, pp. 411–437, 2020.

[10] L. Weidinger, J. Mellor, M. Rauh, C. Griffin, J. Uesato, P.-S. Huang, M. Cheng, M. Glaese, B. Balle, A. Kasirzadeh *et al.*, "Ethical and social risks of harm from language models (2021)," *arXiv preprint arXiv:2112.04359*, 2021.

[11] M. Young, M. Katell, and P. Krafft, "Municipal surveillance regulation and algorithmic accountability," *Big Data & Society*, vol. 6, no. 2, p. 2053951719868492, 2019.

[12] L. Floridi, J. Cowls, M. Beltrametti, R. Chatila, P. Chazerand, V. Dignum, C. Luetge, R. Madelin, U. Pagallo, F. Rossi *et al.*, "Ai4people—an ethical framework for a good ai society: opportunities, risks, principles, and recommendations," *Minds and machines*, vol. 28, pp. 689–707, 2018.

[13] J. Whittlestone, K. Arulkumaran, and M. Crosby, "The societal implications of deep reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 70, pp. 1003–1030, 2021.

[14] N. Wang, Z. Li, X. Liang, Y. Hou, A. Yang *et al.*, "A review of deep reinforcement learning methods and military application research," *Mathematical Problems in Engineering*, vol. 2023, 2023.

[15] A. T. Azar, A. Koubaa, N. Ali Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, A. Ammar, B. Benjdira, A. M. Khamis, I. A. Hameed *et al.*, "Drone deep reinforcement learning: A review," *Electronics*, vol. 10, no. 9, p. 999, 2021.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[17] S. P. Meyn and R. L. Tweedie, *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

[18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[19] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[20] J. E. Staddon and D. T. Cerutti, "Operant conditioning," *Annual review of psychology*, vol. 54, no. 1, pp. 115–144, 2003.

[21] H. Shteingart and Y. Loewenstein, "Reinforcement learning and human behavior," *Current opinion in neurobiology*, vol. 25, pp. 93–98, 2014.

[22] K. You, M. Long, J. Wang, and M. I. Jordan, "How does learning rate decay help modern neural networks?" *arXiv preprint arXiv:1908.01878*, 2019.

[23] E. Even-Dar, Y. Mansour, and P. Bartlett, "Learning rates for q-learning." *Journal of machine learning Research*, vol. 5, no. 1, 2003.

[24] S. Jordan, Y. Chandak, D. Cohen, M. Zhang, and P. Thomas, "Evaluating the performance of reinforcement learning algorithms," in *International Conference on Machine Learning*. PMLR, 2020, pp. 4962–4973.

[25] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song, "Assessing generalization in deep reinforcement learning," *arXiv preprint arXiv:1810.12282*, 2018.

[26] A. Russo and A. Proutiere, "Model-free active exploration in reinforcement learning," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=YEtstXIpP3

[27] A. Al Marjani, A. Garivier, and A. Proutiere, "Navigating to the best policy in markov decision processes," *Advances in Neural Information Processing Systems*, vol. 34, pp. 25 852–25 864, 2021.

[28] A. Garivier and E. Kaufmann, "Optimal best arm identification with fixed confidence," in *Conference on Learning Theory*. PMLR, 2016, pp. 998–1027.

[29] A. Al Marjani and A. Proutiere, "Adaptive sampling for best policy identification in markov decision processes," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7459–7468.

[30] H. Van Vliet, H. Van Vliet, and J. Van Vliet, *Software engineering: principles and practice*. John Wiley & Sons Hoboken, NJ, 2008, vol. 13.

[31] S. Raschka, J. Patterson, and C. Nolet, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence," *Information*, vol. 11, no. 4, p. 193, 2020.

[32] A. Leff and J. T. Rayfield, "Web-application development using the model/view/controller design pattern," in *Proceedings fifth ieee international enterprise distributed object computing conference*. IEEE, 2001, pp. 118–127.

[33] T. Ollsson, D. Toll, A. Wingkvist, and M. Ericsson, "Evolution and evaluation of the model-view-controller architecture in games," in *2015 IEEE/ACM 4th International Workshop on Games and Software Engineering*. IEEE, 2015, pp. 8–14.

[34] C. Anderson, *The Model-View-ViewModel (MVVM) Design Pattern*. Berkeley, CA: Apress, 2012, pp. 461–499. [Online]. Available: https://doi.org/10.1007/978-1-4302-3501-9_13

[35] W. Haoyu and Z. Haili, "Basic design principles in software engineering," in *2012 Fourth International Conference on Computational and Information Sciences*. IEEE, 2012, pp. 1251–1254.

[36] F. Mattisson, "Deep reinforcement learning a case study of alphazero," 2021.

[37] M. M. Morovati, F. Tambon, M. Taraghi, A. Nikanjam, and F. Khomh, "Common challenges of deep reinforcement learning applications development: An empirical study," *arXiv preprint arXiv:2310.09575*, 2023.

[38] "Conventional commits," https://www.conventionalcommits.org/en/v1.0.0/, (Accessed on 19/01/2024).

[39] G. M. Wiggins, G. Cage, R. Smith, S. Hitefield, M. McDonnell, L. Drane, J. McGaha, M. Brim, M. Abraham, R. Archibald *et al.*, "Best practices for documenting a scientific python project," 2023.

[40] E. Daka and G. Fraser, "A survey on unit testing practices and problems," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 201–211.

[41] M. Koskinen, T. Mikkonen, and P. Abrahamsson, "Containers in software development: A systematic mapping study," in *International conference on product-focused software process improvement*. Springer, 2019, pp. 176–191.

[42] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[43] M. Hausknecht, P. Stone, and O.-p. Mc, "On-policy vs. off-policy updates for deep reinforcement learning," in *Deep reinforcement learning: frontiers and challenges, IJCAI 2016 Workshop*. AAAI Press New York, NY, USA, 2016.

[44] A. Maroti, "Rbed: Reward based epsilon decay," *arXiv preprint arXiv:1910.13701*, 2019.

[45] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: a big comparison for nas," *arXiv preprint arXiv:1912.06059*, 2019.

[46] M. Korecki, C. Carissimo, and T. Lund, "artificial death: learning from stories of failure," in *ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference*. MIT Press, 2023.

[47] H. Hasselt, "Double q-learning," *Advances in neural information processing systems*, vol. 23, 2010.

[48] S. J. Majeed, M. Hutter *et al.*, "On q-learning convergence for non-markov decision processes." in *IJCAI*, vol. 18, 2018, pp. 2546–2552.

Appendix A: **Diary**