An Exploration of Platformers

Marc WuDunn
mwudunn@berkeley.edu
University of California, Berkeley
Berkeley, California

Myron Liu
myronliu@berkeley.edu
University of California, Berkeley
Berkeley, California

ABSTRACT

Platformer games are an important subgenre of action games, which have often been used to evaluate the performance of reinforcement learning algorithms. The goal of these reinforcement learning algorithms is to construct policies that emulate a decision-making process, in order to achieve a particular objective. This is represented by having an agent maximize an expected reward. These rewards are encoded as part of the environment, and are given for performing particular actions in particular states. The sparsity of the rewards is a crucial problem in reinforcement learning, as problems that are easy when rewards are abundant can suddenly become much more difficult and less tractable when rewards are sparse. In this paper, we explore how differing levels of sparsity affects the performance of two reinforcement learning algorithms. To do, we develop a basic platformer-inspired game. In the game, the objective is for the player to reach a specified exit/goal location, while collecting as many 'coins' as possible. The 'coins' exist to guide the player towards to exit/goal, with differing levels of sparsity corresponding to differing numbers of coins on the path to the goal.

We test two reinforcement learning algorithms that we implemented in class, Double Q-Learning and the EX2 Exemplar model policy searching:

Deep double Q-Learning is an off policy reinforcement learning algorithm that avoids overestimation of the action values in noisy environments. This overestimation is caused by using the same Q function to approximate the future maximum action value. A different policy is used in double Q learning than the original Q function to approximate the future maximum action value.

EX2 Exempler model policy searching essentially employs a discriminator function that attempts to distinguish between novel states and states that have been seen before. Then, if a state has not been seen before, the agent should be given a bouns reward for performing the action at that state, so as to incentivize the agent to return to that state and further explore neighboring states (which would potentially be novel as well).
We generate multiple maps for which we test the reinforcement algorithms. Each map is configured with a specific hypothesis in mind - some maps were to explore the importance of the initial start state of the agent to its ability to find the goal and other maps were created with technical difficulty in mind. Furthermore, in each map, we added "coin" rewards for the agent was varying levels of sparsity along the path to the goal to help incentive reaching the exit. The highest level of sparsity had no coin rewards at all which often led to much random motion from the agent which rarely led to the the goal being reached.

We found that for easy maps with a high number of rewards, both double Q-learning and EX2 Exemplar model policy searching models performed well. However, on higher difficulty maps where the exit/goal location was on top of a platform, which required several specific jumps to reach, Double Q-Learning performed poorly, and while the Exemplar model was not able to reach the goal consistently it was capable of following the path of coins. Double Q-learning performed especially poorly when starting at positions which led to actions in suboptimal regions of the map which were difficult to escape from. These regions would have minimal reward and obstacles which effectively trapped the agent into making poor decisions not leading to the goal. The Exemplar model used an exploration heuristic to incentive escaping the suboptimal regions in the map as well as deterred the agent from making mistakes that led to being entrapped in those region in the first place.

Sparsity of rewards is a difficult problem for reinforcement learning to solve - in our experiments, we resorted to defining our own rewards in the environment but future exploration of model-free methods in learning the reward space are warranted. Furthermore, compression of the state space from the continuous model used to a discrete grid system may improve the estimation of the exploration bonuses.

CCS CONCEPTS

• **Reinforcement Learning**; • **Double Q-Learning** ; • **Exemplar Models**;

KEYWORDS

Reinforcement Learning, Q-Learning, Exemplar Models, neural networks

ACM Reference Format:
Marc WuDunn and Myron Liu. 2019. An Exploration of Platformers. In *Proceedings of* . ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Platformer games involve moving and jumping around obstacles to reach a particular goal. The complexity of the state space and sparseness of the rewards for reaching such a goal makes the environment interesting for testing and benchmarking reinforcement learning algorithms. In this paper, we explore two different techniques including Double Deep Q-learning (DDQN) and a heuristic counting-based exploration model (EX2) for a platformer game.

Our environment is comprised of obstacles as well as a goal block and a player block. The player block can take three different moves: jump, left, and right. Each move gives the player a velocity in a particular direction with gravity defined downwards. The main goal of the player is to reach a particular goal defined by a red exit block from a starting position. To achieve this, it needs to perform a series of actions that lead it closer to the goal as well as navigating obstacles which may be in the way. The player is rewarded by collecting "coin" along the path to the exit as well as reaching the exit scaled downwards by the amount of time taken to reach the exit.

Construction of the environment involved setting up obstacles and defining some pre-determined ideal path that the player should take to the exit. Coin rewards were defined along the ideal path with different levels of a sparsity for a particular level to specify difficulty of a particular iteration. This technique allows us to choose the sparsity of the reward space in order to avoid challenges faced by solving problems with incredibly sparse state spaces (see Montezuma's Revenge [3]).

Deep Q-learning has presented itself a potential method for solving a variety of video games. The ability to discount immediate rewards in the present for potential future rewards gives allows for more complex state spaces to be explored. Double deep Q-learning is an off policy algorithm to avoid action selection in noisy environments and allows for retraining using a replay buffer. Furthermore, Q-learning can be combined with methods such as EX-2 discrimination and an additional exploration heuristic bonus to explore sparse reward spaces and avoid being trapped in suboptimal locations in the state space.

This paper explores the usefulness of deep Q-learning in combination with optimistic exploration on a pre-defined platformer environment. We explore stages which worked with just deep Q-learning as well as cases of failure. We also adjusted reward sparsity to explore the way the dimensions of the reward space impact results. Finally, we add optimistic exploration to improve the algorithms ability to 'find' the reward of reaching the exit.

2 RELATED WORK

There has been a great deal of interest in using Reinforcement Learning in sparse environments to solve various games. One difficulty in doing so is that the agent must discover high-reward states while exploring the space, which can be particularly difficult, especially when the state space is fairly high-dimensional. One common approach is to use demonstrations to help guide the agent to high-reward areas, from which it can then improve the results [2]. This is useful because it removes the need to manually specify a reward function, which can lead to suboptimal performance.

Early works utilized Genetic Programming to evolve controllers for these environments, such as in [5], which achieved good performance on singular levels of Super Mario Bros., a platforming game. We were interested in whether we could achieve similar results from the methods described in this class.

Another interesting example of work on a platformer that has yet to be solved successfully is Montezuma's revenge [3]. The game itself for humans is a relatively straightforward task but inference on a incredibly sparse reward space along with the usage of human motifs such as keys prevent the game from easily being modelled by reinforcement learning problems. This paper looks to explore avenues into which we can better model sparse reward spaces to give rise to solutions to harder problems such as Montezuma's revenge.

Double Q-learning [6] is a popular approach in the field of Reinforcement Learning for solving various games. The rewards in these games are usually fairly dense and not temporally separated by a large amount. We leverage this method as a baseline to compare the different levels of reward sparsity in our game, and to compare against the performance of the exploration based method in sparse environments.

Several other methods for performing the exploration rely on counting the number of states an agent sees, and providing the agent with a 'bonus' reward for exploring lesser seen states. This works well for low-dimensional state spaces, but for high-dimensional and continuous state spaces, counting the number of times a state has been seen is non-trivial. Tang et al. [4] explore a hashing-based method for counting states, and achieve state-of-the-art results on Atari games. Another method is leveraging Exemplar models for exploration, as in [1]. We test the latter method for our game.

3 METHODOLOGY

3.1 Game Environment

The game was developed using PyGame, and integrated as an OpenAI gym environment.

3.1.1 Game Map. The game was modeled after a simple platformer. The agent can take any of three actions, moving right, moving left, and jumping. A game map is represented as a 20x30 set of tiles, of size (32px, 32px) each, with the exit/goal located at a specified tile. The goal of the agent is to reach the exit/goal tile, in as little time as possible, while collecting as many coins as it can.

We set up three levels to test the Reinforcement Learning methods. In the easiest map, the exit/goal was located at the end of a straight-line path from the agent's initial position, though platforms were also present in the level. In the second level, the agent starts on top of a platform, with the goal located in the same location as

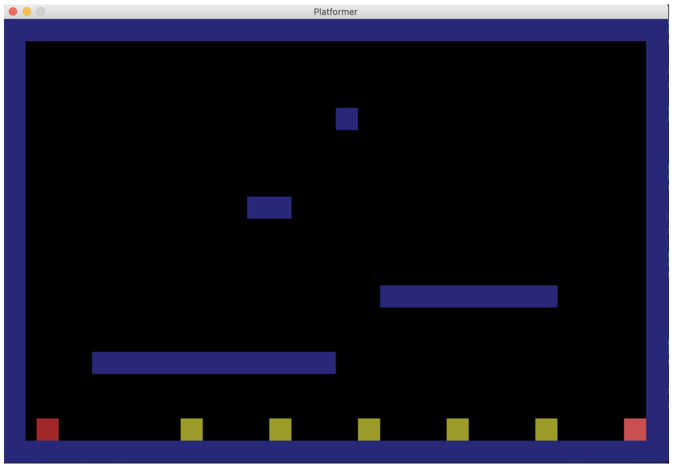

Figure 1: Medium Sparsity Environment. This is a game visualizer built in PyGame. The dark red block is the agent, the yellow blocks are 'coins' defining a sparse path of rewards, the pink block is the goal. The dark blue blocks are obstacles. Here the solution is trivial - the agent must simply hold the right button to reach the goal.

the previous level. For the final level, the goal was located on top of the highest platform. In order to reach this platform, the agent needs to jump off of several other platforms.

3.1.2 Rewards. For the reward function, we initially tested a Manhattan distance metric from the goal, such that we penalize the agent at a rate proportional to it's distance from the goal. We also penalized the agent for falling off the platform. However, our results were poor, even in the most trivial case where the goal is located in a straight line from the agent's initial position, along the opposite wall.

As a result, we added 'coins' to spread along the agent's path. Thus, the rewards for the agent were split into two categories: the reward for reaching the goal and the rewards for collecting coins. Reaching the goal conferred a reward five times larger than collecting a coin, plus an additional reward based on how quickly the agent reached it, with the agent being able to at most double the reward from reaching the goal. The coins were spread out across a path through the level, with varying levels of sparsity. In the hardest case, only reaching the goal conferred a reward, while in the simplest case, coins were present in several grid cells leading up to the goal.

3.1.3 State Space. The observation used to determine the agent's next action consists of three components. The first is the agent's current location, encoded as two float values. Similarly, the velocity of the agent is also encoded. Lastly, a boolean value for each coin is encoded, representing whether a coin had been picked up or not.

3.1.4 Reinforcement Learning. We tested two reinforcement learning methods in our project, double Q-Learning and batch policy search using Exemplar models, as described in [1]. We performed a hyperparameter search for both. We go over both methods briefly. Double Q-Learning differs from regular Q-Learning in that it utilizes one network to select an action, and another network to evaluate that action. This prevents against the overestimation present in regular Q-Learning.
In batch policy search using the Exemplar models, sample trajectories are computed for the game, and then a discriminator is trained on each state to attempt to discriminate whether the state is already present in the replay buffer, or if it is a novel state.

Hyperparameters	**Value**
Learning Rate	1
Gamma	0.99
Batch Size	32
Episode Length	200
Number of Training Steps Per Agent Update	1
Learning Frequency	1
Replay Buffer Size	10000
Learning Starts	10000

Figure 2: Double Q-learning hyperparameters

4 RESULTS

We devised three game maps on which to test the reinforcement learning algorithms. The player is given a limit of 1000 actions per map, roughly twice the amount of time required for a human player to reach the exit for any particular map, while collecting all the coins.

4.1 Double Q-Learning

The first map was tested only using double Q learning and we set the sparsity of rewards to be the main parameter. The rewards were placed as coins along a path that guided the agent to the exit - the sparsity was adjusted by first removing a quarter of the coins equidistant from each other for the medium map and having no coins at all for the high sparsity map. The double Q learning was only successful for the high sparsity reward map as it reached a peak reward of 25 consistently meaning it found the exit from iteration 100000 onwards.

The medium sparsity and high sparsity configurations resulted in mediocre rewards and the agent being trapped due to constant jumping and jittering during midair sequences seen through our visualizer. The lower than 20 rewards for both medium and high sparsity can be viewed as finding the exit but with a lot of time elapsed and thus a suboptimal path was not taken towards the reward. The sporadic nature of the average returns in the episode length indicate the high variance in success of the environment in achieving any reward at all in the sparsest configuration.

The second map configuration involved the same obstacles and the same exit but with the initial state of the agent moved onto

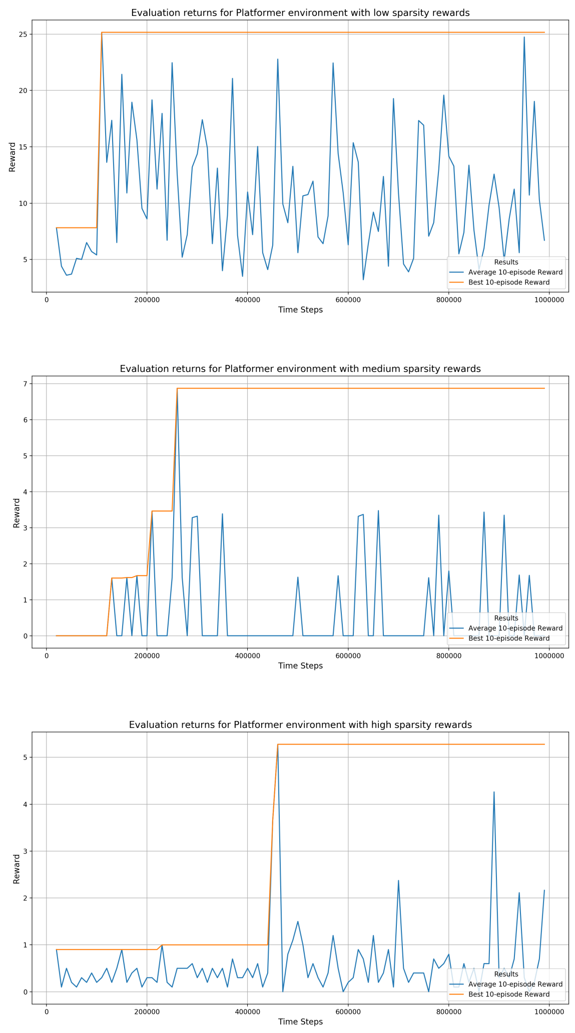

Figure 3: Results for reward different levels of sparsity of rewards on test map 1 (See Figure 2 for map configuration)

one of the platforms. We hypothesized that jumping was the primary issue with the exploration of the agent - namely jumping lead to exploration of more temporary states along a path to the exit that were difficult to revisit owing the sparseness of the states while jumping as well as their temporal infrequency in visits. This is supported by the increased rewards and frequency of rewards we receive as showing in Figure 6. The medium sparsity result is better than the previous map configuration as we observe through the visualizer that jumping could lead to finding a different way to the exit from that initial configuration whereas for the previous configuration jumping often left the agent stuck in a position where it was difficult to randomly perform an action to find the exit.

As we experimented with more maps, we noted that jumps often led to positions of suboptimal minima and thus solving more difficult problems which involved complex jumps (as shown in map

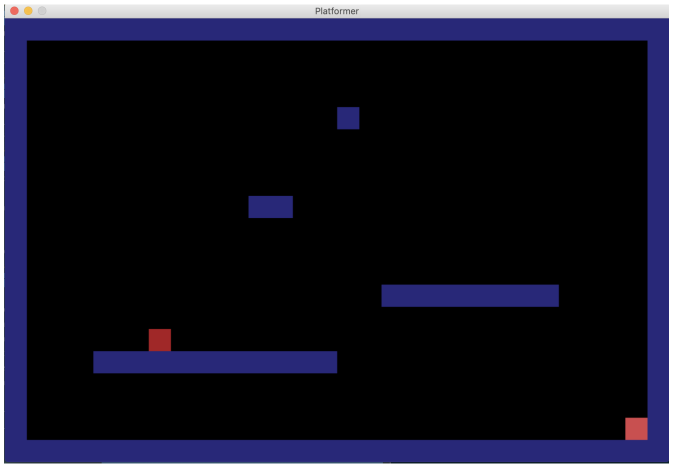

Figure 4: High Sparsity Environment. This is the second map configuration used with the same exit but different starting position for the agent.

3 see Figure 8) along with the sparsity issue lead to many suboptimal local minima which were difficult for the agent to escape from - note that the lower right hand corner was often a 'dead zone' where, if the agent missed the 2nd platform, it would become stuck until the game timed out. Thus, we implemented an EX2 reward exploration bonus to alleviate this problem of being stuck for long times in suboptimal minima. As time progresses, the rewards for being stuck in a certain region in the map decrease and the agent is incentived to leave that region in search of better rewards.

4.2 EX2: Exemplar Models

We tested the exemplar model on the easy map, with a medium sparsity of coins. The exemplar model did not seem to have much difficulty in solving this map, and within a hundred iterations was able to consistently complete it within the time limit, though not necessarily with an optimal score. The results are shown in Figure 6.

We then tested the policy search with exemplar models using the most difficult map, where the exit was located at the top of the highest platform. The hyperparameters are listed in table 10. Coins were spread out generously along a path to the goal, as shown in Figure 7, though they did not follow the quickest path to the goal (which was placed so that successful exploration might discover it). A comparison between the results for the policy search with exploration and the policy search without exploration is shown in figure 7. The peaks observed are times in which the agent solved the problem with the height correlating to how quickly the agent reached the goal. Overall, the exploration-based model performed slightly better than the non-exploration based one, though neither were able to reach the goal consistently.

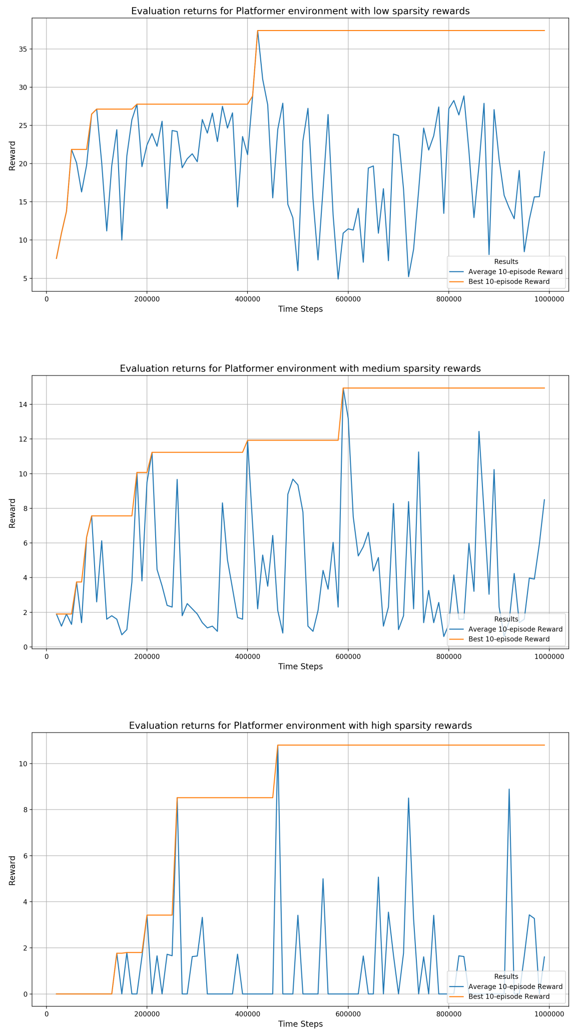

Figure 5: Results for reward different levels of sparsity of rewards on map 2 (See Figure 5 for map configuration)

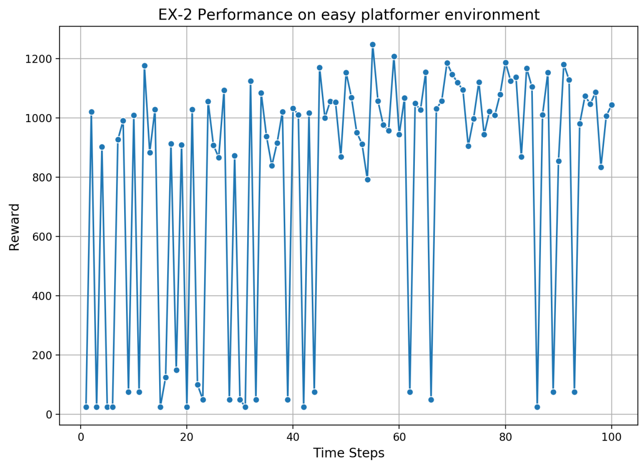

Figure 6: Rewards per 10 episodes for EX-2 exploration on easy environment from Figure 1 with high sparsity (no coin rewards).

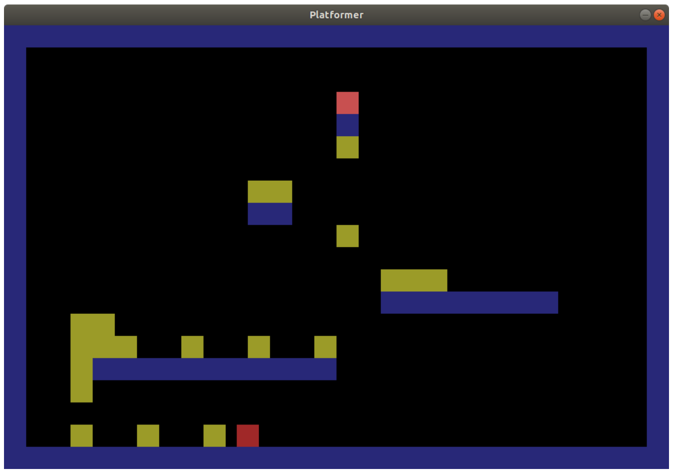

Figure 7: Environment with high difficulty, where the goal (light-red) rests on top of a platform high-up. Starting location is the cell with the red square.

5 DISCUSSION AND LIMITATIONS

We initially set out to explore the problem of transfer learning, by giving an agent a random map and passing in the map itself (and the agent's position and velocity) as the state, hoping that it would be able to solve these maps in general. However, we quickly realized that the agent struggled even with a single, fairly straightforward map. Getting decent results on the maps we presented required a significant amount of hyperparameter tuning, even without solving the even harder problem of transfering the learning to new domains. If we had more time, we would have liked to test out more reinforcement learning algorithms to see if the results improved, and whether the agent could consistently reach the goal even in the harder settings of our experiments.

We were also interested in the idea of partial observability, where the agent only perceives a section of the map at one time, and would need to infer the location of the goal (and any of the coins) in order to solve the problem. Finally, if we could get a reinforcement learning algorithm to consistently solve our problem, we could also try to randomly place the goal location at each iteration and encode it as part of our state, and see if the agent could learn to search the game space and reach the goal.

Sparsity seems to be a difficult problem to solve. We chose to define our own reward functions and apply the idea of subgoals that built towards the main goal of finding the exit in our model,

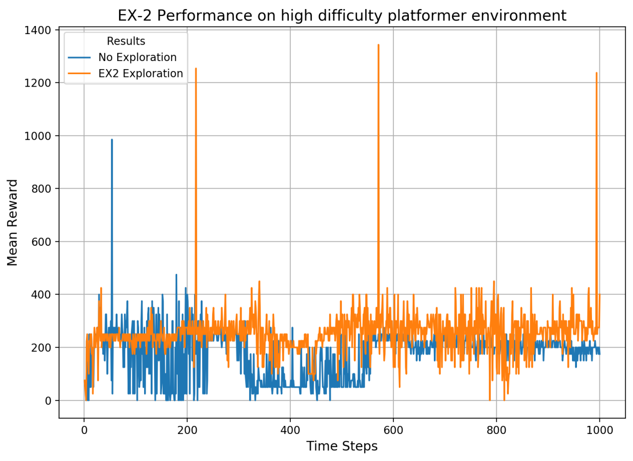

Figure 8: Average reward plot comparing results for with EX-2 exploration and without EX-2 exploration for the high difficulty environment

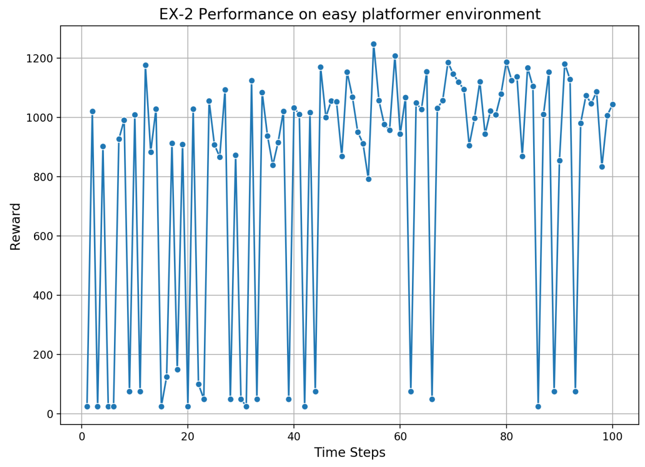

Figure 9: Rewards per 10 episodes for EX-2 exploration on easy environment from Figure 1 with high sparsity (no coin rewards).

Hyperparameters	Value
Learning Rate	5e-4
Bonus Coefficient	1
Dense Network Layers	2
Episodes	1000

Figure 10: Exemplar Model hyperparameters

as we found that more desirable than training a model to imitate a player and then fine-tune the results. However, this did not always work well especially if the agent found itself trapped in a location without many rewards defined. For the most part, when it got stuck it struggled immensely to return to the path and reach the goal. Moreover, defining coins along a path does not directly send the agent to the goal. Thus our choice of reward function in the sparse environment proved to be too disconnected from the goal of reaching the exit itself. Further experimentation with the reward function is necessary especially in handling edge cases where the agent falls off an intended platform or unsuccessfully makes a leap onto a small platform.

6 CONTRIBUTIONS

Marc WuDunn: Developed the game and game logic, implemented the game rendering, integrated EX2 model with the game, implemented the rendering for the environment, performed hyperparameter tuning for the EX2 model.

Myron Liu: Integrated the game as an OpenAI gym environment, developed and tested various reward functions, integrated the double Q-Learning, developed and tested on maps for Q-Learning, performed hyperparameter tuning for double Q-Learning model.

REFERENCES

[1] Justin Fu, John D. Co-Reyes, and Sergey Levine. 2017. EX2: Exploration with Exemplar Models for Deep Reinforcement Learning. *CoRR* abs/1703.01260 (2017). arXiv:1703.01260 http://arxiv.org/abs/1703.01260

[2] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. 2018. Overcoming Exploration in Reinforcement Learning with Demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 6292–6299. https://doi.org/10.1109/ICRA.2018.8463162

[3] Tim Salimans and Richard Chen. 2018. Learning Montezuma's Revenge from a Single Demonstration. *CoRR* abs/1812.03381 (2018). arXiv:1812.03381 http://arxiv.org/abs/1812.03381

[4] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. 2017. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 2753–2762. http://papers.nips.cc/paper/6868-exploration-a-study-of-count-based-exploration-for-deep-reinforcement-learning.pdf

[5] Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jürgen Schmidhuber. 2009. Super Mario Evolution. In *Proceedings of the 5th International Conference on Computational Intelligence and Games (CIG'09)*. IEEE Press, Piscataway, NJ, USA, 156–161. http://dl.acm.org/citation.cfm?id=1719293.1719326

[6] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.