UTRECHT UNIVERSITY
FACULTY OF SCIENCE ARTIFICAL INTELLIGENCE

Exploration in Sparse Reward Games

Examining and improving Exploration Effort Partitioning

Wouter van 't Hof

supervised by
Dr. F.P.M. (Frank) Dignum[1]
Professor John Thangarajah[2]
Dr Fabio Zambetta[2]
Michael Dann[2]

1. Utrecht University
2. RMIT University

November 27, 2018

Contents

Abstract Exploration has shown to be difficult in games where the reward space is sparse. The agent has trouble reaching any reward and therefore cannot learn a good policy. One recent approach to this problem is to assist the agent in finding the reward through means of creating subgoals. Subgoals are states for which the agent receives an intrinsic reward for reaching it. This motivates the agent to reach certain areas and indirectly explore more of the environment. While this approach sounds intuitive and shows promise, the method has its flaws. In this thesis, the flaws of this method have been examined and multiple methods to improve the performance have been explored. The representation of the intrinsic rewards has been altered and has shown success. The other methods alter the constraints on which the subgoals are created, namely the relative distance and the visit rate. They both do not improve the performance, but they do improve the quality of the subgoals.

Introduction

Games have been a respected environment for AI for a long time now. It started with chess, and more recently Go (Silver et al., 2017). This is because games are an excellent environment to test AI in and observe the results. The problems faced in games are well translated to problems faced in real-life. The same AI that is used to solve these games can be applied to real problems.

While some AI can be specialized towards a single game, AI that is generally applicable is in a way more interesting. If an algorithm is specialized, it can not be applied to a different problem without changing some aspects. This usually requires domain knowledge and insight into the problem, which might not always be available. General algorithms which do not require knowledge of the problem might, therefore, be more desirable.

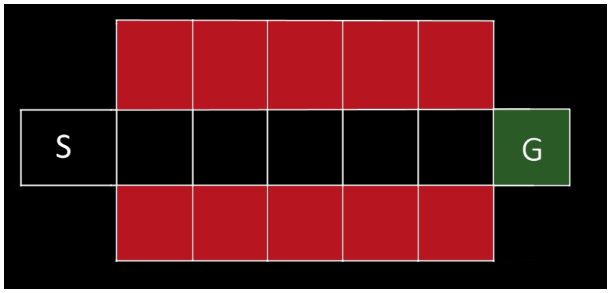
Chess and Go are two examples of games in which AI can reach (and surpass) human performance without any pre-game knowledge. However, not every game has reached this state yet. In a lot of games, AI cannot reach a human-like performance. Unlike most games, Chess and Go are perfect information games with discrete actions. When a game is in continuous time, an AI cannot halt to analyze a move. Nor can an AI observe everything in an imperfect information game. The next step is to improve performance in these kinds of games. While in some of these games AI can perform well by now (such as Pong), a certain type is still very challenging: sparse reward space games.

A sparse reward space game is a game where there aren't many states that give a reward to the player. In a sparse reward space game, the AI has to plan ahead and maneuver its way through difficult levels to reach its goal. These games usually have mechanics unknown to the agent, such as requiring a key to open a door or using weapons to defeat enemies. Finding and reaching these goals while avoiding danger is something that an AI has not been able to reliably do in these type of games.

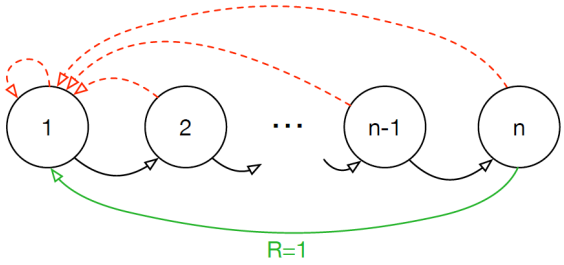
However, progress has been made in the last couple of years. Games in which AI was previously unable to get a positive score now show some improvement. This thesis will discuss some of the techniques used to improve this performance and propose an experiment to further extend one specific approach, namely the approach of Dann et al. (2018), which will be called Exploration Effort Partitioning (EEP).

Problem description

The methods used to create an AI that can learn to play these games are reward driven. By receiving a positive or negative reward, the agent can learn how to behave. This is called *Reinforcement Learning* and will be explained in more detail in a later section. The whole principle behind this approach is to make the agent revisit states that have given him a positive reward and avoid the states that have given him a negative reward. However, this principle fails to work when the agent never reaches a reward to begin with.



(a) The agent starts at S and the reward is located at G. The red spaces indicate a terminal state after which the agent starts again at S. For the agent to reach the goal, he has to randomly sample the right agent six times in a row, which has roughly a probability of $0.25^6 = 0.00024$. However, this cliff/bridge is usually much longer.



(b) A more abstract version of the example in figure 1(a). The agent only receives a reward if it retreats from state n, but to reach state n, the right action needs to be sampled n times. If that action has probablity p, then the probability to reach state n is p^n. This number becomes very small very fast. Image from Schaul et al. (2015).

Figure 1: Examples of the problems with distant rewards

Usually in Reinforcement Learning, the agent starts by randomly exploring until it receives a reward. From then on it starts learning. But when a reward is so far away or difficult to reach that the agent never reaches it by random exploration, the agent never starts learning. This is a troublesome scenario and it's not hard to imagine that this could be the case in real life problems as well. If this issue remains unsolved, the application of Reinforcement Learning to real life problems would be limited. Therefore, it is important to work this problem out in games first such that its approach can be reapplied to the real problems.

The problem can be more easily imagined with the help of a simple picture (figure 1). In these images, the agent only receives a reward if he reaches the far right state. However, the agent does not know the reward lies there, or how the environment looks like at all. Without this knowledge, its best effort is to randomly explore until it receives a reward. The probability that the agent reaches the reward is extremely small. Now imagine a case where there is a tiny reward to the left of the starting state of the agent. With this tiny reward there, the probability of ever finding the larger reward becomes almost non-existent.

The platform and games used to attack this problem and evaluate the results is the Arcade Learning Environment. This environment contains all the 2600 Atari games and is commonly used among researches in this field. Several Atari games contain the problem of rewards that are difficult to reach. A prime example is Montezuma's Revenge, but other games exists as well. In these games, the problem is very present, as regular reinforcement learning does not lead to any learning and improved methods are required.

There are several approaches to this problem. There are multiple approaches to this problem, but the most common one is to aid the agent by adding artificial rewards for reaching 'new' states. If the agent reaches a state that it has not seen before, the agent receives a small artificial reward. With this method, the agent is incentivised to explore more of the environment and is as a result more likely to find these far-away rewards.

This thesis will focus on one of those approaches, namely the one by Michael Dann. This method will be called Exploration Effort Partitioning (EEP). EEP aids the agent by adding subgoals. The agent stores a set of subgoals, which are all dissimilar to each other, and receives a reward for reaching them. At certain time points, the state that has the highest distance to the group of subgoals will be added as a new subgoal to the group. This means that the group of subgoals will expand in the direction that is furthest away from the 'known' area and thus encourage the agent to explore new areas.

EEP manages to have a great performance among the hard exploration games, but the method still has quite some flaws. The subgoals generated are often not as dissimilar as they ought to be. Furthermore, EEP is only tested in games where the rewards are difficult to reach. Since adding artificial rewards is a tricky approach, it is very well possible that the method will not perform as well in games where the rewards lie close. In fact, in some test runs the subgoals have shown to have a negative impact on the performance. In these cases, the baseline of the algorithm without the subgoals performs better. This thesis focuses on improving the subgoal generation and reduce the negative impact that EEP might have on games.

Research Questions

The thesis project will contain two research questions. Mainly because answering the first question is necessary to answer the second question. The two research questions are listed below.

1. **Does EEP perform well across the wide spectrum of Atari games? If it does not, why not?**
2. **Can EEP's performance across the spectrum of Atari games be improved?**

Research Question 1

While EEP performs very well and achieves great results, only a small range of games is tested. There are also several aspects of the algorithm that are slightly flawed and may cause a decline in score in certain situations. Finding these flaws is the core of the first research question.

To find these flaws, firstly a categorization of the different Atari games has to be made. By categorizing them and displaying the algorithm to different types of games, the algorithm will be exposed to more situations and flaws will be more easily detected. For example, the algorithm might perform poorly in games with a dense reward space or games with a high terminal state space.

It is unlikely that EEP will perform well in every game. Once games are found in which it has a decreased performance, the next step is to determine what causes this decrease. Finding the cause and determining the fault in the algorithm might help in adapting EEP, which is the core of the second research question.

Crudely speaking, this research question can be summarized into the following sequence of questions: Where can it go wrong? Where does it go wrong? Why does it go wrong there? Does it go wrong there for other approaches? What should be changed such that it no longer goes wrong there?

Research Question 2

After the first research question has been answered, the flaws of the EEP algorithm will be detected. The core of the second research question is to alter EEP to remove the flaws or to lessen their impact. The two core parts of EEP are the distance measure function (done by the EE function) and the subgoal generation. To improve EEP, a change must be made in one of these two areas.

The goal is to make a change that increases the general applicability of EEP. This means that the change should be robust, and mainly focus on the negative impact that EEP could have. If there the cases where EEP has a negative impact could be decreased, or the negative impact could be reduced, then EEP is more generally applicable. The focus will not be on the strengths

of EEP. While it is likely possible to increase the performance on the target games as well, this is not the main focus of the thesis.

Hypothesis

It is unlikely that EEP will have an increase in performance across all games. Some cases where this is the case have already been perceived. Additionally, in games with a small or completely explored environment, the continuous generation of subgoals sounds incorrect and distracting for the agent. Therefore, the hypothesis to the first research question is that EEP does not perform equally well across the wide spectrum of Atari games.

Concerning the improvement of EEP's performance, the subgoal generation seems to be the most promising candidate to improve. By observing the subgoals it is clear that not all subgoals are dissimilar. The continuous generation on a fixed schedule is very inflexible and could add unnecessary subgoals. Not all the subgoals are of equal quality and this could hinder the algorithm. The hypothesis is that EEP can be improved by changing the subgoal generation section of the algorithm.

Background

Before discussing the hard exploration problem, it is important to first get an understanding of the basics of Reinforcement Learning and to understand how the environment works.

Reinforcement Learning

When an AI learns to solve a problem, it learns what actions it should take in what situations. One of the popular ways to do this is *Reinforcement Learning* (Sutton and Barto, 1998). The principle of Reinforcement Learning is to let an agent learn from interaction with a goal-directed focus. This means to let an agent run in an environment and learn directly from experience. If the agent receives a positive reward from doing a certain action in a certain state, it remembers that doing this is profitable. Likewise, if an agent receives a negative reward it will remember that the particular action in that state should be avoided. In the end, the goal is to know which actions accumulate the highest total reward.

A mapping from each state to a probability distribution over actions is called a *policy*. The task of the agent is to learn the policy that results in the highest total reward. Such a policy is called the optimal policy. The reward is determined by a *reward function*. This function takes the state the agent has reached (sometimes state and action) and returns a reward, which can be negative. The agent internally uses these rewards to create an *action-value function*. The action-value function estimates the expected value of each possible action in a given state. This value is called the Q-value of an action. Actions that lead to a high total reward will have a higher Q-value than other actions.

The perceived worth of the reward is discounted by how long it takes to reach it. For every step/action that needs to be taken to reach the reward, the reward is discounted by the discount factor γ. For example, assume an isolated environment with only one reward of 50 that is 3 actions away from the agent and the discount factor is set to 0.9. The Q-value of that action that leads towards the reward is $0.9^3 * 50 = 36.45$. In this example, the action is assumed to be deterministic. However, if the action is stochastic, that means that the action has multiple outcomes. In that case, the value should be the discounted value of all possible resulting states weighted by the probability of the transition.

When an agent observes new rewards, it should update its Q-values. How much the stored values should be changed according to the new observations is determined by the learning

parameter *alpha*. The higher the learning parameter, the more impact the new information has.

Once the agent has found a policy that gives it a good reward, it can practically follow this policy every run (*episode*). This is called *exploiting*. However, there might be an even greater reward lying just ahead. To find this reward, the agent has to *explore* more of the environment. This creates a delicate problem of exploitation versus exploration. How much do I follow the best-known path to collect good rewards and how much do I explore the environment to possibly find a better reward?

There are several approaches to this problem, but the most popular would be *ϵ-greedy* exploration. This approach introduces one attribute, ϵ, which dictates how much the agent explores. When the agent has to choose an action, the agent chooses a random action with probability ϵ. This means that the agent takes the best-known action (exploits) with probability 1-ϵ, and explores the environment with a random action with probability ϵ.

Reinforcement Learning is a form of semi-supervised learning. The actions chosen by the algorithm are not directly supervised as being correct or incorrect, but there is some sort of supervision in the form of the reward the agent receives from the environment.

Arcade Learning Environment

The environment the project was tested in is the Arcade Learning Environment (ALE). This environment provides a structured way to communicate between an agent and one of the several Atari 2600 games in the environment. The ALE is a common choice for researchers in the field (Machado et al., 2017; Ostrovski et al., 2017; Blundell et al., 2016; Bellemare et al., 2017; Henderson et al., 2017), which means that results among different approaches can be more easily evaluated.

The Atari games contained in the environment cover a wide range of difficulty and are unbiased since they were made by an independent company. Some games are purely reflexive, for example *Pong* and *Breakout*. In these games an AI can score far above human players. Other games require more planning and human players often score better in these games than AI. *Montezuma's Revenge* is a prime example of a game that requires careful planning and a game in which humans score significantly higher than any current AI.

Montezuma's Revenge is a typical game where exploration is difficult and will be used as an example throughout the thesis. A picture of the first room in Montezuma's Revenge can be seen in figure 2. The player controls the character in the middle of the room and needs to collect the key before being able to open the door in the top right. Since a fall of any distance will kill the character, the character has to reach the key by descending the ladder, jumping onto the rope, then to the right platform, descend a ladder again, avoid the skull-enemy and then move up the ladder on the left to obtain the key. The two ends of the lower part of the middle platform are also rotating outwards. Collecting the key is the first time the score increases. Because this is a long path and the agent can die in many different ways along the path, it is very unlikely that the agent will ever reach the key by randomly exploring, which is the standard approach in the conventional reinforcement learning methods.

In the ALE, the observation passed down to the agent (input) is a single 210 x 160 image. Every pixel in the image has an RGB-color vector, which means that the state-space is tremendously large. The action that the AI sends to the environment (output) is one of the 18 possible actions in Atari games. The 18 actions consist of every direction of the joystick (plus the 'no direction') for a total of 9 directions, and for each direction the fire button can be held (active) or not (inactive). Even though some games might not use all those actions, all actions can be chosen and sent by the agent. This means some actions might have no effect in certain games. For example in Pong, if the agent chooses the action 'joystick left & fire', nothing will happen since both 'left' and 'fire' do not do anything in Pong. Whereas 'joystick up & fire' will only

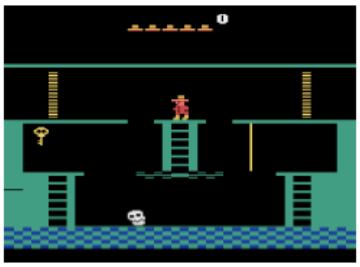


Figure 2: The first room of Montezuma's Revenge (Machado et al., 2017)

move the paddle upwards.

The ALE has numerous settings that can be changed depending on the methodology. One such setting is the life-loss signal setting. In most games, the player has a certain number of lives, usually depicted in one of the corners of the screen. With the life-loss signal on, an episode in reinforcement learning will end when the agent dies once in the game. If the life-loss signal is off, the episode will end when the player loses all his lives and is game-over. Both life-loss on and life-loss off are used by different researchers. The agent will be more scared of death if the life-loss signal is on because this will immediately end the episode. With life-loss signal off, the agent is more indifferent to death. This might sound undesirable, but can be useful in certain situations. For example, if there is a longer path that contains a high risk of deaths, an agent with life-loss signal on, the agent will avoid the path. If the life-loss signal is off, the agent might learn to traverse the path.

Another important aspect of the ALE is its determinism. The ALE is completely deterministic, which means that at the start of a game, a sequence of actions will always lead you to the same state. This allows some very simple algorithms to brute-force their way to a good score. Some randomness can be introduced by the algorithm itself, which is recommended by Machado et al. (2017). They suggest adding a small chance to repeat your last action. This is called *sticky actions* and eliminates any algorithm which relies on the determinism of the ALE.

There are several problems still unresolved in the ALE. Machado et al. (2017) classifies the problems into 5 categories:

1. Representation learning
2. Planning and Model learning
3. Exploration
4. Transfer Learning
5. Off-Policy Learning

All of these categories are actively being researched. The thesis will focus on the Exploration problem. The Exploration problem is about improving the exploration aspect of reinforcement learning such that the agent reliably finds rewards that are difficult to reach.

Relevant Work

In this section, previous research will be discussed. Firstly DQN will be mentioned, which is an important part of most research involving the ALE (Machado et al., 2017; Ostrovski et al., 2017; Stadie et al., 2015) , as well as most other high dimensional environments. Secondly, some previous approaches to the exploration problem will be discussed. And finally, the EEP algorithm will be explained in detail, since the thesis will be an extension of this algorithm.

DQN

Originally, most reinforcement learning approaches used a linear action-value (Q) function to estimate the values of certain actions. This was done because non-linear approximations were unstable and unreliable. However, in order use a linear Q-function, the environment needs to either have a small number of dimensions, or you need to be able to abstract some useful features from the environment. The ALE environment is non-tabular and does not have these properties, which made experimenting difficult. While it is possible to use a linear function approximator in the ALE (Liang et al., 2016), such an approach is usually too specialized towards the environment.

A few years ago Mnih et al. (2015) discovered Deep Q-Network (DQN). DQN combines reinforcement learning and a deep neural network to create an abstract understanding of the environment. They were able to fix the usual instabilities that come with a non-linear function approximator with a few core concepts, namely experience replay and target values.

Experience Replay is performed by storing experiences at certain time points and later use random samples of these experiences to update the Q-values. The target values are the values to which the Q-Values are adjusted. These target values are only periodically updated to reduce the correlations between the Q-Values and the target.

Zahavy et al. (2016) took a closer look at DQN and argued that it is possible to create temporal abstractions from the DQN. These temporal abstractions break the state space into multiple parts and tackle the curse of dimensionality problem. This can further be used to create a model of the game and determine different subgoals.

DQN has fixed the unreliability of non-linear function approximators and has provided the possibility to apply reinforcement learning to higher dimensional environments. It has proved to be a viable approach and has become the standard function approximator in research in the ALE field.

Previous Approaches to the Exploration Problem

There have been many attempts to solve the exploration problem. The two most successful approaches have been a count-based novelty method that uses density models and a hierarchical reinforcement learning method, which breaks the problem into multiple subtasks. Other approaches such as *Prioritised Experience Replay* (Schaul et al., 2015) have improved performance in some games with sparse rewards matrix, but failed to reach any progress in games with a delayed reward such as Montezuma's Revenge.

The count-based novelty approaches use a density model to determine novelty of actions or states. If the agent does an action or reaches a state with high novelty, the agent receives an *intrinsic reward*. An intrinsic reward is treated like a regular (extrinsic) reward and will steer the agent back in that direction. This means that actions or states with high novelty will be visited more often until the novelty wears off.

The difficult part of these approaches is how to define novelty. Determining when a state is novel is crucial to the success. One of the techniques to determine novelty is by using the visual similarity of two states (Bellemare et al., 2016; Ostrovski et al., 2017). From the visual similarity, they can derive a *pseudo-count*, which can then be directly used to determine the novelty. This technique received some good results, but the assumption that visual similarity determines novelty is one which is often not true.

Another approach which uses the novelty concept, but is not count-based is the approach of Stadie et al. (2015). They used unsupervised learning techniques to create a predictive model to determine the novelty. Instead of using a count-based approach, their model assigned higher novelty bonuses to state-action pairs of which the model had a more inaccurate prediction. The

action-state with the most *information gain* receives the most exploration bonus. While this is not directly the same as count-based, information gain and state count have a high correlation. Ostrovski et al. (2017) even argues that intrinsic rewards and count-based methods are two sides of the same coin. However, the model uncertainty is not a good indication for novelty in stochastic environments. The algorithm not working in stochastic environments heavily limits its usefulness.

The other succesful approach to the exploration problem is the hierarchical approach (Kulkarni et al., 2016; Roderick et al., 2017). The hierarchical approach reportedly solves the first room (figure 2) of Montezuma's Revenge faster than the count-based methods (Kulkarni et al., 2016). It firstly creates a goal at the key, but once it fails to reach it in one go, it splits the goal into multiple steps. Instead of reaching the key in one go, the agent's goal is to reach the bottom-right ladder first. This can be done by abstracting the state representation. However, there is currently no way to do something like state abstraction on environments with the complexity of the Atari games. Kulkarni et al. (2016) used hand-crafted features such as 'the agent has the key' to guide the agent. This assigns a requirement to the approach that can often not be fulfilled.

Exploration Effort Partitioning

One recent approach to the exploration problem is the approach of Dann et al. (2018), which will be called Exploration Effort Partitioning (EEP). This approach is based on the novelty concept that gives the agent intrinsic rewards upon reaching areas that have rarely been explored. This is done by creating *subgoals*, which is a group of states which are all dissimilar to each other. Once the agent reaches such a subgoal, the agent receives an intrinsic reward, called a *pellet reward*. The value of this reward is based on the frequency it has been visited. This should reward the agent for going to rarely seen states, which usually directs it towards unexplored areas.

The biggest challenge of this approach is to find a reliable function that can measure the similarity between two states. The function that handles this in EEP is called the *Exploration Effort* (EE) function in the paper. The EE function is a neural network which uses a mixed Monte-Carlo (MC) return function for its target values. This mixed Monte-Carlo (MC) return function is inspired by Bellemare et al. (2016).

The EE function takes two frames from the experience cache which are not too far apart and then looks at the actions taken to get from the first frame to the second frame. Then every possible action (out of the total 18) gets assigned an MC return value. This value is based on the actual actions taken between the frames. If a certain action is performed more often than expected (according to a uniform random distribution), the action receives a higher MC return value. This means that actions with a high MC return value are more likely necessary to move between the two retrieved frames. Thus, in summary, the EE function takes two states as input and returns a list of 18 values. This list represents how likely actions are necessary to move between the two states.

The list of the likelihood of actions can then be used to compare two states for similarity. The two states being compared both generate action likelihood lists to every existing subgoal. If the lists are similar to each other, then the states must be similar as well.

After the EE function has been trained for 2M frames, EEP starts to generate subgoals. While exploring, the program saves the state that is the most dissimilar to all the subgoals and stores it as the subgoal candidate. After a certain amount of steps, the subgoal candidate is added as a subgoal. The amount of steps between the generation of a subgoal increases incrementally after every generation. The first 20 subgoals generated by the algorithm in Montezuma's Revenge can be seen in figure 3.

The subgoals contain a pellet reward. This pellet reward can be collected once per episode

by the agent upon reaching it. This reward starts at 1 but diminishes every time the subgoal is reached. A subgoal is 'reached' if it is the closest subgoal to the current state. This reward is what motivates the agent to reach the subgoals and indirectly explore more of the environment.

EEP has achieved remarkable results in the harder exploration games of Atari. It is the strongest subgoal-based algorithm currently known. Games like *Montezuma's Revenge* and *Venture* reached scores which are comparable to the visual-based methods. Furthermore, a subgoal-based approach is one that aligns with the natural way of human thinking: explore what is lesser known. The method alters the reward function to a lesser extent than previous methods based on visual novelty (Bellemare et al., 2016; Ostrovski et al., 2017). And unlike the hierarchical agents of Kulkarni et al. (2016) and Roderick et al. (2017) the method does not require handcrafted subgoals.

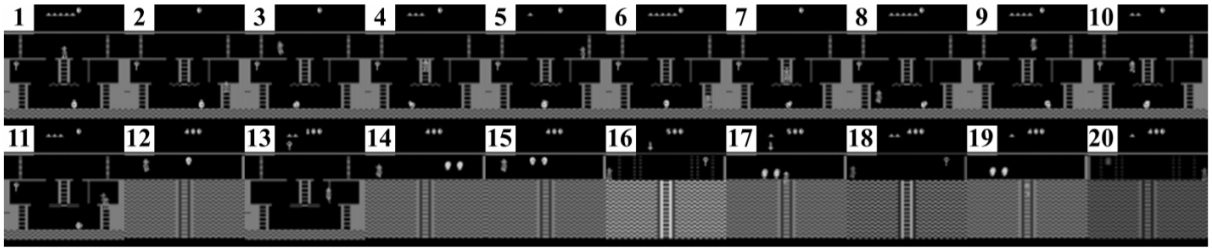


Figure 3: The first 20 subgoals of EEP in Montezuma's Revenge (Dann et al., 2018).

Methodology

Analysing EEP

The EEP algorithm manages to get remarkable results in games like Montezuma's Revenge and Venture, where the reward space is sparse. This is because the algorithm is specifically designed to address the issues in these games. It will direct the agent towards new areas where it will hopefully find rewards. However, it is important to not limit the applicability of EEP to only games where the reward space is sparse. One can imagine that if the environment is filled with rewards, the agent does not need help finding rewards. EEP will probably not bring a large improvement in such games and that is acceptable. It should, however, not have a negative impact. Since the algorithm had not been tested on games with a dense reward space, several games have been run to test the general applicability of EEP.

Besides testing the general applicability of EEP, running these tests will also reveal some more information about the flaws of EEP. By exposing the algorithm to more cases, it is possible to observe scenarios in which EEP has not been tested before. This can bring light to unseen behavior or flaws. This is an important step, because as stated in the hypothesis, it is suspected that EEP will not perform equally among all games.

Due to time constraints, not every game could be tested. Therefore, it was necessary to make a selection of games. To make a varied selection of games, a taxonomy on the games is required. This taxonomy was made with the help of Ostrovski et al. (2017). Firstly the the games were split into two categories, one with games on which it is hard to learn a good policy and one on which it is easy to learn a good policy. It is important that the algorithm does not falter when provided with an easier task/game and will still provide an acceptable performance. The harder games are further split into games with a dense reward matrix and games with a sparse reward matrix. Games with a dense reward matrix like *Ms. Pac-Man* or *Qbert* are filled with rewards and the agent will receive one frequently. These rewards do not necessarily have to be large, they only need to be frequent. In sparse reward space, like *Freeway*, the rewards are very

spread and the agent can often go a long time without finding one. The value of the reward is again not very important. The difficulty and the density of the reward space got split into two categories, no distinction was made between games that are slightly difficult/sparse or very difficult/sparse.

Easy Exploration			Hard Exploration	
Human-Optimal		Score Exploit	Dense Reward	Sparse Reward
ASSAULT	ASTERIX	BEAM RIDER	ALIEN	FREEWAY
ASTEROIDS	ATLANTIS	KANGAROO	AMIDAR	GRAVITAR
BATTLE ZONE	BERZERK	KRULL	BANK HEIST	MONTEZUMA'S REVENGE
BOWLING	BOXING	KUNG-FU MASTER	FROSTBITE	PITFALL!
BREAKOUT	CENTIPEDE	ROAD RUNNER	H.E.R.O.	PRIVATE EYE
CHOPPER CMD	CRAZY CLIMBER	SEAQUEST	MS. PAC-MAN	SOLARIS
DEFENDER	DEMON ATTACK	UP N DOWN	Q*BERT	VENTURE
DOUBLE DUNK	ENDURO	TUTANKHAM	SURROUND	
FISHING DERBY	GOPHER		WIZARD OF WOR	
ICE HOCKEY	JAMES BOND		ZAXXON	
NAME THIS GAME	PHOENIX			
PONG	RIVER RAID			
ROBOTANK	SKIING			
SPACE INVADERS	STARGUNNER			

Figure 4: A rough taxonomy of the Atari games (Ostrovski et al., 2017)

In addition to selecting games from a good mix of difficulty and reward density, games were selected based on how effective and polarizing the algorithm of Ostrovski et al. (2017) was on the game. The approach of Ostrovski et al. (2017) is similar in that it also uses novelty to steer the agent towards new areas. Their novelty was determined by the visual differences between screens. Even though the novelty is based on different aspects, the fundamental approach is similar enough that the performance was taken into consideration. An added similarity is that Ostrovski et al. (2017) also ran tests using a MonteCarlo return function, which can have quite a big impact depending on the game. The similarity of the approach and the addition of the MonteCarlo return function made their results highly influential in the selection of games.

The final selection of games can be found in table 1. The reason that the bottom right field is empty is simply because there are no games with a sparse reward space where the intrinsic rewards decreased the performance.

	Easy	Hard Dense	Hard Sparse
Artificial rewards increase performance	Boxing	Hero Alien	Montezuma's Revenge Gravitar
Artificial rewards decrease performance	Atlantis Battle Zone Riverraid	Amidar Qbert	

Table 1: A rough taxonomy of the selected games to be tested

To determine the influence of the pellet reward scheme, both the baseline of EEP was run as well as the full algorithm on the above mentioned games. The results of these runs can be found in the results section. From these tests it is clear that the pellet reward scheme of EEP often has a detrimental effect of the performance, especially on games with a dense reward matrix. The only improvements were found in games with a sparse reward matrix.

Before diving into the possible cause of the negative impact of the pellet reward scheme, it should be noted that most of these results are still up to par with other approaches and some are even state of the art. Especially if you look at the baseline, the results are unexpectedly high. This means that the baseline of the algorithm has some slight changes that greatly improve the

performance. These changes shall be shortly discussed before continuing to focus on the pellet reward scheme.

Baseline

More often than not, researches do not publish the code of their complete program. Thus, every time research is done in this field, most of the code is built up from scratch. This results in every program being slightly different, since nobody writes code in exactly the same fashion. This is especially true when multiple methods and techniques are combined. The integration and adaptation of every method or technique has some personal twists to it. On top of that, most of these methods include multiple parameters to be set. There is no guarantee that these parameters are set equally among all researches. For example, in Reinforcement Learning, every iteration the agent has to make a decision whether to explore or exploit. There are multiple methods to determine which decision the agent should make. The most common method is *ϵ-greedy*, but there exists multiple other methods as well, such as *Softmax Action Selection* or *Optimistic Initial Values* (Sutton and Barto, 1998). On top of that, these methods also have a parameter to influence the bias towards exploration and exploitation. In fact, EEP originally started with the Softmax Action Selection method. However, the method was changed to ϵ-greedy after it was tested and proven to provide better results. Since there are so many little options and small parameters, it is unlikely that the same algorithm, written by different researchers, actually performs the same.

This is exactly what happened with EEP as well, and the small changes and personal choices created a very solid baseline. The combination of tweaks, personal choices and parameter settings just seemed to work extremely well, and the baseline without any additional concepts already has an amazing performance against a wide spread of games. It's difficult to exactly tell which of these settings caused the baseline to shine, but below are some unusual changes that are likely to have an impact.

One of the changes to the baseline is unique in that it can't be found in any of the literature. This change is also presumed to have the biggest impact and might be the sole reason of the high performance of the baseline. The change is the following: **After not finding a reward for several frames, explore full time**. After reaching and collecting all of the currently known rewards, the agent often has no real idea of where to go next. This means that exploring from that point is a valid option. This rule somewhat reinstates that idea. Another results from this rule is that the agent will no longer get stuck in corners. If the agent doesn't progress, it will always return to full on exploration.

The exploration parameter epsilon has also been changed to a lower value than standard practice. The common value of epsilon is 0.1, while the value of epsilon in EEP and the baseline is 0.01. This was originally done to compensate for the inherit exploration of EEP, which will be discussed in the next section. However, it seemed to increase the performance of the baseline as well. Recently, more researchers has also taken this change into account and the value of 0.01 for the learning parameter epsilon is starting to become more common.

Detrimental effect of the Pellet Reward Scheme

As mentioned earlier, the pellet reward scheme often has a negative influence on the score compared to the baseline. To understand why this is happening, it is important to return to the original principle behind the pellet rewards. The idea behind the pellet reward is to reward the agent for reaching new areas. In doing so, the agent will no longer keep exploring randomly till he finds a reward, but rather try to find all the unknown areas in the environment. This will in turn most likely lead to finding an area or state that contains a reward. This principle holds strong even in the case where the reward space is dense, because even if the agent has already

found rewards in the environment, exploring the unknown remains important.

One could argue that once a good reward or path has been found, the agent should drop the subgoals or intrinsic rewards altogether. The reasoning behind this, is that the purpose of subgoals is to make the agent find a reward and if the reward is found, subgoals are no longer needed. This argument relates to the classic problem of exploration versus exploitation. The core of the problem is that an agent should not explore too much nor should he exploit too much. By exploring too much, the agent will neglect most of its learning and barely collect rewards. Contrarily, if the agent exploits too much, it may never discover that there might be higher rewards nearby. This reasoning can also be applied to maintaining the pellet reward scheme even after substantial rewards have been found. By continuing to explore the unknown the agent can be more certain that there is no better policy to follow. Ideally, the whole state space would have been explored in a timely fashion, but this is often not realistically possible.

The previous paragraphs suggest that the pellet reward scheme indirectly promotes exploration. This is indeed the case, since getting to newer or *unexplored* areas gives a small reward to the agent. Thus, the agent has more incentive to stray off the best found policy (exploitation) to *explore* other areas. To compensate for this, the exploration parameter ϵ has been slightly lowered in the exploration function for the reinforcement learning algorithm. This resulted in a better overall performance.

The implementation of this pellet reward principle must be done carefully., since adding artificial rewards is a delicate issue. It's very difficulty to predict the behavior of an agent and a small change can have a large impact in an unforeseen way. The principle is implemented by means of intrinsic rewards. This seems to be the most straightforward way, since the agent is driven by rewards. However, adding these artificial rewards can radically change the behavior of the agent and must be done with care. Adding random artificial rewards will distract the agent from its true objective. This will likely result in the agent wandering around more randomly and dying to the environment before making progress.

After the observations from the algorithm on Montezuma's Revenge, it can be observed that the subgoals and pellet rewards do steer the agent into new areas and improve the exploration. However, in other games, the addition of the pellet rewards has a negative effect. This means that sometimes the subgoals fail to fulfill their purpose discussed earlier. While it is hard to know exactly what is happening, it is reasonable to assume that the subgoals are of different quality. When the pellet reward scheme fails to improve the score, it is likely that the generated subgoals are detrimental and the agent is better off without them. However, it is unlikely that all of the subgoals are detrimental. Presumably, some subgoals are detrimental, while other subgoals do in fact aid the agent. Identifying when a subgoal is helpful and when a subgoal has a negative impact is key to improving the method.

A subgoal is detrimental if it does not contribute to finding new areas. A subgoal that fails to do this is ultimately just a distraction to the agent. A random intrinsic reward with no purpose. It is possible to determine when a subgoal does not help in finding a new area. By examining the subgoals afterwards, it can be determined which subgoals helped in finding new areas and which subgoals barely contributed. It is, however, more important to determine *why* a subgoal did not lead to a new area. Answering the why-question is a more preemptive way of handling this problem and should be prioritized over the mitigation approach. Unfortunately, it is more difficult to answer why a subgoal did not lead to a new area. Previously it was explained that a subgoal is generated according to the maximum distance to the current known network. Intuitively, such a subgoal should represent the area furthest away and thus most novel, but this is not always the case. Another issue is that a subgoal may have lead to a new area in the beginning, but that area is no longer novel later in the run. This second case shall be shortly discussed before the first and more pressing issue will be investigated.

Weak Subgoals

A subgoal may lead to an area that used to be new, but no longer is. These subgoals did exactly what they were supposed to do, but are at this point no longer useful. These subgoals should no longer have an impact on the agent. If the impact of these subgoals does not fade, the subgoals will distract the agent and drag it back to previously explored areas. For example; In Montezuma's Revenge, the player respawns in the room he dies in. Without diminishing impact of the subgoals, the agent is often found returning to previous rooms solely to collect the pellet rewards of older subgoals. This diminishing impact is implemented by the decreasing reward that subgoals generate over time. In EEP, the pellet generated from the subgoals is dependent on the visit count of that subgoal. The more a subgoal is visited, the lower the reward from the pellet. The exact formula to determine the pellet reward is

$$\beta / \sqrt{max(1, visits)}$$

, where β is a parameter which is usually set to 1, and visits is the number of times the corresponding subgoal is visited. This scales nicely with the novelty of the area. The more often it is visited, the less novel it is.

The other case is that a subgoal never led to a new area to begin. This case is much trickier to deal with. Statistically it is difficult to determine whether a subgoal has lead or will lead to a new area. Luckily, this can be done by simply looking at the subgoals with human eyes. It is easy to see that sometimes a subgoal gets created which is very similar to an older subgoal [ref figure]. These are often unnecessary and will only interest the agent in areas well explored. With some domain knowledge, it is also possible to spot subgoals that lie between two older subgoals. In this case, the subgoal lies on a path that is again well explored and will not assist the agent in finding new areas.

These subgoals are problematic because they are random intrinsic rewards that will distract the agent, as explained earlier. TO give an example to illustrate this; In Boxing, EEP generates subgoals all around the ring. The agent will run circles in the ring trying to collect these subgoals, while ultimately, he should be focusing on boxing with his opponent.

One way to deal with impracticable subgoals is to measure their usefulness during the run and then remove a subgoal if it appears to be not useful. However, this is not the direction chosen for a few reasons. One reason is that it is not trivial to measure the usefulness of a subgoal. One could argue that the visit count is a good indication of how useful a subgoal might be, but a subgoal with a high visit count may as well have been a good subgoal that has fulfilled its purpose. Removing a subgoal that has fulfilled its purpose impacts the structure of the information that the agent has and is undesirable. The distance of candidate subgoals will be shifted if a subgoal is removed. A subgoal that is currently not contributing much might also have an impact later in the run, and removing those subgoals prematurely will also likely negatively impact the performance. All this considered makes removing subgoals an impracticable approach.

Instead of afterwards removing bad subgoals, a better approach would be to prevent these bad subgoals from ever being made. To do this, it is necessary to be able to predict when a subgoal is going to be bad. In other words, it requires some criteria that will determine the novelty of the subgoal. How well will the subgoal complete its task of leading the agent towards a new area. As discussed moments ago, a subgoal is likely to be unimportant if it is too similar to another subgoal or if it lies between two other subgoals. The criteria should be made with these two case in mind.

Similarity can be expressed in distance. If two subgoals are very similar, the distance between each other is very small. Contrarily, the distance between two very different subgoals is very high. Note that this is not related to the physical distance in the game itself. EEP contains

a function to measure this distance (similarity), namely the EE function, which is one of the main contributions of the algorithm. Since distance represents similarity, and there is a method to measure this distance, it seems obvious to use this distance as a criteria to generate subgoals.

This is exactly what EEP currently does. It creates the subgoal with the highest distance to the current network of subgoals. It does this on a fixed schedule. While this simple method may seem intuitive, it does not always create a good subgoal. They key reason for that is the fixed schedule the subgoals get generated on. EEP will always generate a subgoal when it is time to create a subgoal. This means that if no good subgoal has been found in the given time, it will generate a useless subgoal. Furthermore, if a good subgoal has been found right after the creation of one, it might take a while before it is added to the network. Thus, to improve the generated subgoals, it makes sense to move away from this fixed schedule.

Instead of a fixed schedule, it would be ideal to create a subgoal straight away if it is too dissimilar form the current network. This begs the question: When is a subgoal too dissimilar? This is not a question that is trivial to answer, but intuitively, the distance plays a role again. To be more precise, a subgoal is dissimilar enough if the distance exceeds a certain threshold. However, finding the right value for this threshold is the issue. The distances calculated between states widely differ from game to game. Having different thresholds for different games means that your parameters are specifically tuned to a environment, which is bad practice and undesirable. However, choosing one fixed value for this threshold means that it will not generally perform well across multiple games or environments.

If the threshold cannot be fixed, then it should be dynamic or adaptive. The threshold should be formed based on known or measured information. This led to the idea of a *relative distance constraint*. This constraint creates a distance threshold based on the distances previously measured. This ensures that the criteria is adaptive to the game and more generally applicable. The core idea behind the constraint is that by placing the distance of a state on a relative scale, you can determine the 'novelty'. If a state is much further away than usual, it is likely novel.

Another measurement that came into consideration is the visit count of a state. A state that is often visited during runs is probably on a known path. Such a state does not lead to a new area and would be unsuitable as a subgoal. If we can roughly estimate that a state would be visited a lot, then it can be rejected as a subgoal, even if its distance is high. If a state would barely be visited, it is possible for the state to lie off the usual path and thus direct towards a newer area.

Model

In this section all the changes made to EEP will be discussed. Firstly the change in the pellet reward representation and afterwards the different approaches to changing the subgoal generation will be discussed.

Hidden Layer Adjustment

The change to the hidden layer was not the first change that was explored. Initially, the generation of the subgoals was taken under examination. However, during the extensive tests of the baseline versus the pellet reward scheme, some peculiar behavior was noticed. An agent would often backtrack to earlier subgoals that were not collected. For instance, if the agent in Montezuma's Revenge would not spawn in the first room, it will later in the episode return to the first room to collect all the subgoals there. Even the very first subgoals which had been visited hundred thousands of times. This means that the influence of the subgoals did not wear off as it should have.

In the previous section it was mentioned that the influence of a subgoal should wear off over

time as the area corresponding to the subgoal will lose its novelty. This was done by decreasing the reward with every visit, such that after many visits, the reward should be nearing 0. This idea is still solid, however, it was not correctly represented in the complete model.

EEP contains a hidden layer in the neural network to represent the pellet rewards. In this hidden layer, every pellet reward is represented by means of a 0 or a 1. A 1 would mean that the pellet reward has been collected and a 0 means it's still available to the agent. This originally represented the fact whether a subgoal has been visited or not, but this is ultimately the same information. A binary representation intuitively work for state visitation, but it does not carry all the information about a pellet. Because it also represents the collection of a pellet reward, it did not make a lot of sense to keep the representation binary. A pellet reward deteriorates over time and after many visits has a value of close to 0. Whether this pellet reward of 0 has been collected or not should not have much influence over the agents decision.

This notion led to a change of the pellet reward. The change from not collecting a deteriorated pellet reward to collecting a deteriorated pellet reward should be minimal. In fact, a reward can become worth so little that it might be more accurate to say that there is no reward at all. This change should not be an equally big binary toggle from 0 to 1 as the other pellet rewards. Thus, the change in the hidden layer was adjusted, such that if the pellet is collected, the corresponding value in the hidden layer is changed to the value of the pellet reward, rather than 1. If a pellet was only worth 0.001, then if it is collected, the corresponding position in the hidden layer changes from 0 to 0.001.

This accurately translates the value of a pellet reward to the neural network such that the agent can behave accordingly. With this change the agent will no longer chase after subgoals which have fulfilled their purpose and no longer mean anything.

Relative Distance

The relative distance constraint aims to provide a dynamic threshold on which it is possible to accept or reject subgoals. The idea originated from trying to step away from a fixed schedule. If there exists criteria on which one can accept a subgoal, then the subgoal can be accepted or rejected accordingly at every step. This eliminates the need of a fixed schedule and removes some of the aforementioned problems associated with it. There is no longer the possibility of a long waiting time after a good subgoal has been found and there is no longer a need to add a bad subgoal.

The Relative Distance constraint measures for every visited state the distance to its nearest subgoal. Comparing distances of states of one subgoal to the distances of states of another subgoal does not make sense, since they wildly differ and are not relevant to another. Instead, states that neighbour the same subgoal should be compared. By comparing the states surrounding the same node, it is possible to examine them in a meaningful way. Therefore, distances get grouped by nearest subgoal in the algorithm. The full algorithm is described below:

1. Store the distance to the closest subgoal in a list corresponding to that subgoal
2. If the subgoal has the highest distance measured, store it as the 'best found subgoal'
3. Every few frames (5k), check if the best found subgoal is far away enough from the nearest subgoal to be considered novel
4. If it can be considered novel, add it as a subgoal, otherwise continue

By comparing local distances, it can be observed when one distance is significantly larger than the others. If a node with a significantly larger distance is found, the node likely lies in another area. Since this other area does not have a subgoal related to it, the found node should

turn as a subgoal and act as a reference point to the newly discovered area. By progressing this way, the agent should create new subgoals only in new areas.

To do this properly, it needs to be determined when a distance is significantly larger. For this, a normal distribution was used. This is not completely correct since the visited nodes are not normally distributed around the subgoals, but the distribution is good enough for this approach. Every 5k frames a normal distribution would be fitted on the current measured distances, and if the chance of one of these distances occurring is smaller than a predefined threshold, it is said that the distant is significant. If multiple distances were significant, only consider with the largest distance.

However, during testing a few problems became clear. First of all, the EE function is not always accurate enough in estimate distances. By putting more focus on this function, the inaccuracy becomes more apparent. Another issue is that the visited states around the subgoals are not completely normally distributed. While at the beginning the agent is mostly randomly exploring and the states lie all over the place, after some learning this is no longer the case. The agent quickly then moves between subgoals. This causes states that lie perfectly between two subgoals to theoretically also have a high relative distance.

A distinction had to be made between a state that lies at the edge of the explored area, and a subgoal that lies in between two subgoals. While trying to tackle this problem, a new criteria came to mind. One defining difference between these two types of states is the amount of times it is visited. This feature was made into its own criteria, separate from the relative distance rate.

Visit rate

By examining the visit rate of a potential subgoal, it is possible to determine if it lies in a known area. A subgoal candidate that is visited in a lot of runs likely lies on a known and traversed path of the agent. Such a subgoal would not direct the agent towards new areas and should not be generated. On the other hand, intuitively, if a subgoal is barely visited it is likely to not lie on the regular path of the agent and is worth exploring more and thus a profitable subgoal.

To implement this, a candidate subgoal needs to have some sort of examination period in which the visit rate is observed. If after or during this examination period the visit rate is too high, the candidate lies on a known path and is rejected. Contrarily, if the visit rate is low enough, the candidate lies off the known path and is accepted and a new subgoal is generated.

The visit rate threshold on which a subgoal is accepted or rejected is set to 0.5. However, this constraint should me more lenient at the start of the process. This is because of the inaccuracy of the EE function. Because the accuracy of the EE function is not perfect, there is a slight probability that a state will be determined to be closest to a wrong subgoal. This is troublesome at the beginning because there is only one (or a few) other subgoals. It is highly likely that during an episode, one of the many states visited will be wrongly categorized as near the wrong subgoal. This means that the visit rate will be estimated higher than it should be. To compensate for this, the threshold is set to the following function: $max(0.5, 1 - (0.02 * x^2)$, where x is the number of subgoals. This leads to the thresholds at different subgoal counts that can be found in table 2.

Initially the examination period was set to a certain amount of episodes, namely 100. If the visit rate after these 100 episodes lies above the aforementioned threshold, the candidate would be rejected. However, it is possible to fasten the process quite significantly using bounds. For example, if out of 20 episodes, the candidate was visited 20 times, it is likely that the visit rate will be too high and the candidate would be rejected. Likewise, if the candidate was visited 0 times out of 20 episodes, it is highly likely to be accepted. These bounds were computed with the Adjusted Wald interval. The Adjusted Wald interval was chosen because it is simple to compute and is more accurate for smaller sample sizes than the regular Wald

#subgoals	Threshold
1	0.98
2	0.92
3	0.82
4	0.68
5	0.5

Table 2: Initial visit rate thresholds

interval. The confidence interval was set to 95%. This means that if the lower bound of the interval is above the threshold, the chances of the candidate being accepted is below 5%. By repeatedly calculating these bounds and checking if they are completely above or below the threshold, the candidate can be accepted or rejected at a much earlier episode than the final 100.

The method works as follows:

1. Keep track of how many episodes the current candidate has been the best candidate and how often it would be visited
2. If the current state would be closest to the candidate subgoal, add one to the would be visited count
3. Every few frames (5k), determine the visit rate of the candidate subgoal
4. If the visit rate is high enough, generate the subgoal. If it is low enough, reject the candidate

However, when there are some issues hidden in this method. There are some instances in which a candidate subgoal would have a low visit count, but not lead to a new area. These instances lead to the creation of subgoals which ultimately distract the agent. One of these instances occurs in dead ends. Dead ends usually dont lie on the agents path since they dont lead anywhere, but they might get randomly explored every now and then. This leads to low visit counts, which creates subgoals in dead ends, which in turn leads to the agent wasting them there.

The bigger issue is in games where the respawn location changes. If the agent spawns at a different location after it dies, the paths will differ from previous lives. For example, in Montezuma's Revenge, you respawn in the room you die in. After the agent has collected the key and left the door, it will likely never revisit the first room. This causes states in the first room to have a low visit rate, without being off the known path. This issue is directly linked to the episode termination setting in the ALE which is discussed in an previous section. The EEP currently starts a new episode after the agent loses a life. If the setting was switched the other way, the new episode would start after game over. Since there are no Atari games which start each at a different location, this problem wouldn't exist with the setting set to terminate on game over.

Nonetheless, the visit rate criteria is more elegant and performed better when the aforementioned issues are not present. Therefore, the decision was made to change the main criteria to the visit rate and try to find solutions to the problematic cases. The tried solution was to incorporate the relative distance criteria as a secondary criteria.

Combined criteria

By examining both the relative distance criteria and visit rate criteria individually, it became clear they both have some issues. The biggest issue of the distance criteria was that it cannot

differentiate between states that lie far off the known area and states that lie within the known area. And the bigger issue of the visit rate is that different episodes might take different paths and thus visit certain states in known areas less often.

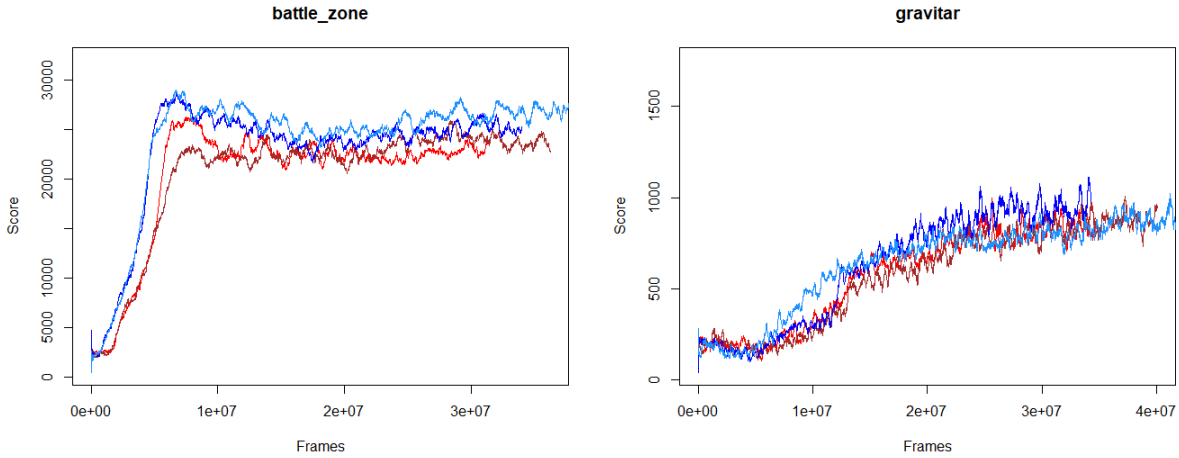
By combining the two criteria, the issues become less apparent. Because the visit rate is slightly more accurate and elegant, that criteria is chosen as the main criteria. If a candidate passes that criteria, the candidate gets tested on a softer relative distance criteria. By doing this, the candidates that have a low visit rate due to variation in spawn locations still have to pass the relative distance check. These candidates often dont pass this check as they lie near familiar subgoals. On the other hand, the weakness of the relative distance criteria is less apparent since subgoals on the known path are less likely to pass the visit rate constraint.

There is still a slight chance that a bad subgoal candidate satisfies both these constraints. For this to happen, the game needs to have a variable spawn location and the candidate subgoal needs to lie on a path that only gets traversed once and it needs to lie far enough from known subgoals, which is most likely between two subgoals. This can still happen, however, the probability of such a bad subgoal slipping through is significantly lower than the older methods.

Results

Before the results are shown, it is important to note that these performance graphs do not convey the full information. As is custom, these graphs show the average score of one full episode, containing all lives till game-over. Since EEP terminates and starts a new episode on life-loss rather than game-over, some information gets lost. For example, in Montezuma's Revenge, whether the agent retrieves the key and opens the door all in its first life or whether it needs multiple lives to achieve this makes quite a difference. An agent that can in one life dodge navigate the platforms, dodge the skull and make it back to the door is way better trained than an agent that suicides into the skull to make it disappear and suicides after getting the key to get back to the starting position. During training this would be a score of 400 in life one and zero in life 2 and 3 versus a score of 0 in life 1, 100 in life 2 and 300 in life 3.

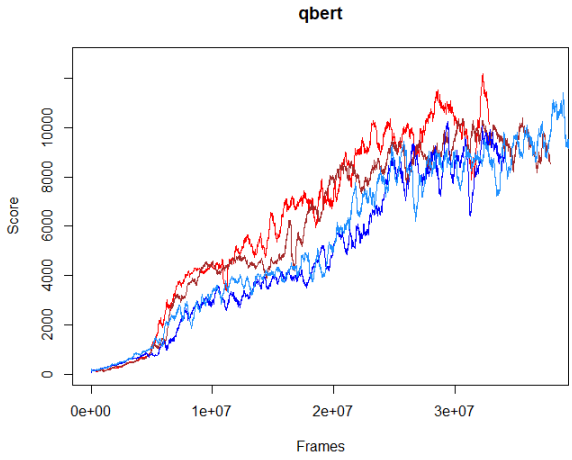
To start evaluating the results it is important to first have a recent and actual reference point to compare to. Therefore, the first runs were made using EEP in its original state. Both the baseline of EEP was tested as well as the full algorithm with pellet rewards. Since the algorithm has been shown to work on the targeted games Montezuma's Revenge and Venture, these first tests were performed on other games. This provides an image on the general performance of EEP and its robustness. The resulting performance graphs can be observed in the figure below. The blue lines indicate baseline runs of EEP and the red lines were runs with the pellet reward scheme enabled.



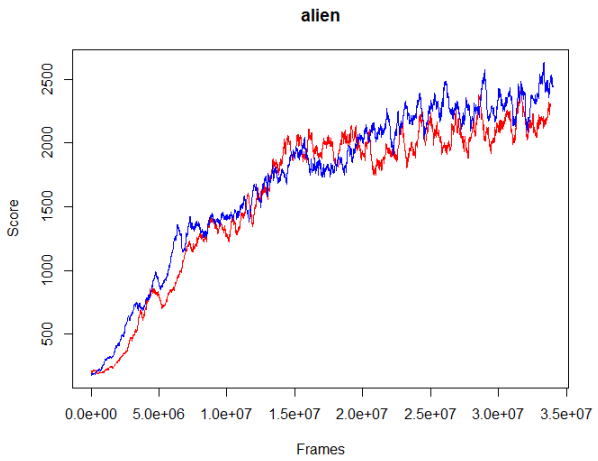
(a) Battle Zone

(b) Gravitar

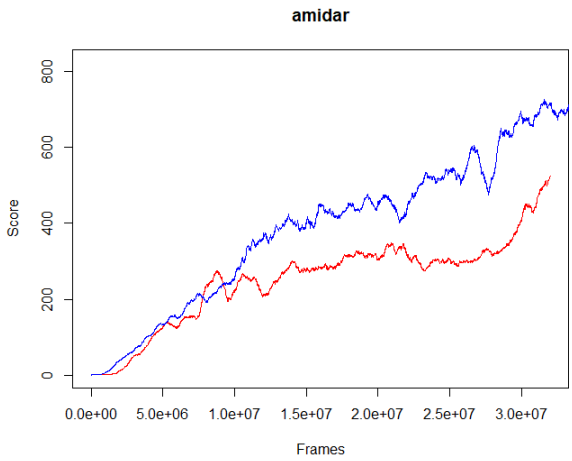
(c) Hero



(d) Qbert

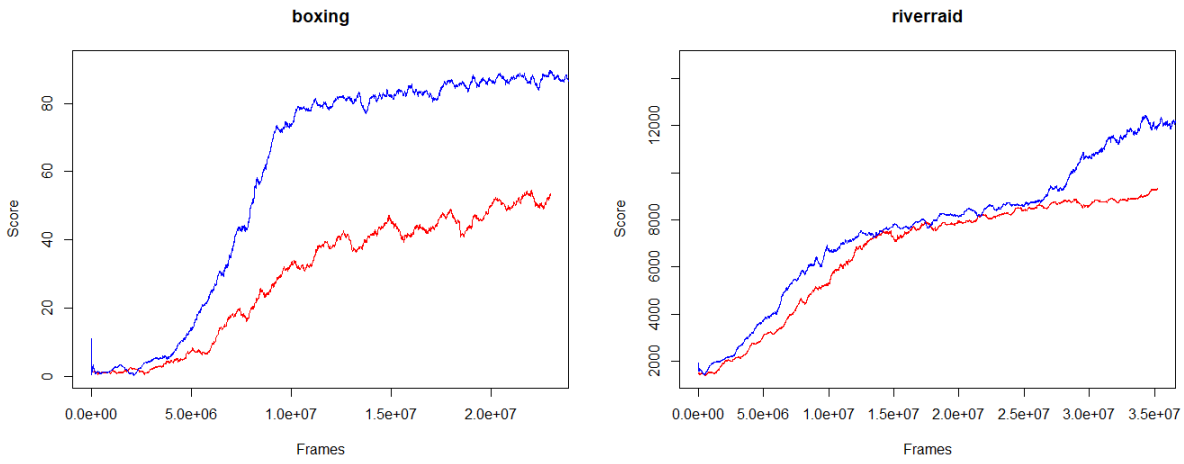


(e) Alien



(f) Amidar

Figure 5: Runs to compare the baseline of the algorithm to the pellet reward scheme



(a) Boxing **(b)** Riverraid

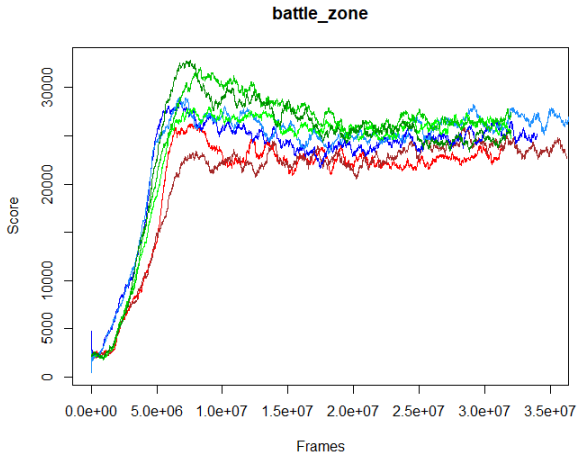
Figure 6: Runs to compare the baseline of the algorithm to the pellet reward scheme

There is one other game that got tested, but the results were discarded. That game is Atlantis. In this game, the player protects the city of Atlantis from approaching ships. Every time a ship approaches the city, a building gets destroyed. If all the buildings are destroyed, it is game over. However, the buildings regenerate after a certain amount of levels are cleared. At some point, the agent has such a good policy that it barely loses a life in this game. This means that the buildings regenerate faster than they are destroyed. Thus, the agent never reaches game over and the score keeps accumulating. After a certain number of frames, the score gets so high that it overflows. This turns the score into an extremely large negative score and completely messes up the averages. Due to this, the results of this game are left out and the game was not tested again in the future.

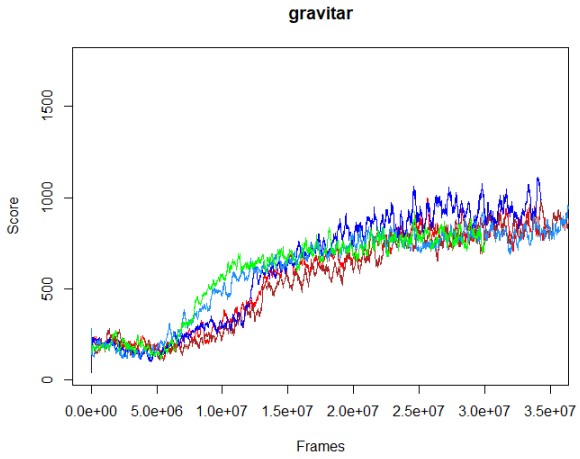
From these results it is clear that the pellet reward scheme, which is the core concept of EEP, does not always positively affect the performance of the agent. In fact, it only seems to be beneficial where exploration is hard (Hero and Qbert). But not all games with hard exploration show an improvement. Amidar also shows a decline in performance while Gravitar remains roughly equal. In other cases, such as Battle zone and Riverraid, the pellet reward scheme has a distinct negative effect. This confirmed the initial hypothesis that EEP does not have a positive effect on games where exploration is not hard.

While testing the distance constraint, the results were not really satisfactory. In Montezuma's Revenge, the agent often did not manage to reach the 2500 score. During the testing of this method two other approaches came to mind, the adjustment of the hidden pellet layer and the visit rate, which were mentioned previously. The adjustment of the hidden pellet layer was implemented first, since it seemed more like a correction rather than a adaptation. The results of this adjustment can be found in the figure 7. The green lines are runs with the new adjusted hidden layer representation.

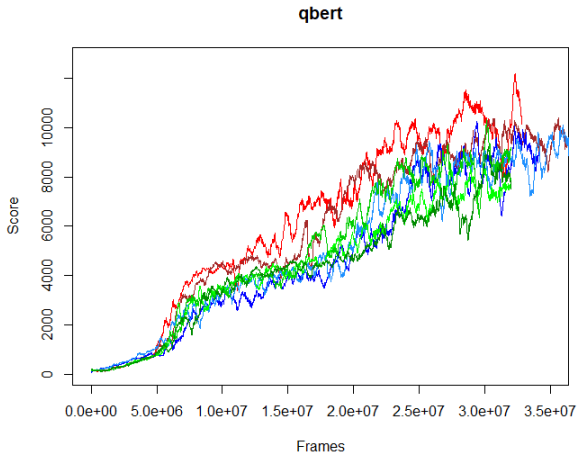
The adjustment to the hidden layer seems to overall be an improvement. Only in Qbert it can be said to perform worse than the original pellet reward scheme, but even then it still remained equal or better to the baseline. Since this change felt more as a correction and seemed to be a near overall improvement, it was made permanent and all the future tests were made with this change implemented.



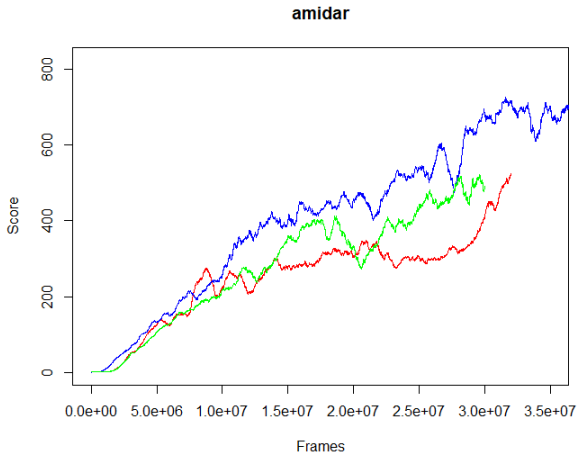
(a) Battle Zone



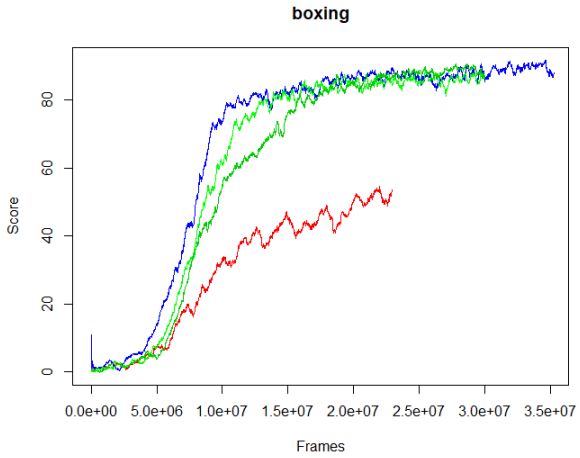
(b) Gravitar



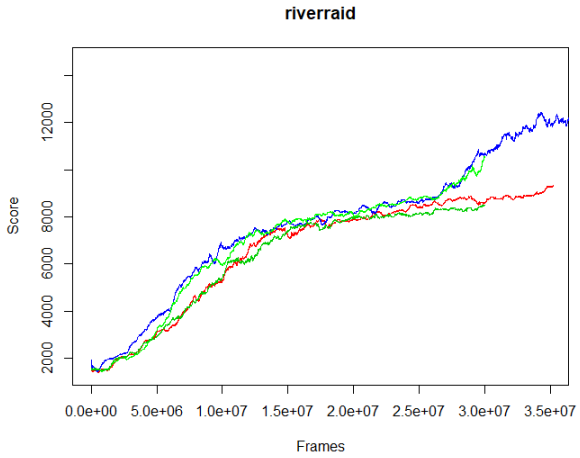
(c) Qbert



(d) Amidar



(e) Boxing



(f) Riverraid

Figure 7: Green lines are the runs with the hidden layer adjusted. Red is without the hidden layer adjustment and blue is baseline.

The hidden layer adjustment was also tested on Montezuma's Revenge. The subgoals generated from that run can be seen in the figure below. The red crosses indicate that the visit rate of the subgoal was above 0.5 at the time the subgoal was generated. A lot of these subgoals are very similar and are likely to be redundant. By removing the subgoals that have a high visit rate at creation (the ones with a red cross), some of the unnecessary subgoals already get removed. However, it should be carefully noted that early subgoals will always have a slightly higher visit rate due to the nature of the EE function. To compensate, the threshold allows higher visit rates for the earlier subgoals. From these subgoals, a visit rate threshold of 0.5 seemed to be a safe start and the program was tested with this threshold as the main constraint.

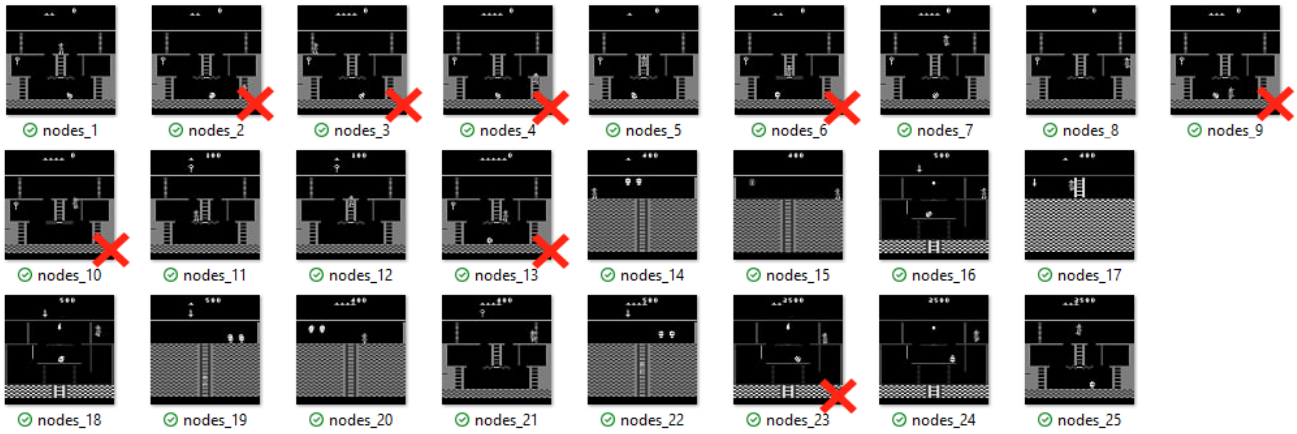


Figure 8: The subgoals generated by Montezuma's Revenge with only the hidden layer adjustment. The red crosses indicate which subgoals were visited more than 50% of all the episodes they were under examination.

The next result is from the Visit Rate test. The idea was again tested on Montezuma's Revenge. The red line is now the adjusted hidden layer and the green line is the visit rate test. As can be seen, the visit rate was slightly slower in reaching 2500, but that might be because it had not reached a score of 500 in the learning phase as the adjusted hidden layer run had. Even though only one run does not convey much information, some issues immediately became apparent. The performance does look decent enough, but the generated subgoals were concerning. The algorithm had generated 64 subgoals unlike the usual 20-24. This is due to the changed spawn location problem, which was discussed earlier. Another observable point of the subgoals is that it only generated 5 subgoals in the first room, instead of the usual 10 to 13. This could indicate that the subgoals were more meaningful, as it led to the escape of the first room in smaller numbers.

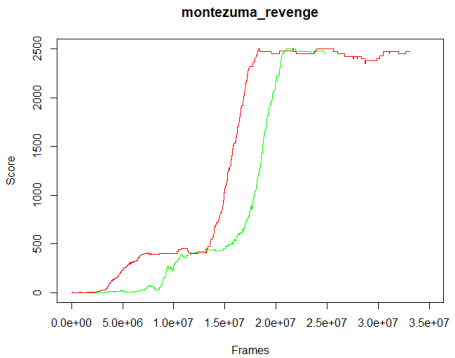


Figure 9: Montezuma's Revenge with only the visit rate constraint. Red is without the visit rate and green is with the visit rate.

To address the problem of many subgoals, the distance constraint was added to the visit rate constraint. To estimate a good relative distance threshold, the subgoals generated from the previous were analyzed. In figures 10 and 11 the subgoals of the two previous Montezuma's Revenge runs are sorted on their relative distance in their appropriate subgoal group. It creates a better image of how the distance measure ranks the subgoals. Since the distance constraint was added as a secondary constraint, the more lenient threshold was initially chosen. This is the threshold of 0.01, which contains all the subgoals within the red outline in the figures.

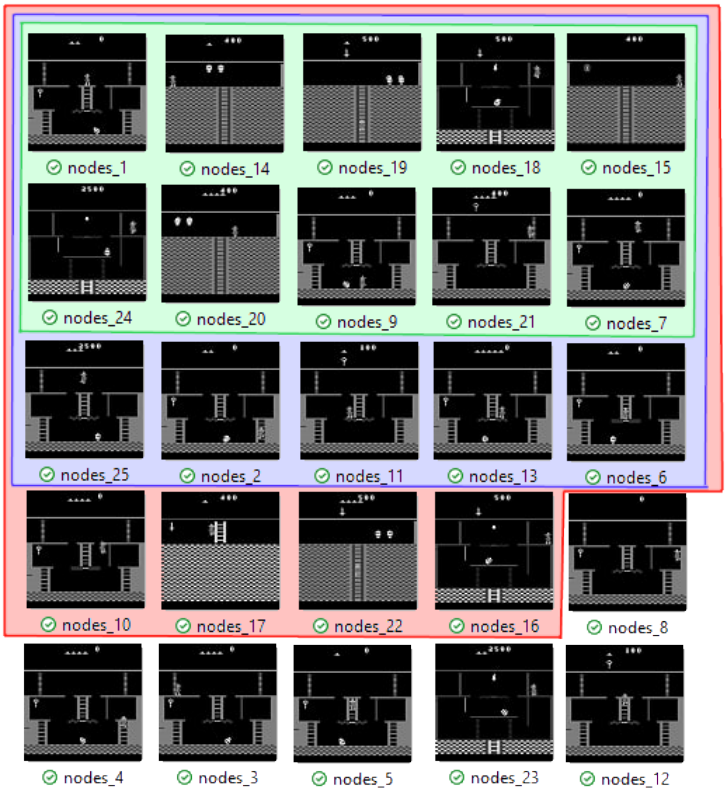


Figure 10: The subgoals generated by Montezuma's Revenge with only the the hidden layer adjustment. The subgoals in the red field all have a likelihood of less than 0.01. The blue field indicates likelihood of less than 0.001. The green field indicates a likelihood of less than 0.0001.

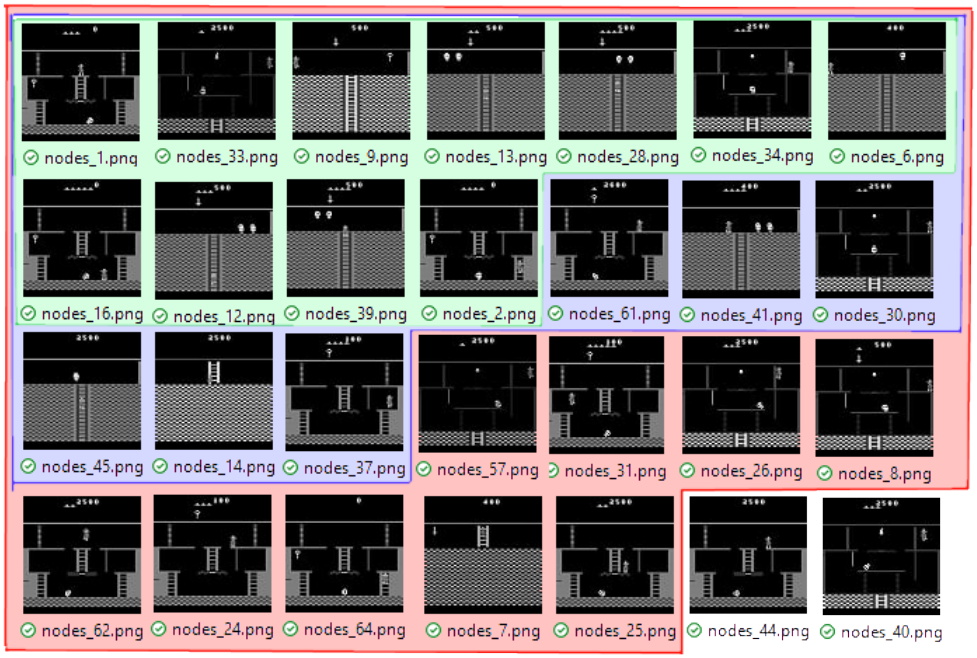
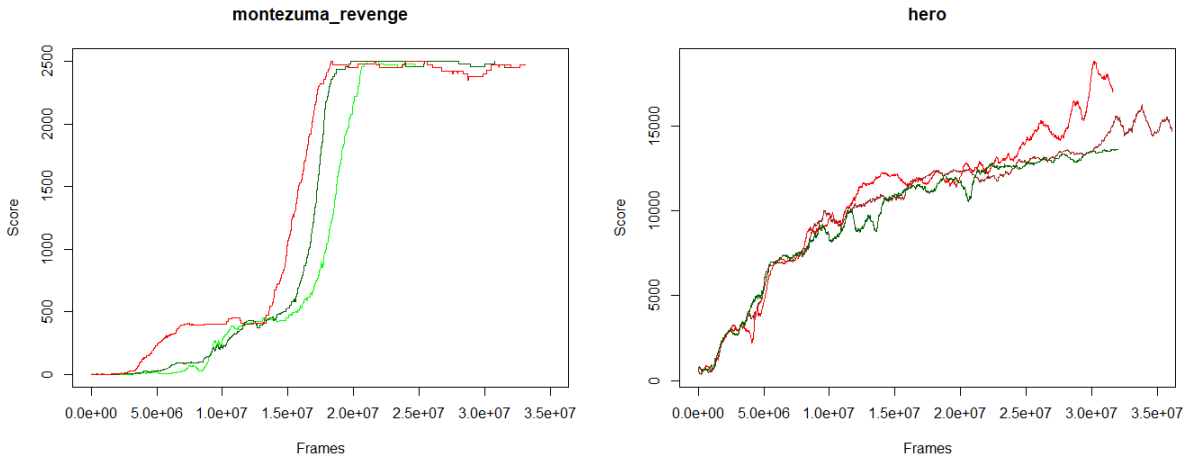


Figure 11: The subgoals generated by Montezuma's Revenge with only the visit constraint enabled. The subgoals in the red field all have a likelihood of less than 0.01. The blue field indicates likelihood of less than 0.001. The green field indicates a likelihood of less than 0.0001. Only the 28 subgoals with the lowest likelihood are shown (out of 64 subgoals total).

And thus the final method was created, with a primary constraint of the visit rate with a threshold set to 0.5 (more lenient early on) and a secondary constraint of the relative distance with a threshold of 0.01. The results of the runs with these settings can be found in figures 12a and 12b. For Montezuma's Revenge, the mixed method did better than the one purely based on the visit rate, but it was still a bit slower than the original pellet with adjusted layers. However, the when examining the subgoals, they are of much higher quality and there seem to be way less similar subgoals.

Unfortunately, the other runs got terminated early due to Amidar crashing. Amidar created 96 subgoals, causing the memory to overload. Creating this many subgoals is of course unintended and has only been an issue in Amidar and Qbert. These two games have many different paths that can be taken and every path changes the screen enough such that the algorithm creates a new subgoal. To combat this, the distance threshold was lowered to 0.001 and 0.0001. With the threshold set to 0.001, the run of Qbert was able to complete (figure 13), but Amidar was still crashing. Only at 0.0001 would Amidar finish as well.

Such a low threshold could however heavily impact the other games. Therefore, a variety of other games were tested again with this same threshold. The results of this can be seen below. The low threshold did indeed have a big impact on several games. For example, in Montezuma's Revenge, the agent did not achieve the score of 2500 in three of the five runs. Even in the runs where it did reach a score of 2500, it was much slower than the other methods.



(a) Montezuma's Revenge **(b)** Hero

Figure 12: Runs with the relative distance threshold set to 0.01 (green).

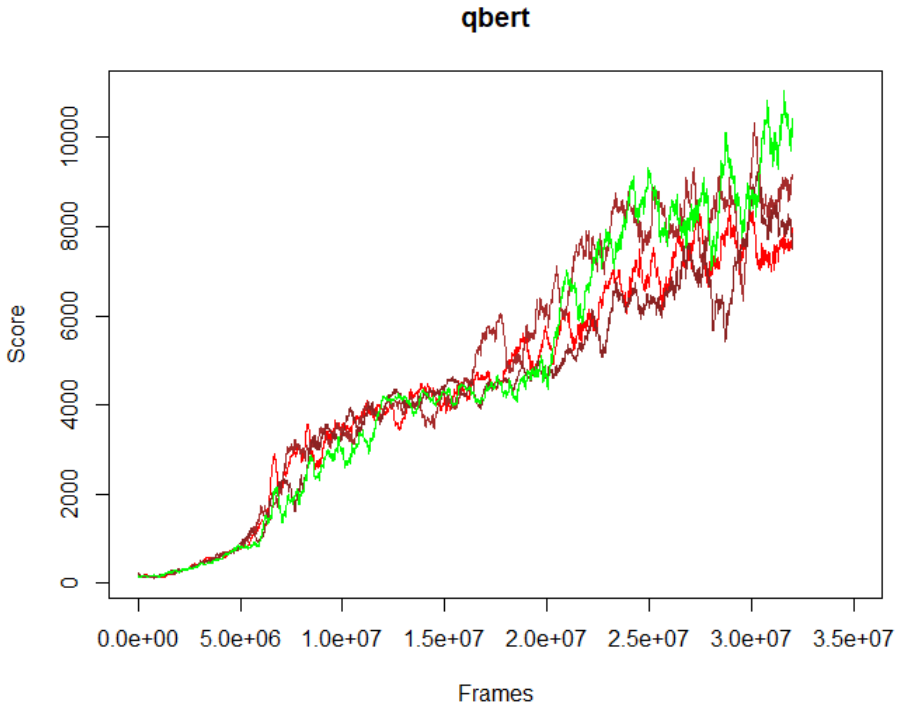
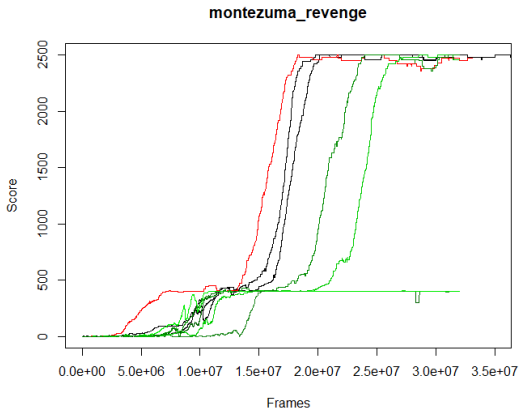
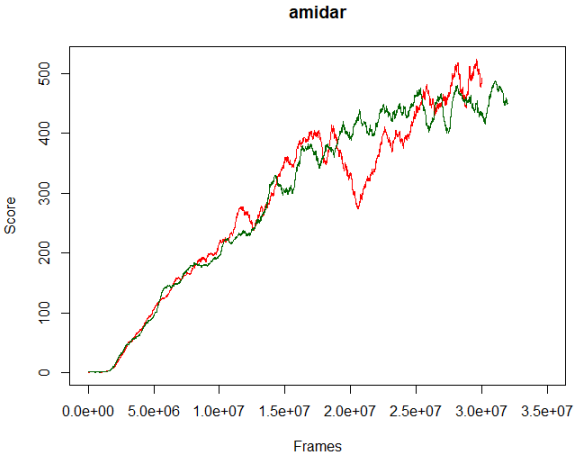


Figure 13: The run of Qbert with a relative distance threshold of 0.001. The three red/brown lines are again runs with only the adjusted hidden layer and no distance constraint.

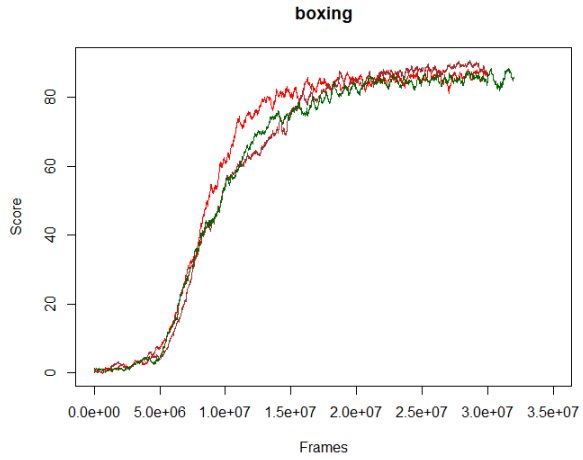


(a) Montezuma's Revenge. The black/grey lines are the runs with a relative distance threshold set to 0.01.

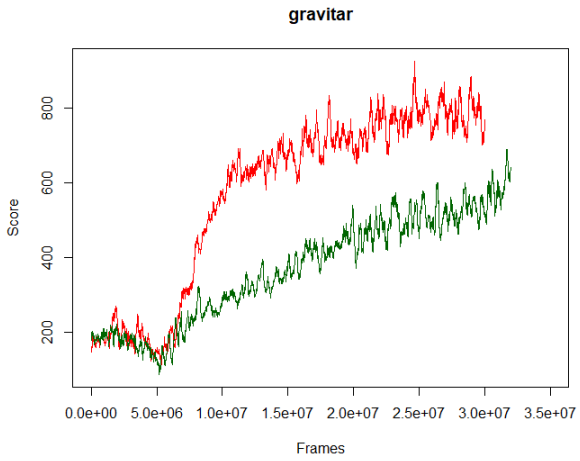


(b) Amidar

(c) Battle Zone



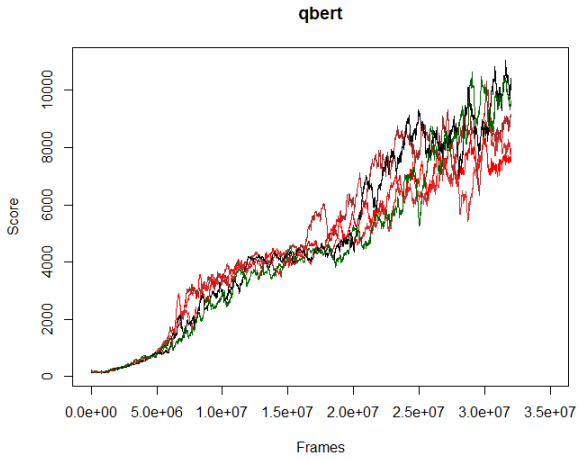
(d) Boxing



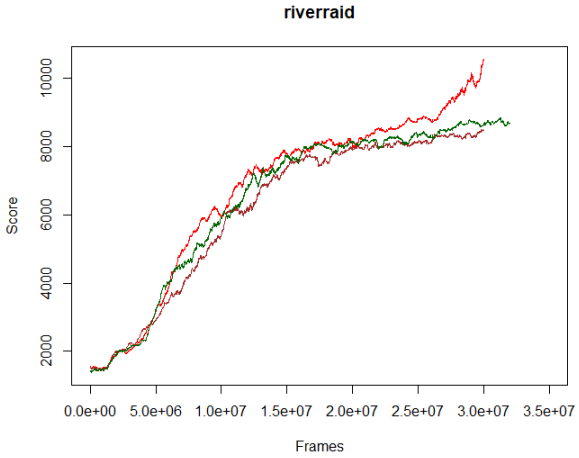
(e) Gravitar

(f) Hero

Figure 14: Comparison between runs with only the hidden layer adjustment and runs with the combined constraints and the distance threshold set to 0.0001



(a) Qbert. The grey lines represent the runs with threshold 0.001.



(b) Riverraid

Figure 15: Comparison between runs with only the hidden layer adjustment and runs with the combined constraints and the distance threshold set to 0.0001

Discussion

The adjustment to the hidden layer had the biggest impact on the performance. Especially in the game Boxing did it seem to make a major difference. This is likely due to the size of the environment. The whole game of boxing takes place inside the same small ring (see figure 16). By exploring this game, the agent creates subgoals in different locations in the ring, such as the corners. Because the diminishing value of the subgoals was not well perceived by the agent, the agent would try to collect these subgoal pellets for far too long. By focusing on collecting the pellets rather than hitting the opponent, the agent obviously scored worse. By correctly representing the value of the subgoals, the agent would focus on the game's true objective and achieve a higher score.

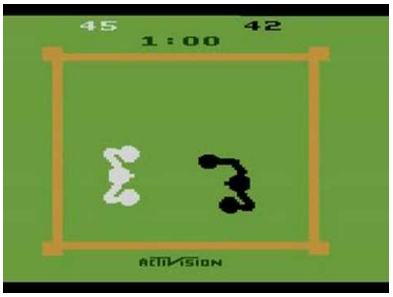


Figure 16: A screenshot of the game Boxing

The effect of the alternations on the subgoal generation is not so clear. In Montezuma's Revenge, they are slightly slower than the original method, often needing a few more frames to reach the same milestones. Without any constraints, EEP generated subgoals on a fixed schedule. This schedule created more subgoals than the adapted generation method in the early stages. The schedule generated around 12 subgoals before the second room was reached, while the constrained generation method only generated around 6 before the second room was reached (see figure 17). Regardless, the agent reached the second room in Montezuma's Revenge faster with the original subgoal generation. This could mean that having more subgoals may be beneficial, regardless of the fact if they are similar or not.

One explanation for this could be that the subgoals keep the agent alive. If the agent attempts to collect a subgoal, it is more likely to stay alive than when he is exploring. Even if the subgoals do not lead the agent in the right direction, keeping the agent alive may be more important. Another reason could be that the extra subgoals reinforce the others.

However, after the first door has been opened and the second room has been entered, the next milestone is to collect the sword and kill an enemy. The original method reaches this milestone faster, but not if you place it relative to reaching the second room. After reaching the second room, the alternated subgoal generation needs less time to kill an enemy with the sword. This is likely because it does not have as many subgoals in the first room. All the subgoals in the first room slow the agent slightly down, as the agent wants to collect the pellet rewards still. This effect was greater without the adjustment to the hidden layer, but is still present.

Thus, the extra subgoals in the first room help the agent reach the second room faster, but then slow it down slightly with future progress. It would probably increase the performance to be slightly more lenient with the first couple of subgoals. Currently, the visit rate threshold slowly descends to the steady 50%, but an anti-monotone function that descends even slower could be better. That would generate a few more subgoals, without having so many that they slow down future progress.

In other games, the method needs heavy parameter adjustments. Qbert and Amidar both have many parallel paths that the player can take. Each path changes enough on the screen such that the algorithm thinks it's a distant state. This causes too many subgoals to be generated and

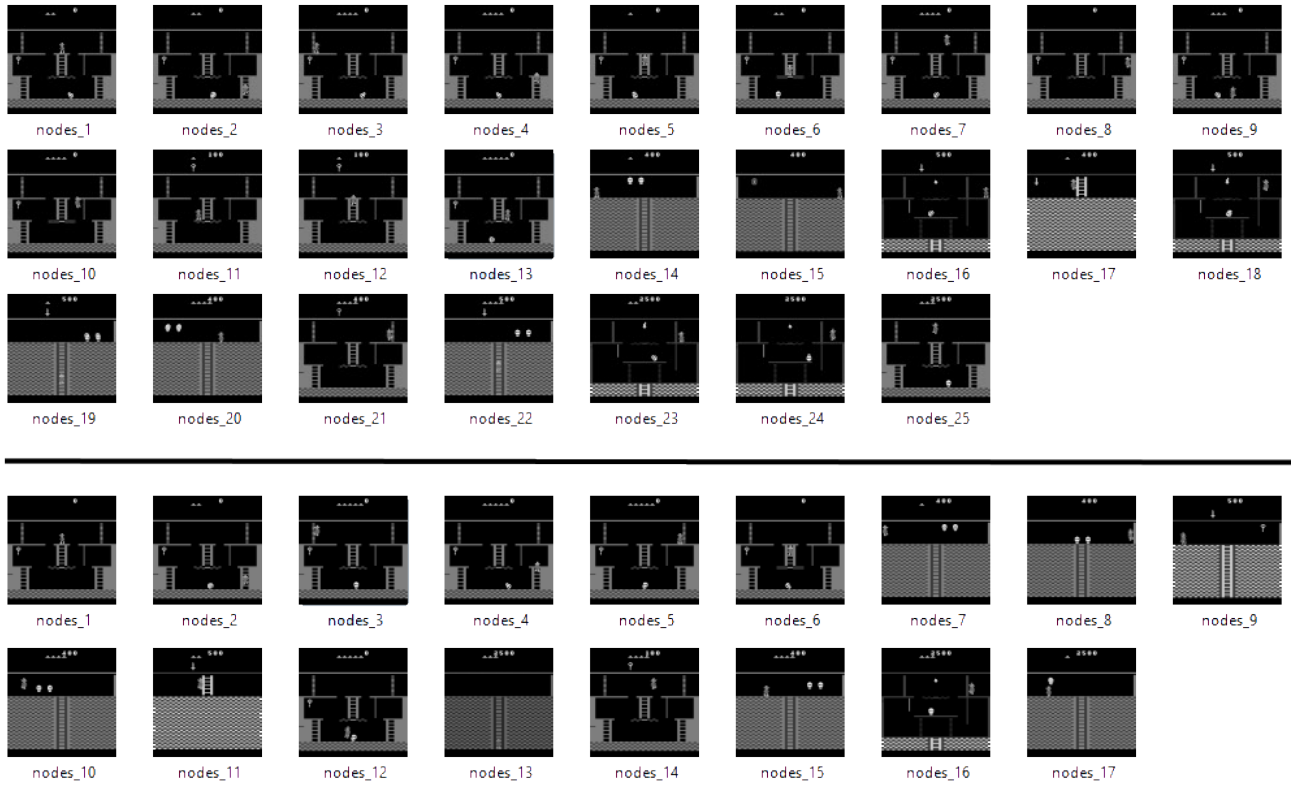


Figure 17: Top: Subgoals generated with the original method. **Bottom:** Subgoals generated with the added constraints

the memory to overload.

This issue reveals the image dependency of the EE function. Ideally, the EE function would more closely represent distance based on the actions between two states, but this is unfortunately not the case here. In this thesis the problem has been tried to fix by increasing the distance threshold, but the required threshold to make it work was too extreme. This affected other games like Montezuma's Revenge and is not an acceptable solution. The best way to handle this issue is to change to EE function to more reliably represent the distance based on action. I've recently receive notice that this change had been implemented in the latest version of the algorithm by Michael Dann.

By looking at the graphs from the latest test with a very strict distance threshold, there is another observation that can be made. The major impact of the strict threshold is that there won't be many subgoals generated. But for a lot of the games, the performance graph looks very similar to the one without the strictness of the threshold, and thus many more subgoals. This likely means that the subgoals barely have an effect in these games. These games have a dense reward space and the effect of the subgoals is not very noticeable. This is because in the sea of rewards, a few extra rewards (which diminish to 0) at certain points will not change the behavior all too much.

The relative distance threshold has been tested at different values, but another variable to test is the visit rate threshold. It's currently set to 0.5 on all test-runs, but a lower (or perhaps higher) threshold could change things drastically as well. This would also slightly reduce the problem of different spawn locations. For example, if the visit rate threshold would be decreased to 0.2, the problem would likely barely occur in Montezuma's Revenge, as the agent has 5 lives and will always visit the earlier rooms in at least one live. Unfortunately, there was not enough time to test all possibilities, and these tests are left to future iterations.

Conclusion

Reinforcement learning in an environment with a sparse reward space is a difficult problem. The EEP algorithm manages to achieve a great performance in these types of environments, while also performing excellent in other games. However, the algorithm is not without faults.

The list of subgoals generated often had some very similar subgoals. The agent also showed some peculiar behavior from time to time that indicated that something was amiss. This coupled with the fact that the algorithm was only tested on a handful of games rose the question whether EEP was really robust. Does it perform equally among all games?

After having tested more games, it is clear that EEP does not perform equally among all games. While EEP has an overall great performance, the additional elements that EEP contains (most importantly the pellet reward scheme) do not always cause an increase of performance. In the target games, namely games with a sparse reward space, the pellet reward scheme of EEP was beneficial and boosted the performance. However, in a lot of other games the pellet reward scheme had a negative impact on the performance.

One of the flaws of EEP was its attachment to decayed subgoals. In some cases, this flaw caused the agent to return to previous areas, or to collect subgoals that were no longer relevant. This flaw has been reduced with a change to the representation of the pellet rewards in the neural network. Instead of a binary value to represent whether the subgoal was visited, the value was changed to the exact reward that the corresponding pellet contains at that moment. This change caused the agent to more accurately estimate the value of collecting certain pellets and no longer left him returning to old locations and generally improved the algorithm across different environments.

The results of the previous change show that it is possible to improve EEP. It was done so by changing the hidden layer in the neural network. However, it was hypothesized that EEP could also be improved by changing the subgoal generation method. This was tried by a combination of two different constraints, a constraint on the visit rate of a candidate subgoal, and a constraint on the relative distance of the candidate subgoal. The visit rate constraint was the main constraint and observed how often a candidate would be visited if it were a subgoal. The relative distance constraint compared the distance of the subgoal candidate to the average distance of a local group of nodes. Both constraints had their advantages, but also had distinct flaws. By combining the two methods, most of the flaws were heavily reduced and this became the main approach.

Unfortunately, the constraints did not improve the overall performance. It did, however, generate a list of subgoals that is more dissimilar to each other than the list of subgoals that the original EEP generated. This was initially hypothesized to improve the performance of EEP. However, it either was too slow in generating these subgoals, or the dissimilarity of the subgoals was not as important as initially hypothesized.

Nonetheless, the performance of EEP with the added constraints was not far behind the version without the constraints. With some more experimentation, this method could potentially outperform the original method. Not all values for the parameters were extensively tested, and the recent change that the original creator added to the algorithm should also positively affect the approach of this thesis. Furthermore, there are some additional changes that could boost the performance of this method.

Future Work

One issue that has persisted through all criteria is the fact that the candidate subgoal is selected on other criteria than the creation of the subgoal. The candidate subgoal is the visited state that has the highest distance to any subgoal. However, the state with the highest distance might

not necessarily be the candidate that will pass these constraints. Another state with a lower minimum distance might be a better candidate. It's dependent on the neighboring states. It might be especially problematic if a candidate with a high distance keeps being found and rejected multiple episodes. One way to approach this issue is by having multiple candidate subgoals, preferably neighboring different subgoals. This opens the algorithm and allows it to test multiple areas at once and reducing the chance to get 'stuck' on one candidate. However, this approach would have required a lot of the code to be rewritten and was not possible given the time frame.

Another possible addition would be to make the visit rate threshold adaptive. Currently the threshold starts high, but quickly decreases to a fixed value. By making the visit rate dependent on the nearest subgoal, the issue of multiple spawn points should be reduced. For example, only create the subgoal if the candidate if visited less than half the times of the nearest subgoal. This way, if the agent only traverses a path once in his full episode (all lives), then the neighboring subgoal will also have a low visit rate and thus the candidate will be discarded. This option was shortly explored, but discontinued as the EE function sometimes would inaccurately appoint neighboring subgoals.

Bibliography

M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016. [p]

M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017. [p]

C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, and D. Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016. [p]

M. Dann, F. Zambetta, and J. Thangarajah. Real-time navigation in classical platform games via skill reuse. In *IJCAI 2017*, pages 1582–1588. AAAI Press, 2017. [p]

M. Dann, F. Zambetta, and J. Thangarajah. Deriving Subgoals Autonomously to Accelerate Learning in Sparse Reward Domains. Submitted to NIPS 2018, 2018. [p]

M. Dann, F. Zambetta, and J. Thangarajah. Integrating skills and simulation to solve complex navigation tasks in infinite mario. *IEEE Transactions on Games*, 10(1):101–106, 2018. [p]

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016. [p]

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017. [p]

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016. [p]

T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016. [p]

Y. Liang, M. C. Machado, E. Talvitie, and M. Bowling. State of the art control of atari games using shallow reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 485–493. International Foundation for Autonomous Agents and Multiagent Systems, 2016. [p]

M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *arXiv preprint arXiv:1709.06009*, 2017. [p]

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. [p]

G. Ostrovski, M. G. Bellemare, A. v. d. Oord, and R. Munos. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017. [p]

M. Roderick, C. Grimm, and S. Tellex. Deep abstract q-networks. *arXiv preprint arXiv:1710.00459*, 2017. [p]

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. [p]

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550 (7676):354, 2017. [p]

B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015. [p]

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. [p]

T. Zahavy, N. Ben-Zrihem, and S. Mannor. Graying the black box: Understanding dqns. In *International Conference on Machine Learning*, pages 1899–1908, 2016. [p]