Exploration by Random Network Distillation in Sparse Reward MiniGrid Environments

Sam Walsh

Master of Science in Artificial Intelligence
The University of Bath
2024

Exploration by Random Network Distillation in Sparse Reward MiniGrid Environments

Submitted by: Sam Walsh

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Master of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

Exploration methods have been at the heart of reinforcement learning research since the inception of the field. Recent research has utilised intrinsic motivation heavily as an additional reward signal to guide exploration in sparse reward environments, with promising results. Investigating intrinsic motivation, this project conducts a systematic analysis of Random Network Distillation (RND) in MiniGrid environments characterized by sparse or absent rewards. It also evaluates RND's ability to generalise well in sparse reward environments. It finds that although RND suffers from the detachment problem and vanishing intrinsic rewards, it nonetheless outperforms baseline methods in these scenarios and shows promise when generalising to new environments.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor Jessica Nicholson for her guidance and advice over the past months. Also, thanks go to my friends and family who encouraged me to undertake this MSc. Finally I would like to thank my wife, Jane; without her unwavering support this would not have been possible.

Chapter 1

Introduction

1.1 Problem Description

Reinforcement Learning encompasses various methods for solving problems modelled as Markov Decision Processes (MDPs). It is concerned with the interaction of an agent and its environment, with the agent learning to perform actions that maximise the reward it receives. Many important real-world problems can be effectively addressed by reinforcement learning, however, a key problem is how to design a reward function for the agent. Ideally, a goal could simply be specified, with the agent working out the best way to accomplish it. In this case though, the agent has no intermittent rewards to guide it toward the goal. This is the problem of sparse reward.

Exploration is useful in reinforcement learning to allow an agent to find sparse rewards in an environment. Exploration methods have been at the heart of reinforcement learning research since the inception of the field. Recent research has utilised intrinsic motivation heavily as an additional reward signal to guide exploration in sparse reward environments, with promising results. The field of intrinsically motivated reinforcement learning is still nascent, with no consensus on the best approach to the sparse reward problem.

1.2 Research Objectives

This project focuses on Random Network Distillation (RND), proposed by Burda et al. (2018), a method of generating intrinsic reward which encourages a reinforcement learning agent to explore an environment. The project aims to:

- Situate RND within the wider intrinsic motivation literature.
- Evaluate the exploration and performance of RND in sparse reward environments, in comparison to baseline algorithms.
- Evaluate the exploration of RND in no reward environments, in comparison to baseline algorithms.
- Evaluate the ability of RND to generalise, using knowledge from a simple setting to improve performance on a more complex one.

1.3 Motivations

RND has many advantages over other intrinsic motivation algorithms. It solves the "Noisy-TV" problem (described in Chapter 2) experienced by state-count and prediction-based methods while avoiding the computationally demanding aspects of other approaches such as Go-Explore (Ecoffet et al., 2021) and Agent57 (Badia et al., 2020a). It integrates simply into well established reinforcement learning algorithms such as Proximal Policy Optimisation, making it well suited to a wide range of problems, and it is compatible with modern distributed approaches. Despite these advantages, there are relatively few investigations into RND. This project seeks to provide a novel analysis of RND, taking a systematic approach rather than simply evaluating on benchmark environments.

1.4 Structure

- **Chapter 2** defines MDPs and the theory behind reinforcement learning formally, introducing key algorithms in traditional and deep reinforcement learning. Exploration and intrinsic motivation are expanded on in the literature review, with focus on state-of-the-art methods, including RND, along with environments they are suited to and problems they face.
- **Chapter 3** details the environmental and experimental design for testing RND, how experiments will be empirically evaluated and details of implementation.
- **Chapter 4** analyses the results of the experiments both quantitatively and qualitatively, using these results to draw inferences about the behaviour of RND agents.
- **Chapter 5** provides a summary of the experimental findings with reference to the objectives above, suggesting improvements and providing direction for further investigations.

Chapter 2

Background and Literature Review

This chapter formally introduces reinforcement learning and the settings in which it is applicable, going on to describe some of the key algorithms in the field which are applied in subsequent chapters. Exploration and intrinsic motivation in reinforcement learning are then studied to provide background and situate this investigation of RND within the wider literature.

2.1 Reinforcement Learning Background

2.1.1 Markov Decision Processes

Reinforcement learning is formally a method for finding solutions to MDPs. These are characterised by the following five-tuple:

$$MDP = \langle S, A, T, R, \rho(s_0), \gamma \rangle$$

Where:

- S is a set of states (finite or infinite), representing an environment.
- A is a set of actions (finite or infinite) which can be taken in the environment.
- T is the transition probability. $T(s'|s, a)$ is the probability of transitioning into state s' if an action a is taken in state s.
- R is the reward function. $R(s, a)$ is the reward received immediately if an action a is taken in state s.
- $\rho(s_0)$ is the initial state distribution, used to generate the initial state s_0.
- γ is a discount factor ($\gamma \in [0, 1]$).

Importantly, MDPs adhere to the *Markov Property*. This states that transitions from one state to the next depend *only* on the current state and action, not on prior events.

A Partially Observable Markov Decision Process (POMDP) is a special case defined in the same way as an MDP, with two additional features:

$$POMDP = \langle S, A, T, R, \rho(s_0), \Omega, O, \gamma \rangle$$

Where:

- Ω is a set of observations received by the agent about the current state.
- O is the observation function $O(o|s', a)$, which represents the probability of receiving observation $o \in \Omega$ given that action a has just been taken and the new state is s'.

2.1.2 Reinforcement Learning Theory

Figure 2.1 shows the reinforcement learning paradigm. Here the agent is interacting with an environment modelled as an MDP. In each time-step an agent observes the current state and takes some action. The environment provides a reward to the agent and moves to the next state, based on it's transition probabilities. The process then continues iteratively. Many problems in robotics, game-playing, economics and other fields can be modelled as MDPs, and as such, the solutions provided by advances in reinforcement learning are incredibly valuable.

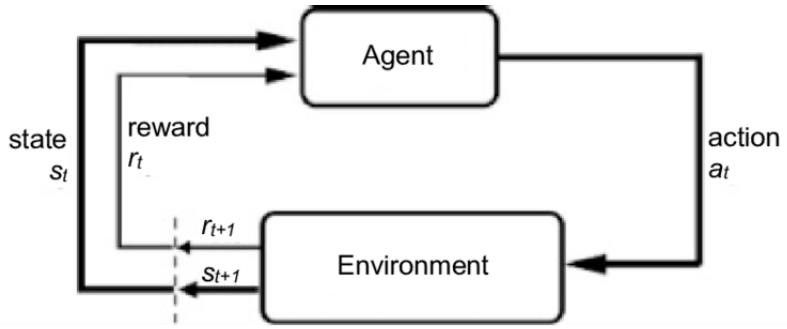

Figure 2.1: The reinforcement learning paradigm, Sutton and Barto (2018)

There is an important distinction to be made between model-based and model-free reinforcement learning. Model-based learning is so called because here the agent knows or learns a model of the transition probabilities in the environment, which can help it to choose the correct actions for a given state. Model-free reinforcement learning instead uses only the rewards available to guide it's actions. The methods used in this project are model-free approaches, subsequent sections will therefore focus on these.

Policies

A policy, π, is a function which maps states to actions: $a_t = \pi(s_t)$. This tells the agent how to behave in a given state. The policy can be either deterministic or stochastic. Reinforcement learning can be thought of as the process of learning an optimal policy. However, a goal is required for the policy to be optimised towards.

The Goal in Reinforcement Learning

As rewards are received from the environment after every timestep (though the reward may be zero), reinforcement learning can solve an MDP if it can maximise the overall rewards received over the steps in the environment. In the literature the total environment steps are referred to as a trajectory, episode or rollout. Concretely, the goal of the agent is to maximise it's cumulative reward over the trajectory. This (discounted) sum of rewards is the return for the trajectory, $R(T)$:

$$R(T) = \sum_{t=0}^{T} r_t$$

for the finite horizon case, and

$$R(T) = \sum_{t=0}^{\infty} \gamma^t r_t$$

where $0 < \gamma < 1$ for the infinite horizon case.

Given that the agent does not have perfect knowledge of the environment, it can only maximise it's *expected* return, $E[R(T)]$. Combining this with the policy, the goal of the agent is to learn an optimal policy π^* which maximises expected return over the trajectory:

$$\pi^* = \arg\max_{\pi} E_{T\sim\pi}[R(T)]$$

Value Functions and Bellman Equations

A value function, $V^\pi(s)$, is a way of determining the value of a state, s. This is the expected return for the trajectory taking s as a starting point and acting according to a particular policy until the end of the trajectory:

$$V^\pi(s) = E_{T\sim\pi}[R(T)|s_0 = s]$$

Similarly, an action-value function $Q^\pi(s, a)$ is a way of determining the value of a state-action pair, (s, a). This is the expected return for the trajectory with s as a starting point, taking action a (not necessarily the action dictated by the policy), then subsequently acting according to a particular policy until the end of the trajectory:

$$Q^\pi(s, a) = E_{T\sim\pi}[R(T)|s_0 = s, a_0 = a]$$

Both the value and action-value functions can be defined recursively. These Bellman Equations (Bellman, 1958) are incredibly useful in a variety of reinforcement learning methods as they present the problem in a way which suited to iterative computational methods. The value function can be reformulated as:

$$V^\pi(s) = E_{a\sim\pi}[r(s, a) + \gamma V^\pi(s')]$$

Where a is the action according to the policy, $r(s, a)$ is the reward received for that action and $\gamma V^\pi(s')$ is the discounted value of the next state. Similarly the action-value function can be reformulated as:

$$Q^\pi(s, a) = E[r(s, a) + \gamma E_{a'\sim\pi}[Q^\pi(s', a')]]$$

Where $\gamma E_{a'\sim\pi}[Q^\pi(s', a')]$ is the expected value of the next state-action pair (s', a') if action a' follows the current policy.

Traditional Reinforcement Learning Methods

Traditional methods are efficient in simple, finite environments with small state representations and action spaces. The following methods give a good grounding in applied reinforcement learning, but have for the most part been surpassed in recent years by deep methods, which are better suited to large, continuous state and action spaces.

A key distinction in various approaches to solving MDPs in model-free reinforcement learning is off-policy (value-based) methods compared to on-policy methods. In off-policy methods, such as Q-learning (Watkins and Dayan, 1992), the agent is not restricted to following the action dictated by the current policy, rather it learns how to improve it's policy by taking actions with the highest action-value function. Tabular Q-learning uses a Q-table (a table storing state-action pairs and their associated value). The agent takes actions in the environment according to the highest value action for the state it is in (occasionally taking a random action to explore), observes the next state and reward, and updates it's Q-table using the following rule derived from the action-value Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$$

Where α is the learning rate ($0 < \alpha < 1$) and $\gamma \max_a Q(s', a)$ is the estimated value of the next state if the highest value action is taken. Using this iterative approach, the agent converges on a policy which maximises return.

On-policy methods, such as REINFORCE (Williams, 1992), on the other hand, take actions according to their policy during a trajectory, then update the policy afterward based on the gradient of the policy function parameterised by θ:

$$\theta \leftarrow \theta + \alpha \cdot G \cdot \nabla_\theta \log \pi_\theta a|s$$

Where G is the total return from the current state onwards, and $\nabla_\theta \log \pi_\theta a|s$ is the gradient of the logarithm of the policy with respect to the policy parameters, evaluated at the action a in state s.

Deep Reinforcement Learning

Deep reinforcement learning applies deep learning techniques to MDPs. Much of the theory behind traditional methods still applies, but policies are instead represented by neural networks, which take states as input and output actions. Gradient descent and backpropagation can then be used to optimise the policy in a similar way to supervised learning. Two key deep reinforcement learning algorithms are described below, and applied in Chapter 4.

2.1.3 Deep Q-Networks (DQN)

Mnih et al. (2015) proposed DQN as a method for attaining human level or better performance on a wide range of Atari games. Their approach is theoretically very similar to tabular Q-learning, with the key difference being that the Q-table is represented by a neural network $\hat{q}_\theta$.

As with Q-learning, the agent takes actions in the environment according to the highest value action for the state it is in (occasionally taking a random action to explore) and observes the next state and reward. Rather than updating $\hat{q}$ straight away though, instead the tuple (s, a, r, s') is stored in a replay buffer D. Periodically a random batch from the buffer is selected for training. This reduces bias towards more recent experiences. The following is calculated for each tuple, j, in the batch:

$$y = \begin{cases} r_j & \text{if } s'_j \text{ is terminal} \\ r_j + \gamma \max_{a'} \hat{q}_2(s'_j, a', \theta_2) & \text{otherwise} \end{cases}$$
$$\hat{y} = \hat{q}_1(s_j, a_j)$$

Where y corresponds to $r + \gamma \max_a Q(s', a)$ in tabular Q-learning and $\hat{y}$ to $Q(s, a)$.

Gradient descent is then performed on the loss function for the batch: $\nabla_{\theta_1} L(\theta_1, \hat{y}, y)$ to update the parameters of $\hat{q}_1$. The loss function corresponds to the difference between the terms in tabular Q-learning, but often in DQN, Huber loss is used to mitigate large or erratic policy updates.

Another technique used here is to utilise two q-networks $\hat{q}_1$ and $\hat{q}_2$, which helps to improve the stability of learning as otherwise the the network would be updating towards a moving target during training. Periodically, the weights of $\hat{q}_2$ are updated so $\theta_2 = \theta_1$ to ensure the target network remains accurate. These are the key ideas behind DQN, however, full pseudocode is detailed in Appendix B.

2.1.4 Proximal Policy Optimisation (PPO)

PPO was proposed by Schulman et al. (2017) as a method for improving training stability in deep policy gradient algorithms. PPO is widely used today in many different settings due to its efficacy and suitability for paralellisation. PPO is an on-policy algorithm, and is part of a family of actor-critic methods in which the actor network learns a policy and the critic network learns a value function. Essentially PPO, along with all policy gradient approaches, works by iteratively increasing the probability of actions which increase return, and decreasing those that lead to lower return. It is important to note that because the policy is stochastic (the choice of action in a given state is defined probabilistically), some amount of random exploration is built into the method. To understand how PPO works in more detail, it is instructive to look at the policy gradient objective function (used in methods such as REINFORCE):

$$L(\theta) = \hat{E}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t]$$

$\hat{A}_t$ is the advantage function, which is a key part of PPO and other policy gradient approaches. In simple terms, the advantage function describes how much better a given action is than the on-policy action for a state:

$$\hat{A}_t(s_t, a_t) = \hat{Q}_t(s_t, a_t) - \hat{V}_t^\pi(s_t)$$

In PPO, the critic provides the value estimate $\hat{V}_t^{\pi}(s_t)$. $\hat{Q}_t(s_t, a_t)$ is calculated from the total discounted actual return received during a trajectory (from state s_t onward). The critic network is trained by minimising the difference between the predictions and observed returns.

Calculating the objective function for the actor requires two parts. Firstly, the policy gradient objective function is augmented slightly:

$$L(\theta) = \hat{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta^{\text{OLD}}}(a_t|s_t)} \hat{A}_t \right]$$

This function suggests that if an action is more likely under the new policy than the old one, the ratio of the policies will be > 1. If the advantage of the action is positive, it increases $L(\theta)$, likewise if it was negative it decreases $L(\theta)$. This suggests that a policy can be improved by maximising $L(\theta)$. However, to stop large updates to the policy a clip-ratio ϵ is used to keep policy updates within certain bounds. This aids exploration and stability. The final objective function is therefore:

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

Where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta^{\text{OLD}}}(a_t|s_t)}$ is the policy ratio. This objective function is simply a clipped version of function $L(\theta)$ above. For example if $\epsilon = 0.2$ the ratio of the old to new policies can be at maximum 1.2, at minimum 0.8.

The policy is updated based on the collected trajectories through multiple epochs of stochastic gradient ascent on this objective, then trajectories are collected with the new policy and the process continues until convergence. As with DQN, these are the key ideas behind PPO, however, full pseudocode is detailed in Appendix B.

2.2 Exploration in Reinforcement Learning

As mentioned briefly in Chapter 1, many important real-world problems, for example navigating in a physical environment or learning to perform complex actions through trial and error, could be effectively addressed by reinforcement learning, as they are easy to set up as MDPs with a set of observations and actions. However, these problems are often inherently difficult to specify a dense reward function for. If a self-driving car was to be trained to navigate a track, a reward for reaching the finish line could be specified, but it would be nigh-impossible to specify a reward for every single correct motor action taken. Exploration is useful in reinforcement learning for two key reasons:

- To allow an agent to find sparse rewards in the environment.
- Even in dense reward settings, exploration is vital to reduce the probability of a policy getting stuck in local maxima. This is referred to as the exploration-exploitation trade-off.

Exploration methods have been at the heart of reinforcement learning research since the inception of the field, the next sections examine several methods common in the literature.

2.2.1 ϵ-greedy Exploration

ϵ-greedy exploration (Sutton and Barto, 2018) is the simplest way to balance exploration and exploitation in reinforcement learning. Here the agent takes the best action according to it's current knowledge with probability $1 - \epsilon$ and a random action with probability ϵ. The parameter ϵ is generally set close to one, and decays toward zero as the agent takes actions in the environment. Despite it's simplicity, it is still used extensively in modern reinforcement learning (Dabney, Ostrovski and Barreto, 2021; Gimelfarb, Sanner and Lee, 2020).

2.2.2 Action Probability Exploration

This method uses a probability distribution over action values, so higher value actions are more likely to be selected, but not guaranteed. This method is built into policy gradient approaches in deep reinforcement learning such as PPO (Schulman et al., 2017), where a softmax layer can be applied to the actor output to calculate action probabilities.

2.2.3 Boltzmann Exploration

Similar to Action Probability Exploration, this method uses the current estimates of each possible action's value and the probability of selecting any action, a, are given by the Boltzmann distribution:

$$P(a) = \frac{e^{\frac{Q(a)}{T}}}{\sum_b e^{\frac{Q(b)}{T}}}$$

Where $P(a)$ is the probability of selecting action a, $Q(a)$ and $Q(b)$ are the values of actions a and b respectively, and T is the temperature parameter, which controls the uniformity of the distribution (higher values of T increase exploration instead of exploitation). Wang et al. (2013) use Boltzmann exploration along with Q-learning in multi-agent reinforcement learning, while Haarnoja et al. (2017) propose a novel "soft Q-learning" algorithm where optimal policies are expressed as a Boltzmann distribution from which samples are drawn.

2.2.4 Entropy Maximisation

This method introduces an entropy term (a measure of a policy's randomness in selecting actions) to the objective function in deep-reinforcement learning. This extra term acts as a regulariser, as the objective is now to maximise the expected returns and the policy entropy:

$$J(\pi) = E_\pi[R] - \beta H(\pi)$$

Where $J(\pi)$ is the objective function $E_\pi[R]$ is the expected return under the policy, $H(\pi)$ is the entropy and β is the coefficient used to trade-off exploration (more weight to the entropy term) with exploitation. Entropy maximisation is useful for learning robust policies in complex environments with long time horizons (Pitis et al., 2020), as it penalises certainty in more deterministic and sub-optimal policies early in the learning process.

A key feature of these methods is that they all utilise randomness to select actions which trade off exploitation for exploration, however there is no additional signal which guides exploration in the environment. Intrinsic motivation provides this.

2.3 The Inspiration Behind Intrinsic Motivation

The entire field of reinforcement learning is arguably biologically inspired, and overlaps heavily with developmental psychology and neuroscience. For example, Schultz, Dayan and Montague (1997) study both behavioural and neuroscientific findings through quantitative reinforcement learning methods (temporal-difference). It is intuitive to understand intrinsic motivation as a method for exploration by analogy to humans and the wider animal kingdom, and this inspiration was explicitly mentioned in Barto, Singh and Chentanez (2004) when they proposed the idea of intrinsic motivation in reinforcement learning.

Humans are driven to explore their environments when young (Von Hofsten, 2007). This behaviour is evidently not driven by the direct reward received for doing so, but it aids in the pursuit of a higher goal, perhaps to build a more robust model of the environment which is useful for their survival in the long run (Meder et al., 2021). Exploration in humans is characterised by motivations such as curiosity, surprise and a general interest in novelty. While these motivations are undoubtedly more inscrutable than the intrinsic reward functions we could design in reinforcement learning, they nonetheless provide some reassurance that these methods are worth investigating as tools to aid exploration.

2.4 Intrinsic Motivation Methods

Intrinsic motivation, in a reinforcement learning context can be broadly defined as a reward signal, separate from the rewards in the environment, which guides the agent to explore in a directed way. Many different algorithms have been proposed to achieve this. This section provides an overview of some of these algorithms, broadly categorised into three methods. It is important to note that there are no rigid boundaries between the methods, and many approaches draw from more than one. In addition, as research on intrinsic motivation has developed, terms such as "novelty", "surprise" and "curiosity" have been used widely but without strict technical definition, making it harder to parse subtle differences between approaches. Nonetheless, this section attempts to provide some useful distinctions.

2.4.1 State-count Methods

A common and intuitive way to add intrinsic motivation is to encourage the exploration of novel or unseen states (or state-action pairs). Strehl and Littman (2008) utilise a simple tabular state-action pair count and introduce an intrinsic reward inversely proportional to the count: $R \propto \frac{\beta}{\sqrt{n(s,a)}}$, where $n(s, a)$ represents the number of times action a has been taken in state s. This provides the agent with higher reward for exploring novel states and actions.

A key problem with using counts is that they do not generalise to settings with large state-action spaces, especially when considering real-world or other complex problems with continuous state spaces. Count-based approaches become infeasible as almost all states will only be visited once, however, it is possible to generalise these approaches to suit continuous environments.

Tang et al. (2017) utilise hashing as a method of dimensionality reduction, allowing state count methods to be used even in highly complex environments. Bellemare et al. (2016) use a probability density model to estimate the probability of observing state s ($p(s)$). They use this to derive a pseudo-count $\tilde{n}(s)$:

$$\tilde{n}(s) = \frac{p(s)(1 - p'(s))}{p'(s) - p(s)}$$

Here $p'(s)$ differs from $p(s)$ as it represents the updated probability, after the state has been observed. This pseudo-count then represents the new information from state s, if the probability changes a large amount after the observation it indicates that the state is less familiar. The intrinsic reward derived from this is then the same as in count based approaches: $R \propto \frac{\beta}{\sqrt{\tilde{n}(s)}}$.

State-count and pseudo-count methods are intuitive and conceptually fairly simple, however they require tables or density models to be maintained and updated, which can be quite computationally expensive in high-dimensional environments. Furthermore aside from visitation counts, there are no in-built measures to assess similarity between states.

2.4.2 Prediction-based Methods

Often referred to as "surprise" or "curiosity" based methods in the literature, prediction or prediction-error based methods aim to learn a model of environmental dynamics throughout training, receiving intrinsic rewards for scenarios where their knowledge is poor and larger updates are required to their models. With these methods, clear parallels can be drawn to how animals or humans learn and explore, using experiences to build an increasingly robust model of the environment and seeking out unfamiliar situations which are likely to improve this model. For example, similarities can be drawn between these methods and the Bayesian-brain hypothesis (Hipólito and Kirchhoff, 2023).

Houthooft et al. (2016) develop a Variational Information Maximizing Exploration (VIME) method in which the agent has a probabilistic model of the environment which predicts the next state given the current state and action. The intrinsic reward is then calculated based on the Kullback–Leibler (KL) divergence between the model before and after the new state is observed, rewarding the agent for maximum information gain. This approach pushes the agent to explore areas of the environment which it is most uncertain about.

Pathak et al. (2017) take a similar approach, but using two models. A forward model, which predicts features of the next state given the current state and action, and an inverse model, which predicts the action taken based on the current and next state. The inverse model learns the features of the environment which are changed by the agents action, and this knowledge then informs the forward model's next prediction. The intrinsic reward is calculated from the forward model as:

$$r_t^i = \|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|^2$$

Where $\hat{\phi}(s_{t+1})$ is the forward model prediction of the next state, and $\phi(s_{t+1})$ is the actual observation. This Intrinsic Curiosity Module (ICM) can be integrated with other reinforcement learning algorithms, using the intrinsic reward as an additional signal for policy updates.

Achiam and Sastry (2017) propose a model which learns the transition probabilities of the MDP and uses these to define a "surprise" metric (the difference between the predicted and observed next state, as in ICM), which is used as the intrinsic reward. Similarly to ICM, this model can be integrated with standard reinforcement learning algorithms through the reward function.

2.4.3 Skill Learning Methods

Intrinsic motivation is not only useful as a method to encourage pure exploration, it can also be used to incentivise agents to learn hierarchical collections of skills. While building hierarchical skills is not the focus of this project, it is interesting to note that intrinsic motivation is useful in solving these problems, as these skills are often necessary to make progress in many environments with very sparse rewards (or no rewards at all).

Vigorito and Barto (2010) were pioneering in their use of intrinsic motivation here. Their approach uses Bayesian network structure-learning guided by intrinsic motivation to build simple skills (essentially a set of sub-policies) which accumulate into more complex skills over time.

Bougie and Ichise (2020b) use the idea of learning skills which contribute to intermediate goals. In the absence of domain knowledge, and therefore not knowing what these goals might be, the intrinsic reward motivates the agent to return to an intermediate state many times. The agent only moves on to other intermediate states once it has an understanding of the dynamics which led to that state.

2.4.4 Random Network Distillation (RND)

Burda et al. (2018) propose RND as an exploration method designed to tackle hard exploration problems, especially in Atari. The method is similar to count-based methods in that it provides an exploration bonus for visiting novel states, but it draws from prediction-based methods too, as the bonus is not based on visitation counts but the error term of a predictor network (NB: this is where the similarities end as RND is not aiming to build a model of environment dynamics). Figure 2.2 shows the relatively simple approach that RND uses to calculate intrinsic reward. A target and predictor network with identical architectures are randomly initialised at the start of training. Each takes states as input and outputs some prediction. The difference between the predictions is the intrinsic reward received by the agent in that state.

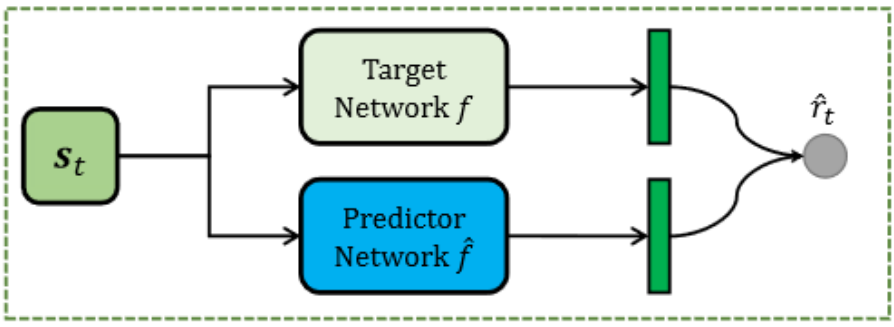

Figure 2.2: RND networks and prediction, Yuan (2022)

While the target network is fixed and not trainable, the predictor network is trained to minimise the mean-squared error (MSE) between the two networks:

$$\text{MSE} = \|\hat{f}(x;\theta) - f(x)\|^2$$

Where $\hat{f}(x;\theta)$ is the predictor network, parameterised by θ, and $f(x)$ is the target network. The effect of this is that the predictor network learns to make better predictions for states that it has seen before, consequently the intrinsic reward for visiting those states go down, which motivates the agent to explore states that it has not seen before.

As with many of the other methods detailed above, RND is only a methodology for generating intrinsic reward, not a reinforcement learning algorithm in itself. However, it integrates well with PPO as an extra module to encourage exploration. This is achieved in two ways. Firstly, recall that the PPO critic is trained by minimising the difference between it's state value predictions, $\hat{V}_t^{\pi}(s_t)$ and the observed returns, R_t. The intrinsic reward stream can be integrated here such that $R_t = R_t^I + R_t^E$, where R_t^I and R_t^E and intrinsic and extrinsic returns at step t respectively.

Secondly, the PPO advantage function below is used in training the actor (along with the policy ratio):

$$\hat{A}_t(s_t, a_t) = R_t - \hat{V}_t^{\pi}(s_t)$$

This advantage incorporates both overall return ($R_t = R_t^I + R_t^E$) directly, and value estimates from the critic, which integrates intrinsic reward into the actor as well. These are the key ways in which RND and PPO can be integrated. Full pseudocode can be found in Appendix B.

Following the initial success of RND, additional research has been conducted with it in various different sparse reward environments. Sovrano (2019) uses RND in combination with experience replay techniques to improve agent performance and sample efficiency in Atari games. Rao et al. (2022) analyse how RND can help convergence in a multi-reinforcement learning agent paradigm. Yang et al. (2019) utilise it alongside soft-actor critic methods for robotic manipulation in space, and Miceli-Barone, Birch and Sennrich (2022) use it to identify out-of-domain language in language models, helping to increase their robustness when operating in domains far removed from their training distribution.

2.5 Intrinsic Motivation Benchmark Environments

From the examples mentioned above, it is clear that intrinsic motivation is a in important research direction for exploration in reinforcement learning, applicable in a wide range of environments including game playing, continuous control, robotics and many other scenarios that require guided exploration.

There are however, several environments which feature heavily in the literature, acting as benchmarks on which to test new intrinsic motivation methods.

- **Atari:** Atari (Atari Documentation, n.d.) is a collection of 57 Atari 2600 games which are used widely across the literature due to their diversity, sparse rewards and difficulty. These were the benchmark environments used by Mnih et al. (2015) to test DQN, and

have since set the standard for testing intrinsically motivated reinforcement learning algorithms including RND (Burda et al., 2018).

- **ViZDoom:** ViZDoom (Kempka et al., 2016) is a set of environments built within the classic 3D shooter, Doom. These environments are well suited to encourage exploration in a simple 3D maze with sparse rewards. ViZDoom is explored with intrinsic motivation in Mikhaylova and Makarov (2022) and Pathak et al. (2017).
- **Super Mario:** The classic Nintendo game Super Mario (Paquette, 2024) is also used extensively and is particularly hard for reinforcement learning due to simultaneous actions, for example moving right and jumping at the same time. Bougie and Ichise (2020a) implement a progress-driven agent here, and Pathak et al. (2017) implement their intrinsic curiosity module, seeing if policies learned in one level generalise well to others.
- **MiniGrid:** MiniGrid (MiniGrid Documentation, n.d.) is a suite of customisable gridworld environments. They are ideal for visualising agent exploration, and testing state-of-the-art algorithms featuring sparse rewards and arbitrary complexity. Examples of using MiniGrid to investigate intrinsic motivation include Andres, Villar-Rodriguez and Del Ser (2022) and Zhang et al. (2021). MiniGrid is explained in more detail in Chapter 3.
- **MuJoCo:** MuJoCo (MuJoCo Documentation, n.d.) stands for Multi-Joint dynamics with Contact. It is a physics engine designed for modelling robotics and biomechanics, so is highly applicable to real-world problems of robotic control. Investigations using the DeepMind Control Suite, such as Zhang et al. (2019) also use MuJoCo as the physics engine.

This is by no means an exhaustive list, but these environments constitute a large portion of the investigations in the literature due to their ease of implementation (Atari, MiniGrid and MuJoCo are available through the Gymnasium API for example), sparse rewards, and the network effects of many researchers using them.

2.6 Problems in Intrinsic Motivation

Intrinsic motivation is designed to help alleviate the poor exploration of reinforcement learning algorithms, however, research has given rise to some unsolved or partially solved problems specific to intrinsic motivation:

- **The "Noisy-TV" problem:** This problem affects pseudo-count and prediction-based methods alike as both are searching for novel states in the environment (either due to low pseudo-count or to improve their model of environment dynamics). If an agent is exploring, for example, a maze where one section is continuously changing in a random (noisy) way, it is incentivised to remain at that point and not explore further, as each state it sees will be different and it will not be possible to improve a model of environment dynamics over time. Once of the reasons that RND was devised as a method was to mitigate the impact of this, as even noisy states are transformed deterministically by the target network.
- **Vanishing intrinsic rewards:** Again a problem affecting both pseudo-count and prediction-based methods. A feature of these methods is decreasing intrinsic rewards as the agent explores, as fewer states are novel or dynamics are better understood. This becomes an issue in large environments where intrinsic reward becomes very low but the

environment remains partially unexplored, or in sparse reward environments the agent may switch to exploiting the extrinsic reward available even though a better policy could be found through additional exploration.

- **The "Detachment" problem:** This is a special case of the vanishing intrinsic reward problem. In some environments the agent may explore one region of the environment incompletely, then start exploring a different region by chance. As the agent must pass through many states with low intrinsic reward to get back to the unexplored part of the first region, it is not incentivised to do so, leaving the unexplored part "detached" from the explored area. This problem is discussed further in Chapters 3 and 4.
- **Computational complexity:** A meta-problem in reinforcement learning, it is tempting to design ever more complex architectures requiring detailed state tracking, complex predictive models or consensus-based methods to work around some of the problems above. However this needs to be balanced against the computational requirements required to support this complexity, as even relatively simple architectures require vast computing resources as state and action spaces become larger and more complex.

2.7 The Current State-of-the-Art

Apart from RND, the algorithms described below are some of the most well known current state-of-the-art intrinsic motivation methods, in terms of their exploration and performance on benchmark environments:

- **Never-Give-Up**: Badia et al. (2020b) propose an intrinsic reward mechanism consisting of an episodic reward and a long term reward which decreases throughout training. This incentivises the agent to explore during each episode even when all regions of the environment have been explored. This is a potential solution to vanishing intrinsic rewards. They combine this with distributed agents to achieve state-of-the-art performance on *Pitfall!*, an Atari hard exploration game.
- **RIDE**: RIDE (Rewarding Impact-Driven Exploration) was introduced by Raileanu and Rocktäschel (2020). They incorporate an intrinsic reward which rewards impactful actions, that is, actions that lead to a larger change in the state of the environment. This makes it particularly useful for exploring environments that change between episodes as the action impact is the key driver of exploration, not the state itself.
- **Go-Explore**: Ecoffet et al. (2021) propose Go-Explore as a method of more directed exploration. First the algorithm explores, then learns to return to promising states (for example ones near unexplored states), then explores further from there. Go-Explore achieved state-of-the-art results on *Montezuma's Revenge* and *Pitfall!*. Scoring higher than Never-Give-Up.
- **Agent57**: Agent57 is so called by Badia et al. (2020a) as it is the first algorithm to surpass human performance on all 57 Atari 2600 games. It does so using a meta-controller to balance exploration and exploitation, selecting a balance which works effectively for the game it is playing, while leveraging the intrinsic reward mechanism from Never-Give-Up.

These algorithms give some indication of the diversity approaches available in current intrinsic motivation literature, with various algorithms excelling in different environments.

Chapter 3

Experimental Design

This chapter sets out the environmental and experimental design for testing RND, including how experiments will be empirically evaluated and details of implementation.

3.1 Overview

The intention of these experiments is to thoroughly investigate the ability of RND to explore novel environments and to solve environments where a goal is explicitly specified (an extrinsic reward function is present), but the rewards are sparse. Pathak et al. (2017) propose a set of experiments to investigate their "Intrinsic Curiosity Module" (ICM). These experiments investigate the capability of ICM in a structured and systematic way, their approach is as follows:

- **Sparse Extrinsic Reward Setting:** Investigating whether an algorithm can effectively find and optimise towards the sparse extrinsic reward in the environment.
- **No Extrinsic Reward Setting:** Investigating the exploration of an environment with no extrinsic reward present.
- **Generalisation to Novel Scenarios:** Investigating how training on a simpler environment can improve performance in more complex scenarios.

This framework provides a structure through which RND can be investigated in a novel way.

3.2 MiniGrid Environments

Tests will be conducted using MiniGrid (MiniGrid Documentation, n.d.), built on the Gymnasium library (Gymnasium Documentation, n.d.). MiniGrid is in many ways the perfect arena for testing exploration as it is highly customisable, rewards are extremely sparse and it is easy to visualise and draw inference from agent behaviour. MiniGrid is a well used environment in the reinforcement learning literature, particularly when investigating exploration and intrinsic motivation (Yuan et al., 2023), (Andres, Villar-Rodriguez and Del Ser, 2022).

MiniGrid environments are customisable gridworlds. In these environments, the agent traverses the grid interacting with objects such as doors and keys, while avoiding obstacles to reach the goal. The environments have the following attributes:

- **Observation Space:** Observations are the 7x7 grid to each side of and immediately in front of the agent, making this environment partially observable (therefore the agent-environment interaction is a POMDP). In addition, any walls or closed doors obscure the agent's view of the grid beyond. The 7x7 grid is represented by RGB pixels.
- **Action Space:** Seven actions are available to the agent. Turn left, turn right, move forward, pick up (an object), drop (an object), toggle (to open/close doors), done (to indicate task completion). All these actions except done are used as part of the test environments.
- **Rewards:** The extrinsic reward built into the MiniGrid environments is $1 - 0.9(\frac{\text{step count}}{\text{max steps}})$ when the goal is reached, and 0 otherwise. By design, there is no intermittent reward received throughout the episode.
- **Termination:** The environment terminates when either the goal is reached, or the max steps are reached. The maximum number of steps differ based on the complexity of the environment.

In all renderings of MiniGrid environments we see the agent as a red triangle, with the highlighted area representing the agent's field of vision. Walls are rendered in gray with various colours of doors. Goal squares are represented in green. Occasionally there are other goals, such as picking up a particular object. There are also objects such as keys which can be interacted with in some environments.

3.2.1 Sparse Extrinsic Reward Setting

To investigate the performance of RND in a sparse reward setting, three environments are used. Firstly, the *sequential rooms* environment is a relatively simple custom environment consisting of three small rooms separated by doors. The agent must open the doors and navigate to the goal square to complete the task. This provides a baseline environment to evaluate the ability of RND to both find the goal and to learn a policy which repeatedly returns to the goal in the minimum number of timesteps (thereby maximising reward).

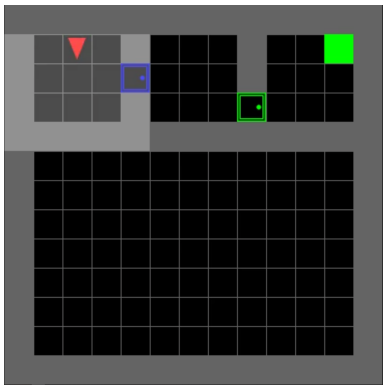

Figure 3.1: *Sequential rooms*

Secondly the *multiroom* environment ("*MiniGrid-multiroom-N6-v0, seed=1*") is a built-in environment which extends to six sequential rooms separated by doors. This extends the region that the agent needs to pass through before receiving any extrinsic reward, testing RND's ability to explore systematically.

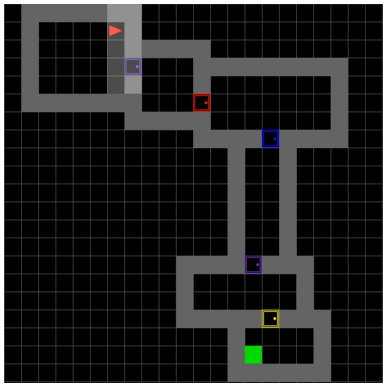

Figure 3.2: *Multiroom (N6)*

Finally the *key-corridor* environment ("*MiniGrid-KeyCorridorS5R3, seed=1*") is a more complex built-in environment where the goal is to pick up the green ball behind a locked door. This tests RND's ability to learn a sequence of actions which must be completed in a fixed order to receive the reward (navigate to the key, pick up the key, navigate to the red door, open the red door, drop the key, pick up the ball).

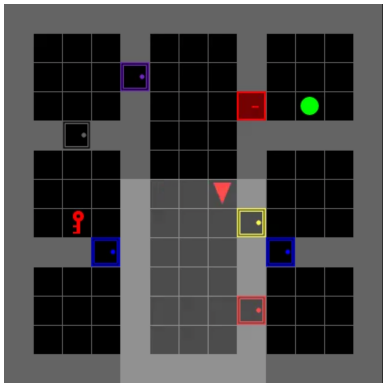

Figure 3.3: *Key-corridor (S5R3)*

3.2.2 No Extrinsic Reward Setting

With no reward in the environment, this setting is purely investigating RND exploration. It has been noted in the literature (Ecoffet et al., 2021; Zhang et al., 2021) that RND suffers from the "detachment problem". As mentioned in Chapter 2.6, this happens when there are multiple regions to explore in an environment. The agent explores some part of one region, then, by chance, starts to explore a separate region. It is then disincentivised to pass through the already explored part of the first region to get to the unexplored section (it has seen these states before so it receives low intrinsic reward). This leaves the unexplored part of the first region "detached" and unexplored. To investigate this behavior a custom MiniGrid environment is used. In this *double spiral* environment, the agent starts in between the two spirals. To explore the environment fully, the agent must first proceed around one spiral. Suppose it moves left and down to the bottom left corner, then, by chance, starts to instead explore the spiral on the right. If it is to fully explore the left spiral it must then pass through many states it has already seen to reach a new one, which it is not motivated to do by the

intrinsic reward. If the agent can reach the centre of both spirals then it does not seem to suffer from the "detachment problem".

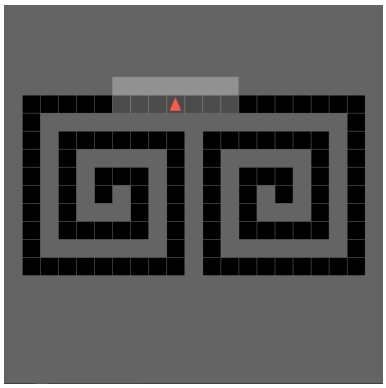

Figure 3.4: *Double spiral*

3.2.3 Generalisation to Novel Scenarios

MiniGrid is particularly useful when studying generalisation, as many environments have a range of grid sizes and levels of complexity. In these experiments, RND is pre-trained on a simple sparse-reward environment until it is solved, then training is continued on a more complex environment, with the learned weights from the simpler environment retained. This tests whether knowledge obtained from pre-training can be effectively adapted and deployed in a new environment. Two environments are particularly well suited to this.

Firstly, RND's exploration can be analysed by generalising in the *multiroom* environment. By reducing the size of the environment to only the first four rooms, and pre-training an agent to traverse these rooms, as shown in Figure 3.5, it is possible to examine whether the pre-trained agent explores the fifth and sixth rooms earlier and therefore solve the environment faster and more reliably.

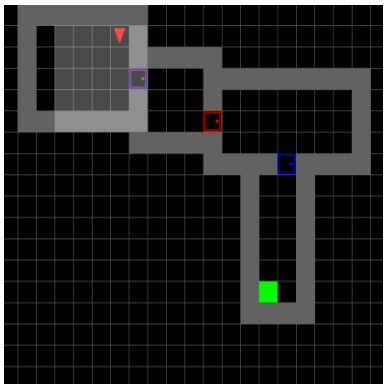

Figure 3.5: *Multiroom (N4)* pre-training

Secondly, the *key-corridor* environment detailed above has simpler configurations, "*MiniGrid-KeyCorridorS3R3, seed=1*" and "*MiniGrid-KeyCorridorS4R3, seed=1*", shown in figure 3.6. These smaller versions of the environment require the agent to learn the same set of skills. RND generalisation can therefore be examined by pre-training in these simpler settings until the agent can consistently reach the goal, then attempting to solve the larger environment. The

random seeds can also be varied to change the colours and positions of the objects, this can be useful in building robustness into the process, reducing overfitting to a single environment.

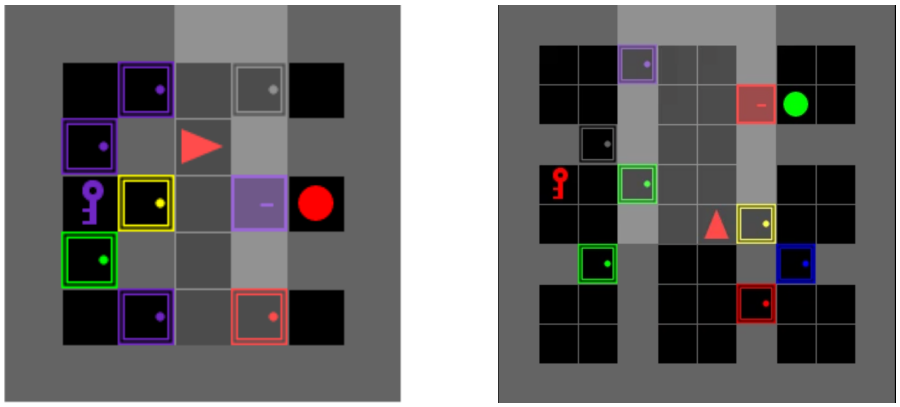

Figure 3.6: Variations of *key-corridor - S3R3* (left) and *S4R3* (right)

3.3 Measurement

The metrics used closely follow those used by Pathak et al. (2017) to evaluate the ICM, and the methods in the wider intrinsic motivation literature.

- **Sparse Extrinsic Reward Setting:** Here we will be looking at average extrinsic rewards per episode. With this metric we can
  - (i) identify whether or not a environment has been solved,
  - (ii) compare each of the learning agents to one another, given the same set of experimental conditions, and
  - (iii) intuit the algorithm's efficiency over time. This is due to MiniGrid's reward function being inversely proportional to the number of steps taken in each episode $(1 - 0.9(\frac{\text{step count}}{\text{max steps}}))$.
- **No Extrinsic Reward Setting:** Here we will be using "first-time visits" to each state during training. For each state in the environment, this evaluates if and when that state was first visited during training. This measures both the extent and efficiency of exploration, while also allowing it to be visualised in an intuitive way. This metric is also used in the sparse reward case to demonstrate exploration.
- **Generalisation to Novel Scenarios:** Here we are looking to measure two distinct goals. Firstly, is it possible to solve sparse reward environments which cannot normally be solved by pre-training on some simpler environment. Secondly does pre-training on a simpler environment allow more efficient exploration of an environment. Both these goals can be evaluated using the same methodologies as for the sparse and no extrinsic reward settings.

Three non-intrinsic motivation algorithms are measured alongside RND, acting as baselines for performance. Firstly, an agent that takes only random actions, secondly an agent using Deep Q-Networks (DQN) (Mnih et al., 2015) and finally a Proximal Policy Optimisation (PPO) agent (Schulman et al., 2017). For the generalisation experiments, only RND is studied as these

test it's ability to generalise to new scenarios using pre-training, rather than it's performance against other algorithms.

3.4 Implementation Details

3.4.1 Training and Evaluation Pipeline

In order to evaluate the performance and exploration of the algorithms throughout training, for each environment and each algorithm, the following pipeline is constructed to obtain the metrics detailed above:

- Training is initialised for a set number of environment steps (400,000 or 800,000 for most environments but variable based on it's complexity).
- First-time state visits are recorded during training so the agent's exploration process can be documented.
- Periodically, training is paused and the learned policy is run on a separate (but identical) evaluation environment for 50 episodes, returning the average extrinsic reward received. This allows for a reliable measurement of performance at different stages in training, without disrupting the actual training process or the recording of first-time state visits. (NB. This step is not run in the no reward case).
- Training is then resumed, with first-time state visits continuing to be recorded.

To account for random initialisation of parameters and consequent variance in performance throughout training in each environment, the above process is repeated ten times on the environment for each algorithm, with averages calculated for extrinsic reward and first-time state visits.

3.4.2 Algorithms

As the observations in all environments are RGB images, a convolutional neural network (CNN) is utilised to reduce the image dimensionality while preserving spatial relationships between the pixels. This network is fixed, and only the flattened observation is passed as input to each agent. The architecture for the CNN can be found in Appendix A.

The DQN implementation is based on the pseudocode found in Mnih et al. (2015), with the PPO implementation based on OpenAI's "Spinning Up" (OpenAI Spinning Up Documentation, n.d.). The RND networks are used with the PPO algorithm, as described in Chapter 2, which allows for the effects of RND to be compared to standard PPO without introducing further variables. This also aligns with the approach taken in the original RND paper (Burda et al., 2018) and a large majority of the literature.

When integrating RND with PPO, the extrinsic reward is weighted more heavily (80%). This gives far more weight to exploitation compared to exploration. However, given rewards are very sparse (zero in almost all timesteps), the effect of this is to provide a stronger signal when the agent does reach the goal, increasing the likelihood that the agent can consistently return to it and therefore solve the environment.

The hyperparameters of each algorithm are tuned to perform well on these environments, and are detailed in Appendix A, along with their architectures.

Chapter 4

Results Analysis

This chapter analyses the results of the experiments both quantitatively and qualitatively, following the structure defined in Chapter 3.2.

4.1 Sparse Reward

4.1.1 Sequential Rooms

In this environment (Figure 3.1), the goal is to traverse the rooms and reach the goal square in the minimum number of timesteps. Figure 4.1 shows the performance of random, DQN, PPO and RND agents on the environment.

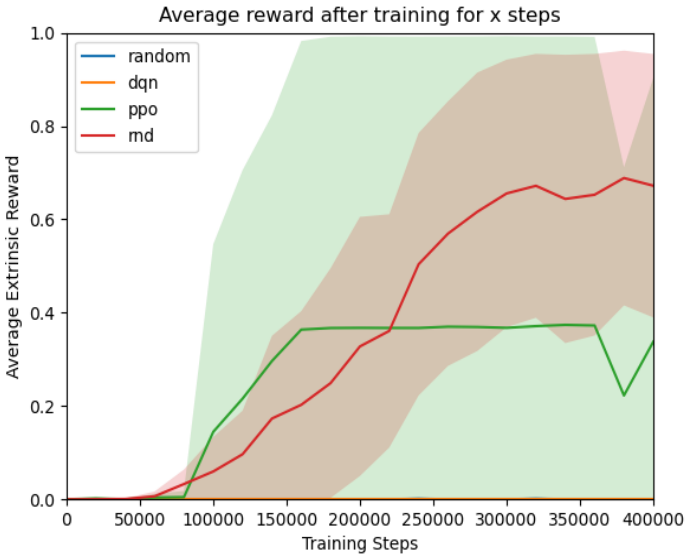

Figure 4.1: Extrinsic rewards - *sequential rooms*

After 400,000 training steps, we can clearly see that RND performs most effectively on the environment, with an average extrinsic reward of 0.62. The shaded areas in red and green display the standard deviation of the algorithm runs, with the large variance explained by the nature of the reward (either the reward signal is found and reward increases to near 1

each episode, or the goal is not found and reward remains at 0). PPO also reaches the goal somewhat regularly, despite the lack of intrinsic reward encouraging exploration of the environment. Another interesting difference between the behaviour of RND and PPO is that PPO appears to learn faster, with the gradient of it's reward curve steeper between 75,000 and 150,000 timesteps. This is indicative of the reward signal during the early stage of training where PPO is receiving only extrinsic reward from the goal, whereas the RND agent is still incentivised to explore the environment through intrinsic reward.

It appears that DQN and the random agent don't reach the goal and therefore do not receive any reward. This can be investigated further by analysing the states visited by each algorithm during training.

Figure 4.2 Maps the first-time state visitation of each algorithm in the *sequential rooms* environment, allowing for a visual representation of exploration (top-left: Random Agent, top-right: DQN, bottom-left: PPO, bottom-right: RND). The red and green squares on the heatmaps show the agent start and goal location respectively.

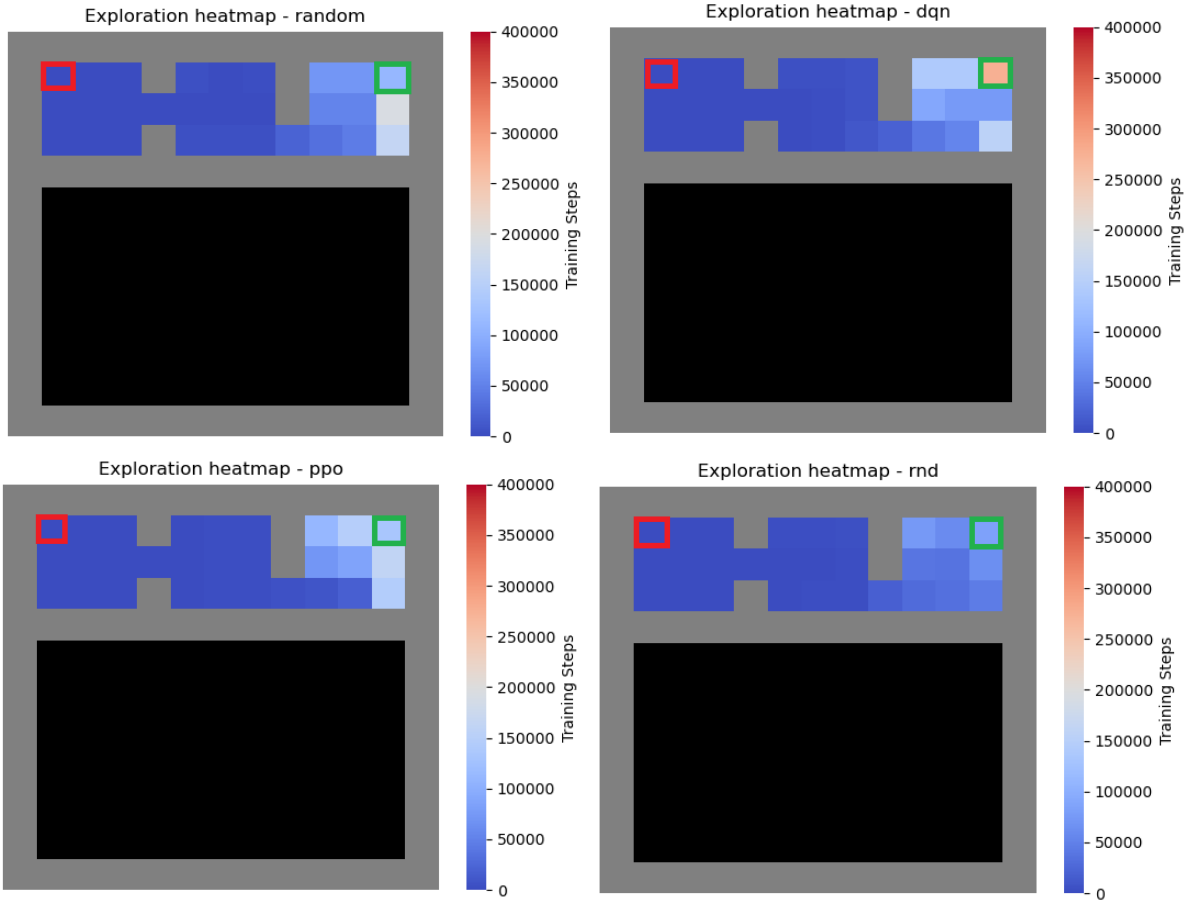

Figure 4.2: Exploration of the *sequential rooms* environment

The heatmaps suggest that the random and DQN agents do find the goal in, on average, 150,000 and 300,000 timesteps respectively. However, the reward curve suggests that the receive zero reward during evaluation. This implies that although the random and DQN agents still explore the environment fully, when running episode evaluations (as described in Chapter 3.4.1) their learned policies do not provide incentive for them to return to the goal

systematically. Only the PPO and RND agents receive a strong enough reward signal to learn the actions of opening and going through the sequential doors to reach the goal repeatedly, thereby solving the environment.

Importantly, RND behaves as expected in that it explores the environment in slightly fewer timesteps than the algorithms which do not utilise intrinsic reward, although this behaviour requires more complex environments to become clear.

4.1.2 Multiroom

The MiniGrid *multiroom* environment (Figure 3.2) is a natural extension of the *sequential rooms* environment, here featuring six rooms separated by doors, with a goal in the final room. Due to the increased complexity of the environment, training was run for 800,000 timesteps as opposed to the 400,000 for *sequential rooms*.

From Figure 4.3, it is clear that none of the algorithms are able to solve this more complex environment, though RND does reach the goal, achieving a small reward after 375,000 training steps.

Figure 4.3: Extrinsic rewards - *multiroom (N6)*

Again, exploration is visualised in Figure 4.4. The random and DQN agents reach only the third room, PPO explores the third room fully and only RND reaches the sixth room and the goal square. Figure 4.4 suggests that RND reaches the sixth room and goal square in between 600,000 and 700,000 timesteps, however this is due to averaging over multiple runs of the environment (if a state isn't reached on every training pipeline run, then for the purposes of the first-time visit averages it is mapped as being reached at the very end of training for those runs where it wasn't achieved). This implies therefore that although RND does explore enough to reach the goal, it doesn't necessarily reach it during every training pipeline run.

To determine whether RND's performance is limited by only running training for 800,000 timesteps, training was continued up to 5,000,000 timesteps. Even with this increase, RND was not able to consistently reach the goal square and solve the environment. The likely

reasons for this are twofold. Firstly, as RND explores the environment, the intrinsic reward due to prediction error decreases. Reward is received when a new room is found, but this reward decreases as more steps are taken in that room. This means that there is only a relatively short window for the extrinsic reward (from the goal) to provide incentive for the agent to return to the sixth room. This window starts the first time the agent reaches the goal, and ends when the intrinsic reward received in intermediate rooms is no longer sufficient to incentivise progression through the rooms toward the goal. Secondly, the agent is required to take many sequential actions to reach the goal (traversing 6 rooms and opening 5 doors). This gives rise to the credit assignment problem where the agent is not certain which actions were necessary to obtain the reward.

These findings are somewhat consistent with Zhang et al. (2021). They find that RND cannot solve similar *multiroom* environments, though they note that RND only reaches the fifth room after 10M environment steps, whereas the sixth room is reached in these tests.

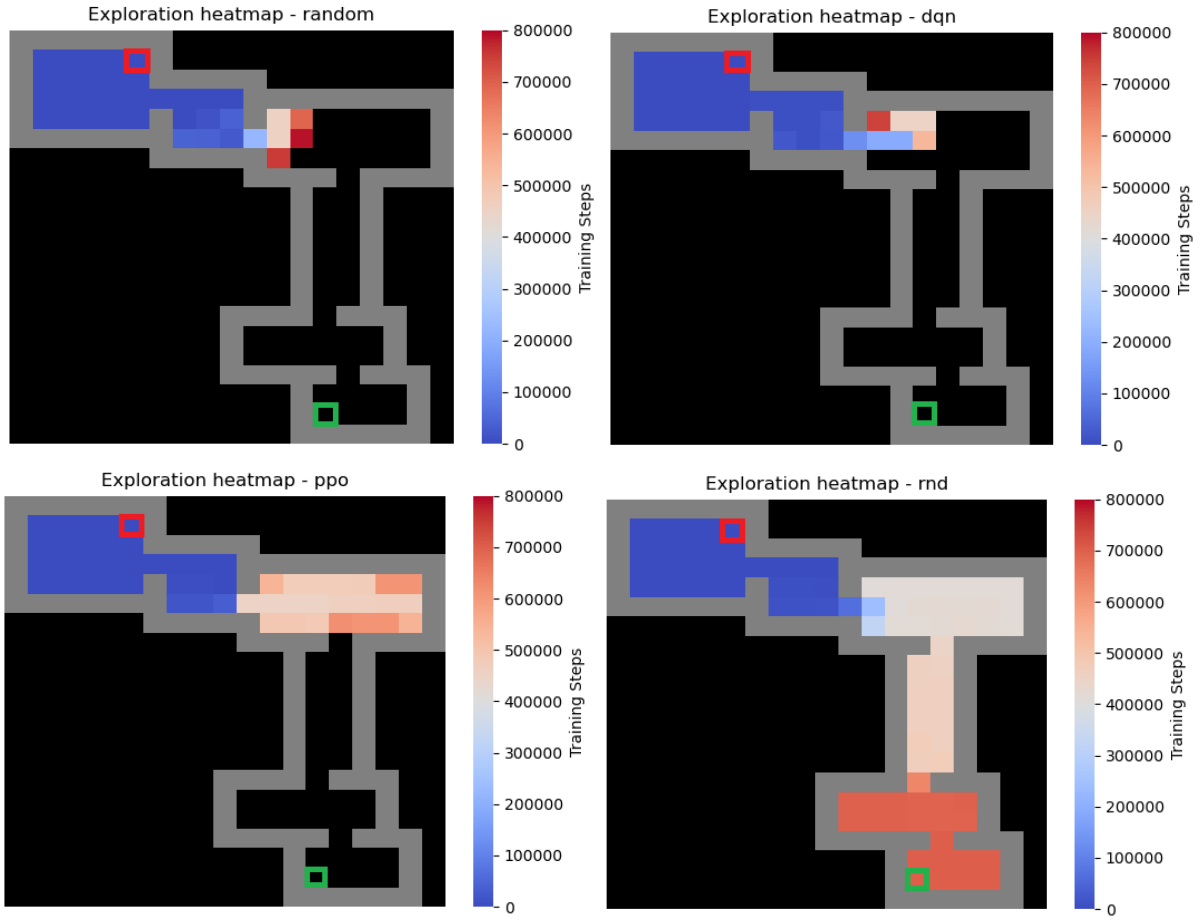

Figure 4.4: Exploration of the *multiroom (N6)* environment

4.1.3 Key-Corridor

The *key-corridor* environment (Figure 3.3) is very difficult to solve using reinforcement learning as it features both sparse rewards and a hierarchical set of skills that must be learned and executed in order. Figure 4.5 shows exploration on this environment. From the starting red square, the agent must move to the top of the grid, opening the door and navigating through the top-left room into the centre-left room, where there is a key. The agent must then pick

up the key and navigate back to the top-right room. It must open the door while holding the key, drop the key and pick up the ball located on the green square. The other rooms in the environment do not contain anything which would help the agent. The heatmaps show a similar trend to other environments, that RND explores the environment more efficiently than non-intrinsically motivated algorithms, reaching the furthest bottom left room within approximately 300,000 timesteps. Other algorithms take upwards of 500,000 timesteps to reach the same point. None of the agents are able to take the key and use it to open the door to enter to top right room, and therefore none reach the goal even once during training. A clear conclusion, from the RND agent's performance here, is that encouraging exploration through intrinsic motivation alone is not sufficient to build the skills required to solve more complex environments.

An important inference that can be made here is that, similarly to the *multiroom* environment, there is only a short window in which RND can pick up the key, open the locked door, drop the key and pick up the ball. Each time the agent takes one of these actions, the intrinsic reward it receives from doing so reduces, as it is better able to predict these states. Over time, the reward reduces to the point where it is similar to the reward received from taking actions which do not advance toward the goal. Therefore the probability of taking the useful actions drops substantially. This evidence from both the *multiroom* and *key-corridor* environments suggests that RND suffers from the vanishing intrinsic reward problem discussed in Chapter 2.6.

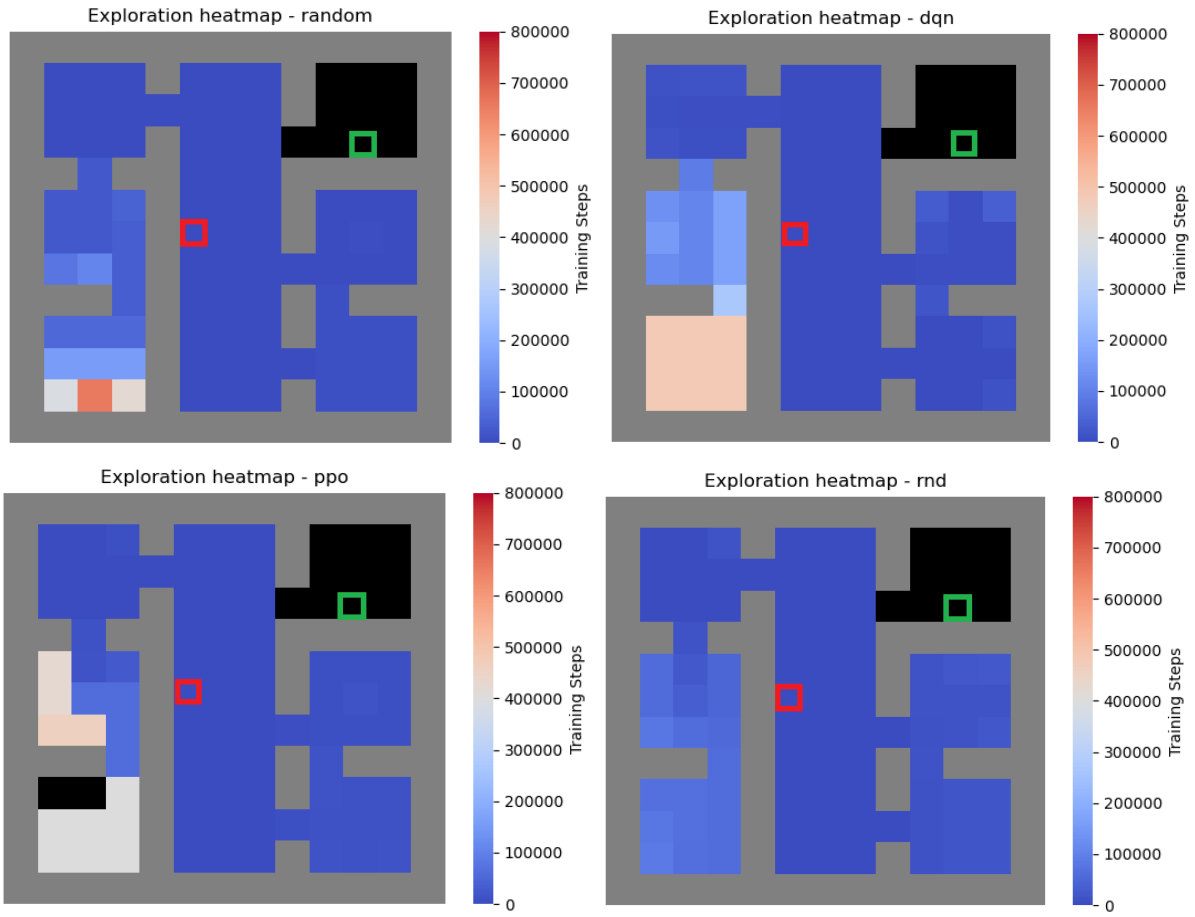

Figure 4.5: Exploration of the *key-corridor (S5R3)* environment

4.2 No Extrinsic Reward

4.2.1 Double Spiral

As noted in Chapter 3, the *double spiral* environment (Figure 3.4) is designed to test RND's susceptibility to the "detachment problem". To investigate this, the testing paradigm is altered slightly, with each heatmap now representing a single training run in the environment, as opposed to an average over 10 runs. This is to identify whether RND is only exploring a single spiral in a single training run, and to visualise how RND exploration differs to random, DQN and PPO in general.

Exploration of the random (top row), DQN (middle row) and PPO (bottom row) agents for three training runs of 800,000 timesteps each is shown in Figure 4.6. The key observation here is that, although PPO explores a higher proportion of the environment than DQN or the random agent, all the algorithms balance their exploration of each spiral approximately evenly during each training run.

In contrast, Figure 4.7 shows RND's exploration over six training runs. In each training run, RND systematically explores one side of the environment, reaching the centre of the spiral in 300,000-400,000 timesteps, however it only partially explores the other side over the entire training run. This supports the "detachment problem" findings of (Ecoffet et al., 2021) and (Zhang et al., 2021).

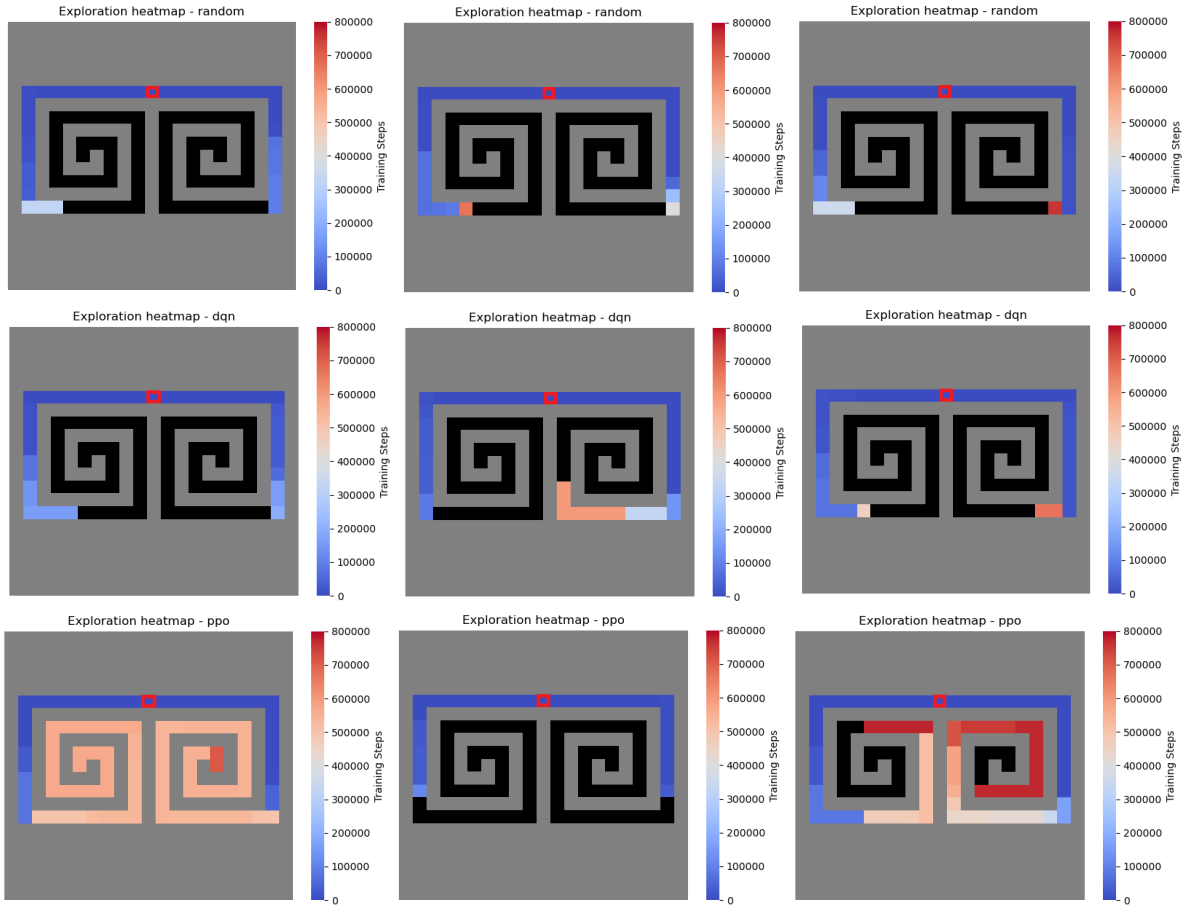

Figure 4.6: Exploration of the *double spiral* environment - Random, DQN and PPO agents

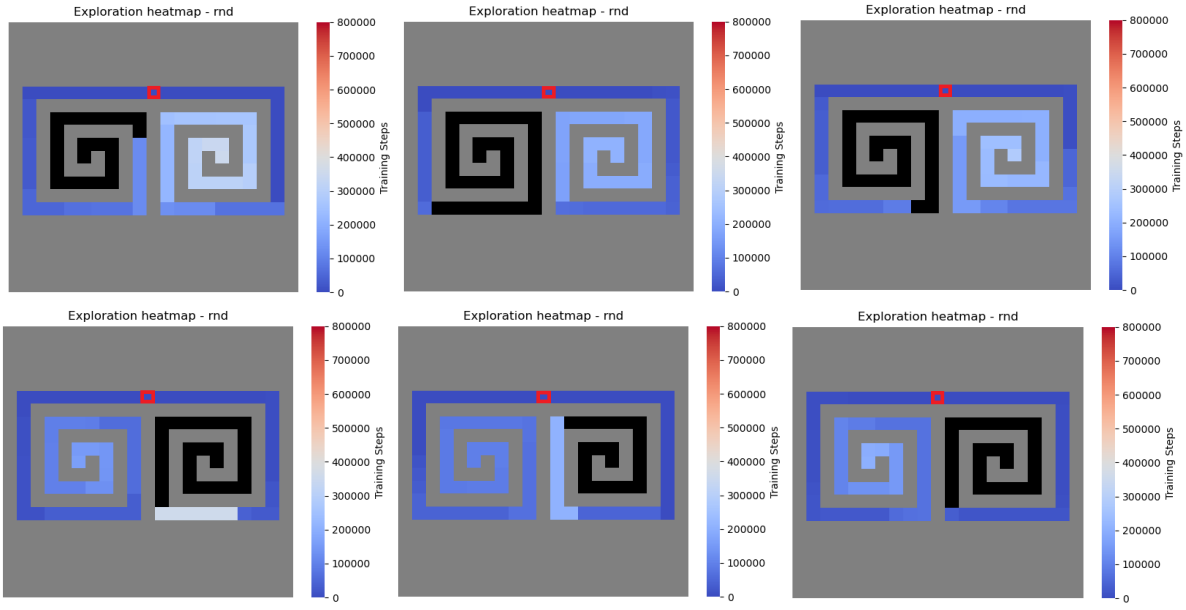

Figure 4.7: Exploration of the *double spiral* environment - RND agent

4.3 Generalisation to Novel Scenarios

4.3.1 Multiroom

As detailed in Chapter 3, this experiment examines whether RND generalises well from a four room (Figure 3.5) to a six room (Figure 3.2) *multiroom* environment. Extrinsic reward and exploration in the pre-training four room environment is displayed in Figure 4.8.

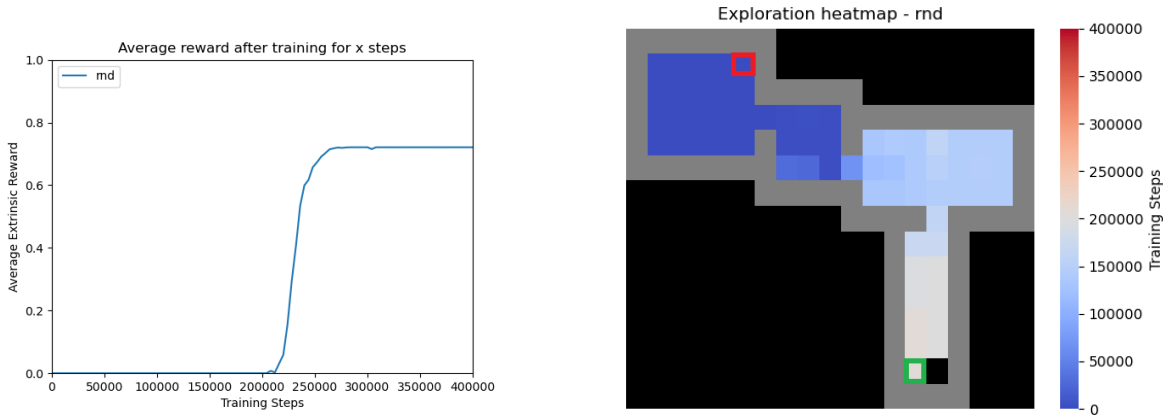

Figure 4.8: RND extrinsic rewards (left) and exploration (right) - *multiroom N4*

In this environment, RND can find the reward and consistently return to the goal, solving it within 250,000 timesteps. Importantly, the goal square in the four room environment is placed directly in front of a door in the six room environment, to aid faster exploration and solving.

The network architecture and hyperparameters chosen for this task were different to aid with generalisation. An additional deep layer was added to all networks and the first layer of the actor network made untrainable, to help preserve the knowledge gained in the four room

pre-training. Additionally the intrinsic weight was reduced to incentivise returning to the reward. Initially this test was implemented with the same configuration as for the sparse reward and no reward scenarios, however this resulted in "catastrophic forgetting", as detailed in Kirkpatrick et al. (2017), with the agent quickly reverting to exploration of the environment guided by intrinsic reward. The exact architecture and hyperparameter choices are detailed in Appendix A.

As slightly different configurations were used here, an RND agent with identical parameters but no pre-training was also tested on the six room environment to allow for a fair comparison. The extrinsic rewards recieved by the agents are detailed in Figure 4.9 and their exploration in Figure 4.10

Figure 4.9: RND extrinsic rewards - *multiroom N6*

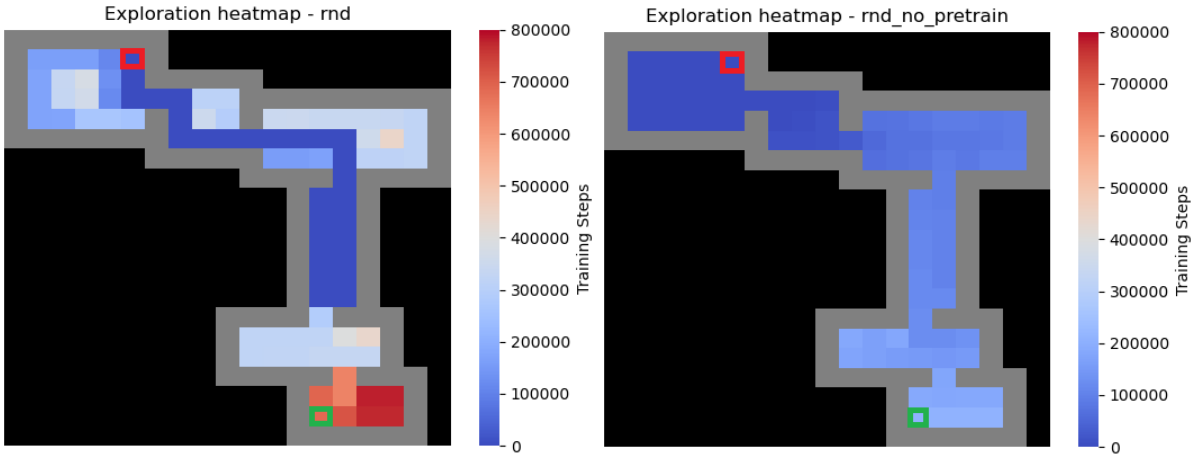

Figure 4.10: RND exploration - *multiroom N6* with (left) and without (right) pre-training

This presents several interesting findings. Comparing RND's exploration without pre-training to that in Chapter 4.1.2, it appears that an additional deep layer allows for more reliable exploration of the environment, with the sixth room reached in under 400,000 timesteps

(averaged across 10 runs). However, even with more weight placed on the extrinsic reward, RND is unable to consistently find the reward and solve the environment.

The pre-trained RND agent explores the environment to the end of the fourth room immediately, as it is simply following the policy learned throughout training on the four room environment. This knowledge is clearly well preserved within the network weights, as other grid squares aside from the path to the four room goal are not explored until much later in training. The sixth room is explored from approximately 600,000 timesteps onward, much later than without pre-training suggesting RND may explore more efficiently without prior knowledge of the environment.

The agent that has been pre-trained appears to more reliably find the reward signal from 700,000 timesteps onward. This suggests that knowledge obtained from pre-training may guide the agent more reliably to the end of the fourth room, from which there is a higher probability of it going on to find the goal in the sixth room, compared to if it is exploring from the start.

This experiment suggests there is some trade-off between performance and exploration, with performance improved by utilising prior knowledge, but efficient exploration achieved by starting from scratch.

4.3.2 Key-Corridor

In this environment, the ability of RND to learn a set of skills in a simple environment and use them in a more complex one is being assessed. To that end, RND is pre-trained on the *key-corridor S3R3* environment (Figure 3.6), with the results shown in Figure 4.11.

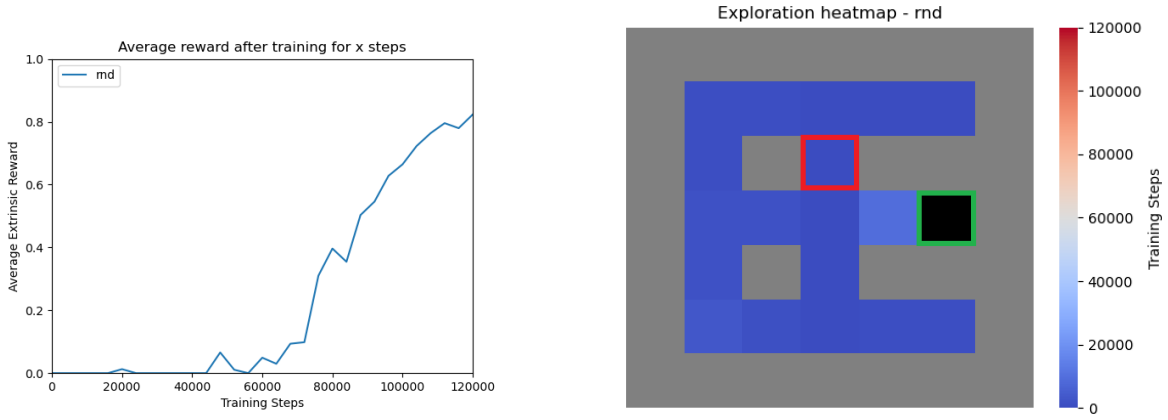

Figure 4.11: RND extrinsic rewards (left) and exploration (right) - *key-corridor S3R3*

Due to the small size of this environment, it is far easier for the agent to navigate to the key, open the doors and reliably pick up the ball to complete the task (note the goal square here is black as the agent doesn't need to move there to solve the environment, only pick up the ball placed there). Having learned how to solve this environment, the pre-trained agent is then applied on the larger *S4R3* (Figure 3.6) and *S5R3* (Figure 3.3) environments. The results for *S4R3* are shown in Figure 4.12 and for *S5R3* in Figure 4.13. *S5R3* is the largest of the *key-corridor* environments and is tested without pre-training in the prior sparse-reward analysis.

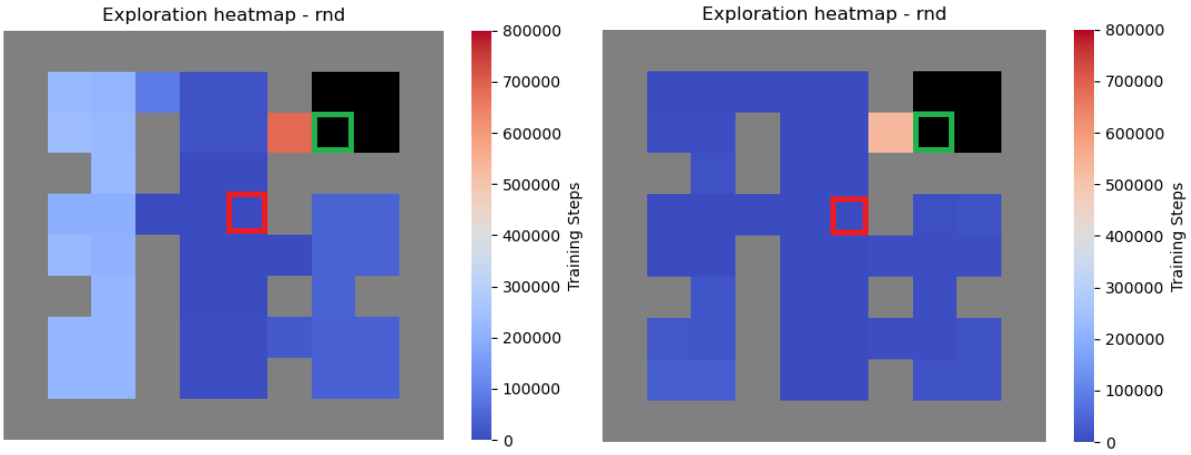

Figure 4.12: RND exploration - *key-corridor S4R3* with (left) and without (right) pre-training

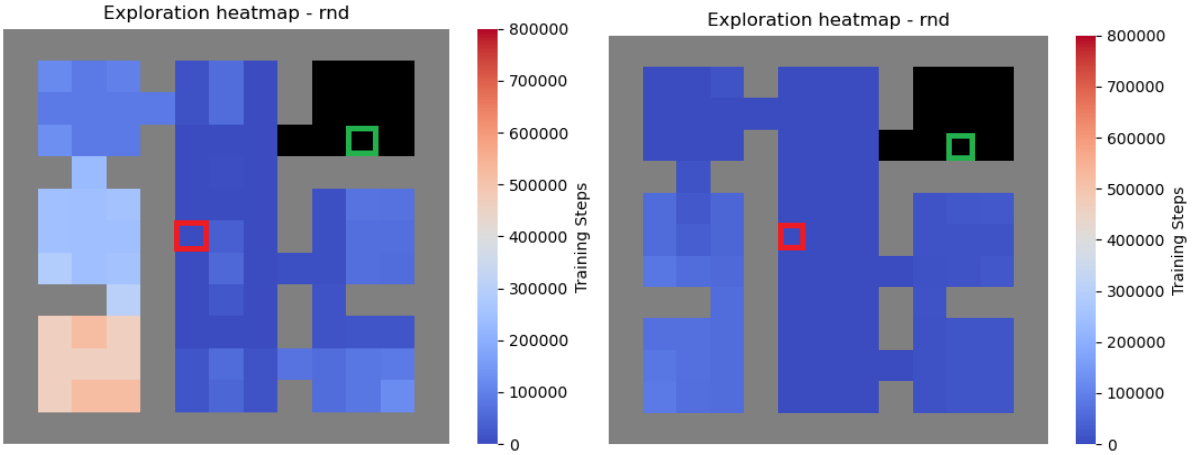

Figure 4.13: RND exploration - *key-corridor S5R3* with (left) and without (right) pre-training

In the *S4R3* environment, the agent is able to navigate to the key, open the locked door and move next to the ball. However it does not drop the key and pick up the ball to complete the task. This is the case both with and without pre-training. In the *S5R3* environment, the pre-trained agent explores more slowly than without pre-training, echoing the results seen in the *multiroom* experiments. Neither agent is able to open the locked door and complete the task.

It is clear that pre-training has not improved performance on this environment. These results were observed with just two deep layers, but no meaningful difference was recorded using the three deep layer architecture and actor network freezing from Chapter 4.3.1. Additionally pre-training the agent on multiple *S3R3* environments to develop a more robust policy did not improve performance on *S4R3* or *S5R3*.

It may be the case that learning a sequence of actions requires a more structured approach such as curriculum learning. The agent can learn the skills of picking up the key, opening the locked door, dropping the key and picking up the ball independently, before combining them to solve the more complex *key-corridor* environments. Such experiments are a possible avenue of future research.

Chapter 5

Conclusions and Further Work

This final chapter summarises and comments upon the results of the project in light of the original objectives, highlighting limitations and suggesting potential improvements and further investigations to be conducted.

5.1 Achievements

Objective 1: Situate RND within the wider intrinsic motivation literature

Chapter 2 describes the key methods within intrinsic motivation - count-based, prediction-based and skill based methods and presents RND as drawing from both count and prediction-based methods. In addition the theoretical groundwork for RND is set out in descriptions of MDPs, reinforcement learning and PPO. Additional descriptions of benchmark environments, issues commonly encountered in the literature, and the current state-of-the-art provide context for this investigation of RND.

Objective 2: Evaluate the exploration and performance of RND in sparse reward environments, in comparison to baseline algorithms

Three sparse reward MiniGrid environments were investigated - *sequential rooms*, *multiroom* and *key-corridor*. Experiments in the *sequential rooms* environment provided evidence that PPO and RND outperform DQN and random agents in a simple sparse reward environment, and that RND explores more efficiently than PPO.

The experiments conducted in the *multiroom* environment corroborated the evidence from *sequential rooms* that RND explores in a more efficient way than the baseline algorithms. However the size of the environment leads to vanishing intrinsic rewards which hampers RNDs ability to consistently find the goal and solve the environment.

The *key-corridor (S5R3)* environment tested RNDs ability to not only explore but also execute a sequential set of actions to achieve the reward. The experiment showed that this task is incredibly difficult to solve via exploration alone, with RND and baseline methods unable to reach the goal.

Taken together, these experiments show that RND explores more efficiently and outperforms methods which do not utilise intrinsic motivation in sparse reward environments. However it's

exploration is limited by vanishing intrinsic reward and a lack of understanding of environment dynamics.

Objective 3: Evaluate the exploration of RND in no reward environments, in comparison to baseline algorithms

The *double spiral* Minigrid environment was used to investigate whether RND suffers from the "detatchment problem" as described in Chapter 2. This experiment revealed conclusively that it does appear to suffer from this problem, in each training run only exploring one spiral fully, whereas baseline methods balanced their exploration across both spirals (albeit incompletely on average).

Objective 4: Evaluate the ability of RND to generalise, using knowledge from a simple setting to improve performance on a more complex one

As RND was not able to solve them initially, both the *multiroom* and *key-corridor* environments were re-evaluated to see if either would benefit from pre-training on simpler versions of the environments. The generalisation of RND from the four room to six room *multiroom* environment demonstrated promising results, with the pre-trained agent exploring more slowly than an untrained agent but learning a policy which reached the six-room goal reliably towards the end of training.

However, even though pre-training on the smaller *key-corridor S3R3* environment was successful, with RND able to solve the environment, this knowledge did not transfer to the larger *S4R3* or *S5R3* environments, with the pre-trained agent performing no better than an agent trained from scratch.

5.2 Limitations and Improvements

There are several limitations and improvements which could be made to this project. Firstly, investigating exploration in reinforcement learning is inherently computationally expensive, with hundreds-of-thousands of iterations and multiple training runs required to understand whether an algorithm performs adequately on a given environment. These experiments were performed on a single GPU which in hindsight placed a constraint on the number and complexity of experiments.

Secondly, the network architecture for the CNN (used to transform the observations from MiniGrid) as well as for the DQN, PPO and RND algorithms was based on those found in the literature from Burda et al. (2018) and the OpenAI Spinning Up Documentation (n.d.). These architectures are proven to work and so were an obvious choice for implementation, however there is little justification in the literature for exactly why these are the optimal choices. This project could have placed more emphasis on finding the optimal architectures.

Following on from the point above, DQN, PPO and RND are highly sensitive to their hyperparameters. While useful results were achieved from each experiment, using a hyperparameter optimisation library such as BayesOpt (Martinez-Cantin, 2014) could have been used to ensure these were tuned optimally.

5.3 Further Investigations

The results of this project naturally lead to three further investigations. Firstly, the no extrinsic reward setting was used to investigate whether RND as affected by the detachment problem. However additional no reward environments could be investigated within MiniGrid or outside, for example Mario or ViZDoom, to further understand RND exploration in comparison to other baseline algorithms or intrinsic reward methodologies.

Secondly, while the issue of RND suffering from the detachment problem was confirmed, how it could be addressed is left for a separate investigation. A possible candidate for this could be to use a two part reward function inspired by Never-Give-Up but using RNDs predictor and target network architectures. This could help to maintain higher intrinsic rewards even for a detached region of the environment.

Finally, generalisation in RND seems to be a difficult problem, especially when the agent is required to learn sequential actions in order to reach the goal (as in *key-corridor*). A further investigation could evaluate whether curriculum learning (Bengio et al., 2009) could help solve this task. For example, the agent could be trained on a simple set of environments, each dedicated to one required action (one for navigating rooms, one to pick up a key, one to open locked doors). In this way it may be possible to incrementally increase the agent's knowledge to the point where it can solve the larger *key-corridor* environments.

5.4 Conclusion

RND is an important part of the ongoing research into intrinsic motivation, being lightweight, integrating well with established reinforcement learning algorithms such as PPO and solving the "Noisy-TV" problem experienced by other methods. Despite this, there has been relatively little directed research into the exploration behaviour of RND. This project fills that gap by contributing a systematic analysis of RND in environments characterized by sparse or absent rewards. It also evaluates RND's ability to generalise well in sparse reward environments. This project acts as a foundational step for further research into RND's potential applications in reinforcement learning.

Bibliography

Achiam, J. and Sastry, S., 2017. enSurprise-Based Intrinsic Motivation for Deep Reinforcement Learning [Online]. ArXiv:1703.01732 [cs]. Available from: http://arxiv.org/abs/1703.01732 [Accessed 2024-01-02].

Andres, A., Villar-Rodriguez, E. and Del Ser, J., 2022. An Evaluation Study of Intrinsic Motivation Techniques applied to Reinforcement Learning over Hard Exploration Environments [Online]. vol. 13480, pp.201–220. ArXiv:2205.11184 [cs]. Available from: https://doi.org/10.1007/978-3-031-14463-9_13 [Accessed 2023-12-11].

enAtari Documentation, n.d. [Online]. Available from: https://gymnasium.farama.org/environments/atari.html [Accessed 2024-01-04].

Badia, A.P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z.D. and Blundell, C., 2020a. enAgent57: Outperforming the Atari Human Benchmark [Online]. *Proceedings of the 37th International Conference on Machine Learning*. PMLR, pp.507–517. ISSN: 2640-3498. Available from: https://proceedings.mlr.press/v119/badia20a.html [Accessed 2024-01-04].

Badia, A.P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A. and Blundell, C., 2020b. Never Give Up: Learning Directed Exploration Strategies [Online]. ArXiv:2002.06038 [cs, stat]. Available from: https://doi.org/10.48550/arXiv.2002.06038 [Accessed 2024-01-04].

Barto, A.G., Singh, S. and Chentanez, N., 2004. enIntrinsically Motivated Learning of Hierarchical Collections of Skills.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D. and Munos, R., 2016. enUnifying Count-Based Exploration and Intrinsic Motivation.

Bellman, R., 1958. Dynamic programming and stochastic control processes. *Information and control* [Online], 1(3), pp.228–239. Available from: https://doi.org/10.1016/S0019-9958(58)80003-0 [Accessed 2024-01-03].

Bengio, Y., Louradour, J., Collobert, R. and Weston, J., 2009. Curriculum learning [Online]. *Proceedings of the 26th Annual International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, ICML '09, pp.41–48. Available from: https://doi.org/10.1145/1553374.1553380 [Accessed 2024-01-05].

Bougie, N. and Ichise, R., 2020a. enExploration via Progress-Driven Intrinsic Rewards [Online]. In: I. Farkaš, P. Masulli and S. Wermter, eds. *Artificial Neural Networks and Machine Learning – ICANN 2020*. Cham: Springer International Publishing, Lecture Notes in Computer Science, pp.269–281. Available from: https://doi.org/10.1007/978-3-030-61616-8_22.

Bougie, N. and Ichise, R., 2020b. enSkill-based curiosity for intrinsically motivated reinforcement learning. *Mach learn* [Online], 109(3), pp.493–512. Available from: https://doi.org/10.1007/s10994-019-05845-8 [Accessed 2024-01-02].

Burda, Y., Edwards, H., Storkey, A. and Klimov, O., 2018. Exploration by Random Network Distillation [Online]. ArXiv:1810.12894 [cs, stat]. Available from: http://arxiv.org/abs/1810.12894 [Accessed 2023-12-11].

Dabney, W., Ostrovski, G. and Barreto, A., 2021. enTEMPORALLY-EXTENDED -GREEDY EXPLORATION.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K.O. and Clune, J., 2021. Go-Explore: a New Approach for Hard-Exploration Problems [Online]. ArXiv:1901.10995 [cs, stat]. Available from: http://arxiv.org/abs/1901.10995 [Accessed 2023-12-11].

Gimelfarb, M., Sanner, S. and Lee, C.G., 2020. {\epsilon}-BMC: A Bayesian Ensemble Approach to Epsilon-Greedy Exploration in Model-Free Reinforcement Learning [Online]. ArXiv:2007.00869 [cs, stat]. Available from: https://doi.org/10.48550/arXiv.2007.00869 [Accessed 2023-12-20].

Gymnasium Documentation, n.d. [Online]. Available from: https://gymnasium.farama.org/index.html [Accessed 2023-12-11].

Haarnoja, T., Tang, H., Abbeel, P. and Levine, S., 2017. enReinforcement Learning with Deep Energy-Based Policies [Online]. *Proceedings of the 34th International Conference on Machine Learning*. PMLR, pp.1352–1361. ISSN: 2640-3498. Available from: https://proceedings.mlr.press/v70/haarnoja17a.html [Accessed 2023-12-20].

Hipólito, I. and Kirchhoff, M., 2023. engBreaking boundaries: The Bayesian Brain Hypothesis for perception and prediction. *Conscious cogn* [Online], 111, p.103510. Available from: https://doi.org/10.1016/j.concog.2023.103510.

Houthooft, R., Chen, X., Chen, X., Duan, Y., Schulman, J., Turck, F.D. and Abbeel, P., 2016. enVIME: Variational Information Maximizing Exploration.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J. and Jaśkowski, W., 2016. ViZDoom: A Doom-based AI research platform for visual reinforcement learning [Online]. *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. pp.1–8. ISSN: 2325-4289. Available from: https://doi.org/10.1109/CIG.2016.7860433 [Accessed 2024-01-04].

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D. and Hadsell, R., 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* [Online], 114(13), pp.3521–3526. Publisher: Proceedings of the National Academy of Sciences. Available from: https://doi.org/10.1073/pnas.1611835114 [Accessed 2023-12-30].

Martinez-Cantin, R., 2014. BayesOpt: A Bayesian Optimization Library for Nonlinear Optimization, Experimental Design and Bandits. *Journal of machine learning research* [Online], 15(115), pp.3915–3919. Available from: http://jmlr.org/papers/v15/martinezcantin14a.html [Accessed 2024-01-05].

Meder, B., Wu, C.M., Schulz, E. and Ruggeri, A., 2021. enDevelopment of directed and random exploration in children. *Developmental science* [Online], 24(4), p.e13095. _eprint:

https://onlinelibrary.wiley.com/doi/pdf/10.1111/desc.13095. Available from: https://doi.org/10.1111/desc.13095 [Accessed 2023-12-20].

Miceli-Barone, A.V., Birch, A. and Sennrich, R., 2022. Distributionally Robust Recurrent Decoders with Random Network Distillation [Online]. ArXiv:2110.13229 [cs]. Available from: https://doi.org/10.48550/arXiv.2110.13229 [Accessed 2024-01-03].

Mikhaylova, E. and Makarov, I., 2022. Curiosity-driven Exploration in VizDoom [Online]. *2022 IEEE 20th Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*. pp.000065–000070. ISSN: 1949-0488. Available from: https://doi.org/10.1109/SISY56759.2022.10036273 [Accessed 2024-01-04].

MiniGrid Documentation, n.d. [Online]. Available from: https://minigrid.farama.org/index.html [Accessed 2023-12-11].

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D., 2015. Human-level control through deep reinforcement learning. *Nature* [Online], 518(7540), pp.529–533. Number: 7540 Publisher: Nature Publishing Group. Available from: https://doi.org/10.1038/nature14236 [Accessed 2023-12-11].

MuJoCo Documentation, n.d. [Online]. Available from: https://gymnasium.farama.org/environments/mujoco/ [Accessed 2024-01-04].

OpenAI Spinning Up Documentation, n.d. [Online]. Available from: https://github.com/openai/spinningup/tree/master/spinup/algos/tf1/ppo [Accessed 2023-12-11].

Paquette, P., 2024. Super Mario Documentation [Online]. Original-date: 2016-08-26T02:10:02Z. Available from: https://github.com/ppaquette/gym-super-mario [Accessed 2024-01-04].

Pathak, D., Agrawal, P., Efros, A.A. and Darrell, T., 2017. Curiosity-Driven Exploration by Self-Supervised Prediction [Online]. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Honolulu, HI, USA: IEEE, pp.488–489. Available from: https://doi.org/10.1109/CVPRW.2017.70 [Accessed 2023-12-11].

Pitis, S., Chan, H., Zhao, S., Stadie, B. and Ba, J., 2020. enMaximum Entropy Gain Exploration for Long Horizon Multi-goal Reinforcement Learning [Online]. *Proceedings of the 37th International Conference on Machine Learning*. PMLR, pp.7750–7761. ISSN: 2640-3498. Available from: https://proceedings.mlr.press/v119/pitis20a.html [Accessed 2023-12-20].

Raileanu, R. and Rocktäschel, T., 2020. RIDE: Rewarding Impact-Driven Exploration for Procedurally-Generated Environments [Online]. ArXiv:2002.12292 [cs]. Available from: https://doi.org/10.48550/arXiv.2002.12292 [Accessed 2024-01-04].

Rao, J., Xu, X., Bian, H., Chen, J., Wang, Y., Lei, J., Giernacki, W. and Liu, M., 2022. A modified random network distillation algorithm and its application in USVs naval battle simulation. *Ocean engineering* [Online], 261, p.112147. Available from: https://doi.org/10.1016/j.oceaneng.2022.112147 [Accessed 2024-01-03].

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal Policy

Optimization Algorithms [Online]. ArXiv:1707.06347 [cs]. Available from: `http://arxiv.org/abs/1707.06347` [Accessed 2023-12-11].

Schultz, W., Dayan, P. and Montague, P.R., 1997. engA neural substrate of prediction and reward. *Science* [Online], 275(5306), pp.1593–1599. Available from: `https://doi.org/10.1126/science.275.5306.1593`.

Sovrano, F., 2019. Combining Experience Replay with Exploration by Random Network Distillation [Online]. *2019 IEEE Conference on Games (CoG)*. pp.1–8. ISSN: 2325-4289. Available from: `https://doi.org/10.1109/CIG.2019.8848046` [Accessed 2024-01-03].

Strehl, A.L. and Littman, M.L., 2008. An analysis of model-based Interval Estimation for Markov Decision Processes. *Journal of computer and system sciences* [Online], 74(8), pp.1309–1331. Available from: `https://doi.org/10.1016/j.jcss.2007.08.009` [Accessed 2024-01-04].

Sutton, R.S. and Barto, A.G., 2018. en*Reinforcement Learning, second edition: An Introduction*. MIT Press. Google-Books-ID: uWV0DwAAQBAJ.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, O.X., Duan, Y., Schulman, J., DeTurck, F. and Abbeel, P., 2017. en#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning.

Vigorito, C.M. and Barto, A.G., 2010. enIntrinsically Motivated Hierarchical Skill Learning in Structured Environments. *Ieee trans. auton. mental dev.* [Online], 2(2), pp.132–143. Available from: `https://doi.org/10.1109/TAMD.2010.2050205` [Accessed 2024-01-02].

Von Hofsten, C., 2007. enAction in development. *Developmental science* [Online], 10(1), pp.54–60. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-7687.2007.00564.x. Available from: `https://doi.org/10.1111/j.1467-7687.2007.00564.x` [Accessed 2023-12-20].

Wang, Z., Shi, Z., Li, Y. and Tu, J., 2013. The optimization of path planning for multi-robot system using Boltzmann Policy based Q-learning algorithm [Online]. *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. pp.1199–1204. Available from: `https://doi.org/10.1109/ROBIO.2013.6739627` [Accessed 2023-12-20].

Watkins, C.J.C.H. and Dayan, P., 1992. enQ-learning. *Mach learn* [Online], 8(3), pp.279–292. Available from: `https://doi.org/10.1007/BF00992698` [Accessed 2024-01-03].

Williams, R.J., 1992. enSimple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach learn* [Online], 8(3), pp.229–256. Available from: `https://doi.org/10.1007/BF00992696` [Accessed 2024-01-03].

Yang, C., Yang, J., Wang, X. and Liang, B., 2019. Control of Space Flexible Manipulator Using Soft Actor-Critic and Random Network Distillation [Online]. *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. pp.3019–3024. Available from: `https://doi.org/10.1109/ROBIO49542.2019.8961852` [Accessed 2024-01-03].

Yuan, M., 2022. enIntrinsically-Motivated Reinforcement Learning: A Brief Introduction [Online]. ArXiv:2203.02298 [cs]. Available from: `http://arxiv.org/abs/2203.02298` [Accessed 2024-01-03].

Yuan, M., Li, B., Jin, X. and Zeng, W., 2023. Automatic Intrinsic Reward Shaping for

Exploration in Deep Reinforcement Learning [Online]. ArXiv:2301.10886 [cs]. Available from: http://arxiv.org/abs/2301.10886 [Accessed 2023-12-11].

Zhang, J., Wetzel, N., Dorka, N., Boedecker, J. and Burgard, W., 2019. Scheduled Intrinsic Drive: A Hierarchical Take on Intrinsically Motivated Exploration [Online]. ArXiv:1903.07400 [cs, stat]. Available from: https://doi.org/10.48550/arXiv.1903.07400 [Accessed 2024-01-04].

Zhang, T., Xu, H., Wang, X., Wu, Y., Keutzer, K., Gonzalez, J.E. and Tian, Y., 2021. NovelD: A Simple yet Effective Exploration Criterion.

Appendix A

Network Architecture and Algorithm Hyperparameters

A.1 Network Architectures

A.1.1 CNN Architecture

Attribute	Value
Hidden layer sizes	[32, 64]
Kernel size	[(8, 8), (4, 4)]
Strides	[(4, 4), (2, 2)]
Hidden layer activation function	ReLU

Table A.1: CNN architecture

A.1.2 DQN Architecture

Attribute	Value
Hidden layer sizes	[64, 64]
Hidden layer activation function	ReLU
Output layer activation function	None
Optimiser	Adam
Learning rate	0.001

Table A.2: DQN architecture

A.1.3 PPO Architecture

Attribute	Value
Hidden layer sizes	[64, 64]
Hidden layer activation function	ReLU
Output layer activation function	None
Optimiser	Adam
Actor learning rate	0.0003
Critic learning rate	0.001

Table A.3: PPO architecture

NB. Actor and Critic both have the same hidden sizes, activation functions and optimisers

A.1.4 RND Architecture

Attribute	Value
Hidden layer sizes	[64, 64] ([128, 128, 128] for *multiroom* generalisation)
Hidden layer activation function	ReLU
Output layer activation function	None
Optimiser	Adam
Actor learning rate	0.0003
Critic learning rate	0.001
RND predictor learning rate	0.0001 (0.00005 for *key-corridor* generalisation)

Table A.4: RND architecture

NB. Actor, Critic, RND Predictor and Target all have the same hidden sizes, activation functions and optimisers

A.2 Algorithm Hyperparameters

A.2.1 DQN Hyperparameters

Attribute	Value
γ	0.95
ϵ	1.0
ϵ-decay	0.999
Minimum-ϵ	0.01
Memory size	10,000
Batch size	250
Target model updates every x steps	100

Table A.5: DQN hyperparameters

A.2.2 PPO Hyperparameters

Attribute	Value
γ	0.99
λ	0.95
Clip-ratio	0.2
Actor training iterations	80
Critic training iterations	80

Table A.6: PPO hyperparameters

A.2.3 RND Hyperparameters

Attribute	Value
γ	0.99
λ	0.95
Clip-ratio	0.2
Intrinsic reward weight	0.2 (0.02 for both generalisation tests)
Actor training iterations	80
Critic training iterations	80
RND predictor training iterations	80

Table A.7: RND hyperparameters

Appendix B

Pseudocode

B.1 DQN Pseudocode

Initialise replay memory D to capacity N
Initialise action-value network $\hat{q}_1$ with parameters $\boldsymbol{\theta}_1 \in \mathbb{R}^d$ arbitrarily

Initialise target action-value network $\hat{q}_2$ with parameters $\boldsymbol{\theta}_2 = \boldsymbol{\theta}_1$

Loop for each episode:
Initialise S
Loop for each step of episode:
Choose action A in state S using policy derived from $\hat{q}_1\left(S, \cdot, \theta_1\right)$
Take action A, observe reward R and next-state S'
Store transition (S, A, R, S') in D
For each transition $\left(S_j, A_j, R_j, S'_j\right)$ in minibatch sampled from D :

$$y = \begin{cases} R_j & \text{if } S'_j \text{ is terminal} \\ R_j + \gamma \max_{a'} \hat{q}_2\left(S'_j, a', \boldsymbol{\theta}_2\right) & \text{otherwise} \end{cases}$$

$\hat{y} = \hat{q}_1\left(S_j, A_j, \boldsymbol{\theta}_1\right)$
Perform gradient descent step $\nabla_{\theta_1} L_\delta(y, \hat{y})$
Every C time-steps, update $\boldsymbol{\theta}_2 = \boldsymbol{\theta}_1$

Figure B.1: DQN, Mnih et al. (2015)

B.2 PPO Pseudocode

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

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

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

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

  typically via some gradient descent algorithm.
end for

Figure B.2: PPO, Schulman et al. (2017)

B.3 RND Pseudocode

$N \leftarrow$ number of rollouts
$N_{\text{opt}} \leftarrow$ number of optimization steps
$K \leftarrow$ length of rollout
$M \leftarrow$ number of initial steps for initializing observation normalization
$t = 0$
Sample state $s_0 \sim p_0(s_0)$
for $m = 1$ **to** M **do**
  sample $a_t \sim \text{Uniform}(a_t)$
  sample $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
  Update observation normalization parameters using s_{t+1}
  t += 1
end for
for $i = 1$ **to** N **do**
  for $j = 1$ **to** K **do**
    sample $a_t \sim \pi(a_t|s_t)$
    sample $s_{t+1}, e_t \sim p(s_{t+1}, e_t|s_t, a_t)$
    calculate intrinsic reward $i_t = \|\hat{f}(s_{t+1}) - f(s_{t+1})\|^2$
    add $s_t, s_{t+1}, a_t, e_t, i_t$ to optimization batch B_i
    Update reward normalization parameters using i_t
    t += 1
  end for
  Normalize the intrinsic rewards contained in B_i
  Calculate returns $R_{I,i}$ and advantages $A_{I,i}$ for intrinsic reward
  Calculate returns $R_{E,i}$ and advantages $A_{E,i}$ for extrinsic reward
  Calculate combined advantages $A_i = A_{I,i} + A_{E,i}$
  Update observation normalization parameters using B_i
  for $j = 1$ **to** N_{opt} **do**
    optimize θ_π wrt PPO loss on batch B_i, R_i, A_i using Adam
    optimize $\theta_{\hat{f}}$ wrt distillation loss on B_i using Adam
  end for
end for

Figure B.3: RND integrated with PPO Burda et al. (2018)