Reinforcement Learning for the Generation of Dungeon-Style Levels

MAT501: Applied Mathematics and Artificial Intelligence

Kayla Wallace
2101726

1 INTRODUCTION

The aim of this project is to create an AI that can generate dungeon-style levels using reinforcement learning. The AI will change the states of the tiles in a given 4x4 grid, adding states that insert enemies and health pickups into the level, and these state changes are rendered in real-time. Applications such as this can aid in the development of games in numerous ways. A fully trained model could generate hundreds of levels, and these could be used to train in-game AI that is required to function in unseen levels/terrain – e.g., user created maps. This model has not learned to the fullest extent that it could, getting stuck at a plateau or local maximum, achieving a positive reward but not the maximum reward.

To create this application, Unity's ML-Agents package was used to allow for the rendering of the generated level maps in real time. At the beginning of an episode, 16 observations are collected, observing the state of each tile in the 4x4 grid. During the episode, the agent will traverse through each tile in the grid, making a decision at each. Using a discrete action branch, the agent can make one of the following decisions at each tile: change the state to empty, to contain an enemy, to contain a health pick-up, or to skip over this tile. The performance of this reinforcement learning agent has been tested using both Proximal Policy Optimisation (PPO) and Soft-Actor Critic (SAC) trainers, with the implementation of reward shaping and without. PPO provided much more stable agent behaviour and a generally higher cumulative reward than SAC did for both sets of tests.

2 BACKGROUND

Reinforcement learning is a type of machine learning in which an AI agent interacts with its surrounding environment, taking actions in order to maximise its reward. What this means is, reinforcement learning rewards the AI for 'good' choices. In the case of this project, the AI would be rewarded for generating a map that contains tile states that meet the target range. Typically, an agent will interact with an environment, and at each timestep (of length T) will: observe the current state of the environment (s), take an action (a), and receive a reward based on the transition between this state and the prior state. It uses a stochastic policy (π) to choose the best actions to take, and tries to find the best policy that will maximise its cumulative reward collected throughout the timestep (T). A formula for this is shown below in Figure 1.

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{t=0}^{T} r_t \right]$$

Figure 1: Reinforcement learning algorithm (Tang and Haarnoja 2017)

In recent years, reinforcement learning has been trained to become proficient in complex games.

One of the most famous examples of this is the 'AlphaGo' AI learning to play the board game 'Go'. 'AlphaGo' competed against Mr Lee Sedol, a winner of 18 world titles, and winning 4 games to 1 in March 2016 (*AlphaGo: The story so far* n.d.).

However, even more recently, reinforcement learning has been used by AAA game companies. Ubisoft have been using reinforcement learning to improve the navigation of NPCs (non-player characters). Instead of using a navigation mesh for the navigation of NPCs, as is commonplace in the games industry at this time, they implemented a reinforcement learning algorithm to teach the agents to get from a starting position to a goal position (Alonso et al. 2020).

The method of generating levels in this report could be useful for training AI that needs to function in player created – therefore unseen – maps. Training AI to complete these reinforcement learning generated levels can encourage challenging but not impossible level design. It can also be used to generalize the agent. This means that with the ability to train something like an enemy agent in lots of different level layouts, it becomes more able to handle varying environments, and therefore can provide better and more realistic actions in game. And with the rising complexity and size of games, automated/assisted creation of game assets is increasingly important. Game designers could set out a range of 'ideal' levels – e.g. for different difficulties – and this AI could generate hundreds of levels for each once fully trained.

ML-Agents provides an interface within the Unity Editor to implement reinforcement learning. It contains 5 main components: the learning environment, the low-level Python API, an external communicator, Python trainers, and a Gym wrapper. The learning environment is what holds the Unity scene and all its components. The low-level Python API is to interface with and manipulate the learning environment, and communicates with Unity through the external communicator. The Python trainers hold the machine learning algorithms in which to train the agents with. Finally, the Gym wrapper implements a way for developers to interact with the simulated environments via OpenAI Gym. (Unity-Technologies 2021)

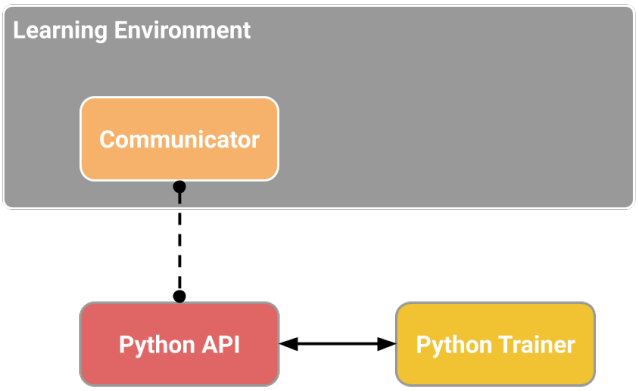


Figure 2: Simplified diagram of ML-Agents structure (Unity-Technologies 2021)

ML-Agents provides an implemention of both Proximal Policy Optimisation (PPO) and Soft-Actor Critic (SAC) reinforcement learning algorithms. PPO has been assigned as the default due to its characteristic of being more general purpose and stable than other reinforcement learning algorithms. PPO is an algorithm created to be simpler to implement than the likes of Trust Region Policy Optimisation (TRPO) and Q-Learning, while keeping up with the efficiency and performance of each (Schulman et al. 2017). The formula for PPO can be seen below in figure 3.

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

Figure 3: Proximal Policy Optimisation (PPO) formula (Schulman 2020)

Where θ is the policy parameter, $\hat{E}_t$ is the empirical expectation over timesteps, r_t is the ratio of the probability under new and old policies respectively, $\hat{A}_t$ is the estimated advantage at time t, and ϵ is a hyperparameter, the value of which is generally 0.1 or 0.2 (ibid.).

In contrast to PPO, SAC is an off-policy algorithm, meaning it can use past experiences to learn.

It can randomly draw a past experience from its replay buffer at any time, making it much more sample efficient than PPO. SAC is also a maximum entropy algorithm and the formula for this can be seen below in Figure 4.

$$\pi^*_{\text{MaxEnt}} = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{t=0}^{T} r_t + \mathcal{H}(\pi(\cdot|\mathbf{s}_t)) \right]$$

Figure 4: Maximum entropy formula (Tang and Haarnoja 2017)

The formula above in Figure 4 gives the stochastic policy, with the left half of the equation (inside the square brackets) being the reward, and the right half of the equation being the entropy. Maximum entropy stochastic policies allow for better exploration and are able to learn alternative ways of completing the task at hand (Fragkiadaki n.d.).

3 DESIGN

It was decided early on that Unity's ML-Agents would be used to implement this application. It was chosen due to it's ability to create a sophisticated system in a shorter amount of time than if all the implementation was done from scratch. Due to the time limit of the project, and the desire for the levels to be generated in real-time, Unity and ML-Agents seemed like an ideal choice. The use of this package does however come at a slight cost, in that most of the mathematical implementation is done in the ML-Agents implementation, and the trainer has been implemented already. Since the mathematics is a core feature of the module, ample research was done to ensure that the developer knew exactly how everything worked within ML-Agents to explain within this report, demonstrating the necessary knowledge.

Unity's ML-Agents includes trainers for Proximal Policy Optimisation, Soft-Actor Critic (SAC), Multi-Agent Posthumous Credit Assignment (POCA), and imitation. PPO was chosen as it has been shown to be more general purpose, easier to tune, and stable than other reinforcement learning algorithms (Unity-Technologies 2021).

In the initial stages of the project, the AI was intended to generate the ground layout in addition to altering the states of these ground tiles to include other items, such as enemies or health pickups. As more thought was put into the input for the AI and the observations it would need to make, it was necessary to simplify design. The input for the AI then became the states of 16 tiles placed in an empty 4x4 grid.

For keeping track of the progress of the algorithm while testing, Tensorboard was chosen. Tensorboard results can be extracted from the ML-agents run using 'tensorboard –logdir results' in the command line and can be viewed on localhost port 6006. This allowed for the generation of several useful graphs based on the learning results, such as: cumulative reward, episode length, policy loss, value loss, curiosity forward loss, and curiosity inverse loss. Another way of keeping track of the algorithm progress during training was by setting an update to appear in the command line every 20,000 steps. This meant steady progress updates were received. Using Unity's Debug.Log feature, the reward at each step was printing to the console as a further method of tracking the algorithms progress.

For this project, discrete action branches are used as opposed to continuous branches. Continuous branches allow for several actions at once, whereas discrete only allow for one at a time. This is more suited to this project as it was only necessary for the agent to change the state of one tile at a time while iterating through the grid. Additionally, only a single branch was needed to store all the actions the agent can take, since they are all changing states. Initially, two branches were going to be used. One holding actions regarding changing states, and the other holding actions allowing the agent to move in any direction providing there was a tile there. However, this caused issues with the agents ability to change states and physics, so

it was decided to stick to a traversal method for the course of this assignment.

4 IMPLEMENTATION

When using ML-Agents for implementation, there are several built in methods that can be used. In this project, the following have been implemented:

- OnEpisodeBegin(): Method run at the beginning of each episode
- CollectObservations(): Called at all steps in which a decision is requested by the agent.
- OnActionReceived(): Called every time the agent receives an action to take. Additionally, this is where rewards are set.

The application also makes use of SetReward() to set the agent's reward and EndEpisode() to end the current episode.

Starting with the OnEpisodeBegin() method, the agent's position is set to that of the first tile in order for it to begin traversing through the entire grid. Two arrays are present in the Agent class, one storing the floor tile game objects, and the other storing the state of each tile in the grid.

In the CollectObservations() method, the tile states array is iterated through and the value at each index of the array is added as an observation. However, AddObservation() method can only take in integers, booleans, vectors, and quaternions. To combat this, a function was written to convert the state of each tile into an integer. This function takes in a tile game object and outputs the corresponding integer value for it's state.

In the OnActionReceived() method, the action buffer branch is defined. Discrete actions are used and the branch size is 4, corresponding to the number of actions the agent can choose to make. This method is also where the traversal of the grid and state changes occur, as well as the reward shaping and end of episode condition checking. This method first iterates over the tiles in the grid, breaking out of this loop when the agent gets to the last tile. The current position of the agent is calculated by comparing it's x and z position values with that of the tiles. Then, a conditional statement has been implemented with each condition being an action that can be executed by the agent. In each of these conditions, the state of the current tile is changed according to the action received and the next position in the grid is calculated, with the agent finally moving onto the next tile. When the agent is at the last tile, a for-loop iterates through each of the tiles, counting how many of each state appears in the new grid. These values are then used in the reward shaping.

The reward shaping is implemented in a way such that +1 is the maximum reward the agent can receive, and -1 is the minimum. The agent is awarded +0.3 reward if the number of each tile state within the grid falls between the constant minimum and maximum target values that are set for each type. A reward of +0.1 is set if the number of each tile state falls one outside of the target range (target max. + 1, target min. - 1). Finally, the agent gets a negative reward of -0.5 if the number of each tile state falls outwith the target range by a value greater than 1. Since there are 3 positive reward conditions that will result in a +0.3 reward, reaching a total of +0.9, the final condition states that if this reward is achieved, set the reward value to 1. After this conditional statement, the SetReward() method is called with the newly calculated reward value and the episode is ended.

5 TESTING

To test the learning algorithm, it was run several times under different circumstances. Each set of results will be shown in this section and will be discussed in section 6. The hyperparameters for each run can be found in Appendix A, and the mean reward and standard deviation at every 20,000 timestep interval can be found in Appendix B.

The application was run twice using the PPO trainer, and twice using the SAC trainer so that a comparison can be made from the results of both. The hyperparameters of each 'config' file are set as evenly as they can be, so to test under fair conditions. The first two runs are a comparison of the

two trainers when the reward shaping discussed in section 4 is implemented.

The last two runs are a comparison of the two trainers when no reward shaping is in place. This means that the agent will gain a reward only if the states of the tiles in the grid fall within the correct range, and will accumulate a negative reward when the states values fall outside this range.

Each of these tests are run over the course of 500,000 steps. In each figure shown below, the red line represents the PPO trained agent and the blue line represents the SAC trained agent with reward shaping in place. The results are as follows:

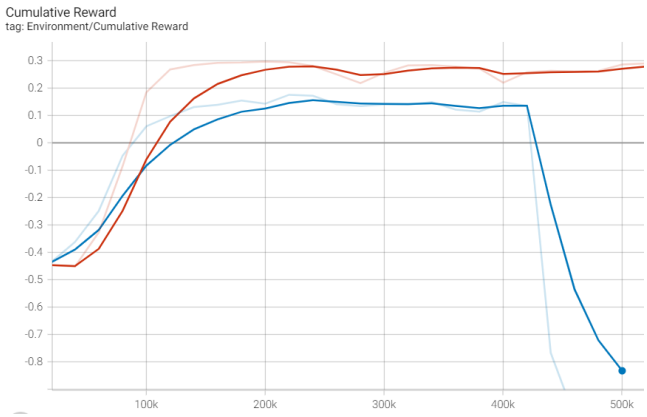


Figure 5: Cumulative reward of PPO vs SAC training with reward shaping

The graph above (figure 5) represents the cumulative reward gained by each agent.

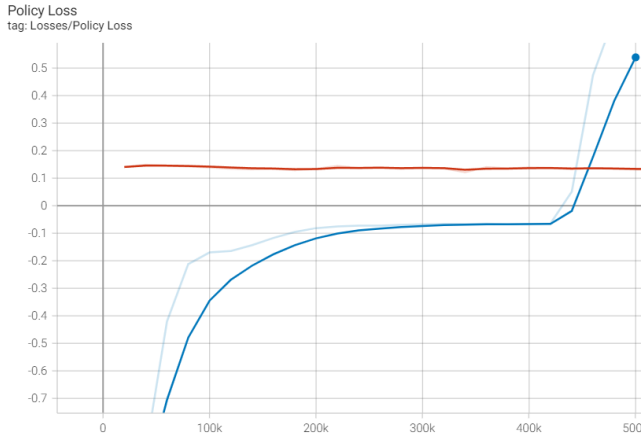


Figure 6: Policy loss of PPO vs SAC training with reward shaping

The graph above (figure 6) represents the mean magnitude of policy loss, which corresponds with how much the policy is changing (AurelianTactics 2018).

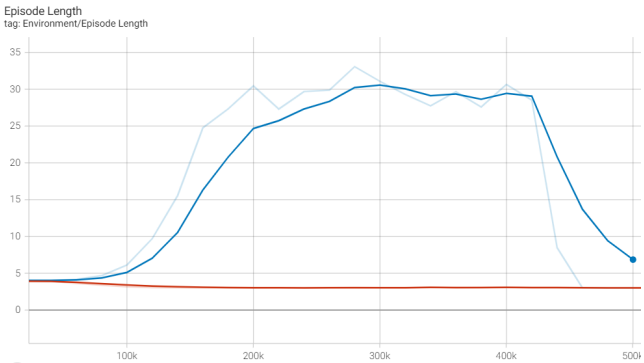


Figure 7: Episode length of PPO vs SAC training with reward shaping

The graph above (figure 6) represents the episode length over the course of the run.

Each of these tests are run over the course of 500,000 steps. In each figure shown below, the pink line represents the PPO trained agent and the blue line represents the SAC trained agent with **no** reward shaping in place. The results are as follows:

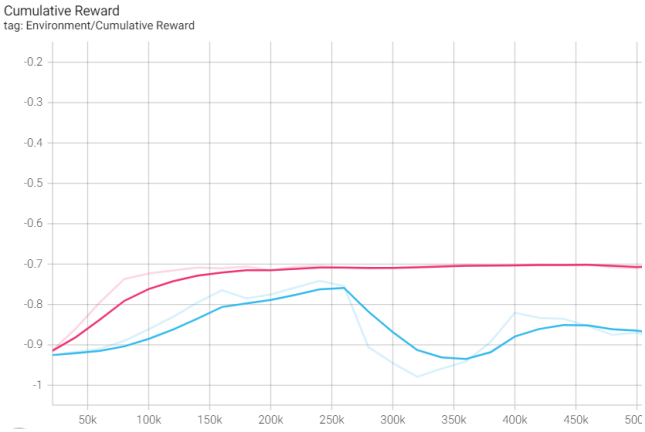


Figure 8: Cumulative reward of PPO vs SAC training with no reward shaping

The graph above (figure 8) represents the cumulative reward gained by each agent.

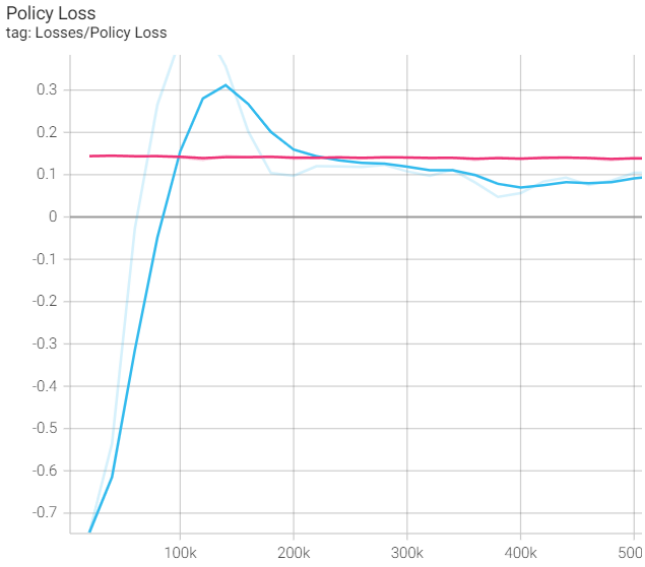


Figure 9: Policy loss of PPO vs SAC training with no reward shaping

The graph above (figure 9) represents the mean magnitude of policy loss, which corresponds with how much the policy is changing (AurelianTactics 2018).

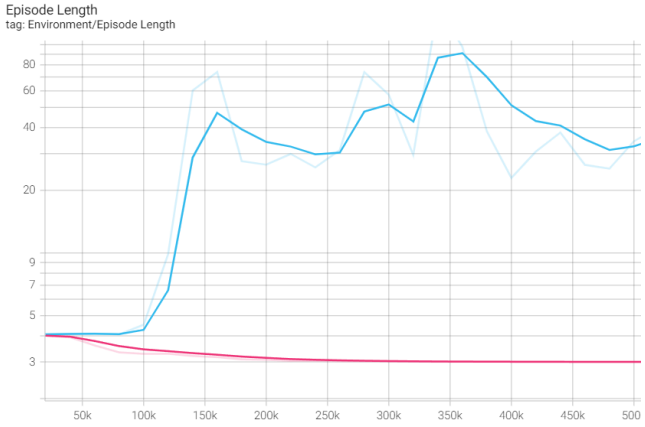


Figure 10: Episode length of PPO vs SAC training with no reward shaping

The graph above (figure 9) represents the episode length over the course of the run. The y-axis had to be displayed in a logarithmic scale due to the size of the peak in this graph.

6 EVALUATION

In the tests that include reward shaping, the AI does begin to learn to generate the 'ideal' level. However, the data clearly shows that it gets stuck in a local maximum and then will usually plateau, gaining a reward of roughly +0.3. A local maximum is a state in which the AI is gaining a reward better than the neighbouring states, but is not reaching the maximum reward (or global maximum), and a plateau is when the current state is the same as it's neighbouring states. (*Introduction to hill climbing: Artificial intelligence* 2021). However, the tests conducted with no reward shaping result in very little to no learning at all. To combat this, a better reward shaping algorithm should be implemented in future iterations of this project. Additionally, there are methods that could be implemented to help the algorithm overcome the local maximum.

6.1 With Reward Shaping

The PPO trainer provided a more steady increase in the mean reward gained by the agent, eventually getting stuck in a plateau, while the SAC trainer proved to provide more varied results with a vast

decrease in the reward towards the end of the run. The PPO trained agent also gained a higher mean reward on average than the SAC trained agent did.

The episode length of the SAC trained agent was far longer and less stable than the PPO trained agent. This shows how a SAC trainer is better suited for a 'heavier' training environment.

The policy loss of both trained agents were not ideal, as in a successful run, this value should decrease over time (AurelianTactics 2018). However, the PPO trained agent proved to have a much steadier policy loss than the SAC trained agent in which the policy loss increased significantly as the mean reward decreased.

6.2 With No Reward Shaping

Similarly to the above evaluation, the PPO trainer provided a much steadier increase in cumulative reward in the non-reward shaping implementation than the SAC trainer did. The PPO agent hit a plateau quite early into the training run, whereas the SAC agent continued to fluctuate.

The episode length of the SAC trainer greatly outweighed that of the PPO trainer. The PPO trainer provided a steady episode length, while the SAC length values are extremely varied. A low of 4.09 and a high of 91.09 was recorded.

Again, the policy loss of both agents were not ideal values. The behaviour of the SAC trainer is slightly more favourable. It increases by a large amount in the beginning but eventually starts to decrease relatively steadily, whereas the PPO behaviour is mostly constant.

7 CONCLUSION

The aim of this project was to create an AI that can generate dungeon-style levels using reinforcement learning. Implemented using Unity and ML-Agents, the maps are rendered in real-time as the agent makes actions. It takes in 16 observations, the state of each tile in a 4x4 grid, and makes actions to change these states. The states can be changed to empty, containing an enemy, or containing a health pick-up. The fourth action that can be taken is to skip the current tile without changing it's state. These state changes are rendered in real-time using the Unity engine. The project was implemented making use of Unity's ML-Agents package to make this possible during the time given for the project implementation. During each episode of a run, the agent traverses over the entire grid, making a decision for which action to carry out at each one. Generating levels like this could assist in the development of game AI (such as enemies or NPCs) that need to navigate unseen terrain/level maps, e.g. user created maps. The agent in this report was trained using both PPO and SAC trainers, and a comparison is made to both when trained with reward shaping and without. In general, PPO provided more stable agent behaviour and a higher cumulative reward value, as the research had suggested.

7.1 Future Work

In future, the reward shaping would need to be altered and the hyperparameters of the trainer tweaked to overcome the local maximum that the agent currently appears to get stuck in. Thereafter, it would be optimal to expand this project to include adversarial reinforcement learning. By implementing a solver agent for the generated levels, the reward for the level generator agent would be calculated based on how well the solver was able to complete the level. This would ideally result in the generator creating levels that provide a good challenge but are not impossible. The reward calculations could also be weighted depending on the difficulty required.

REFERENCES

Alonso, Eloi et al. (2020). "Deep Reinforcement Learning for Navigation in AAA Video Games". In: *CoRR* abs/2011.04764. arXiv: 2011.04764. URL: https://arxiv.org/abs/2011.04764.

AlphaGo: The story so far (n.d.). URL: https://deepmind.com/research/case-studies/alphago-the-story-so-far.

AurelianTactics (Dec. 2018). *Understanding PPO plots in tensorboard.* URL: https://medium.com/aureliantactics/understanding-ppo-plots-in-tensorboard-cbc3199b9ba2.

Fragkiadaki, Katerina (n.d.). *Maximum Entropy Reinforcement Learning*. URL: https://www.andrew.cmu.edu/course/10-403/slides/S19maxentRL.pdf.

Introduction to hill climbing: Artificial intelligence (Nov. 2021). URL: https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/.

Schulman, John (Sept. 2020). *Proximal policy optimization*. URL: https://openai.com/blog/openai-baselines-ppo/.

Schulman, John et al. (2017). "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347. arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

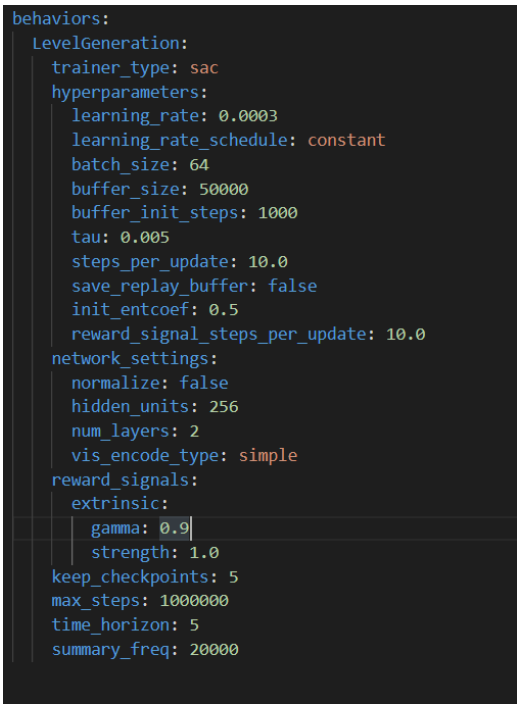
Tang, Haoran and Tuomas Haarnoja (Oct. 2017). *Learning diverse skills via maximum entropy deep reinforcement learning*. URL: https://bair.berkeley.edu/blog/2017/10/06/soft-q-learning/.

Unity-Technologies (Apr. 2021). *ml-agents/ML-Agents-Overview.md at main · Unity-Technologies/ml-agents*. URL: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/ML-Agents - Overview . md # curiosity - for - sparse - reward - environments.

8 APPENDICES

A HYPERPARAMETERS FOR PPO AND SAC TRAINING

A.1 SAC Hyperparameters



A.2 PPO Hyperparameters

```yaml
behaviors:
  LevelGeneration:
    trainer_type: ppo
    hyperparameters:
      batch_size: 32
      buffer_size: 8000
      learning_rate: 3e-4
      beta: 5e-3
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 0.7
        network_settings:
          hidden_units: 128
          num_layers: 2
      curiosity:
        strength: 0.9
        gamma: 0.99
        network_settings:
          hidden_units: 128
          num_layers: 2
        learning_rate: 3e-4
    keep_checkpoints: 5
    max_steps: 1000000
    time_horizon: 1000
    summary_freq: 20000
```

B RESULTS OF TEST RUNS

B.1 PPO Trainer with Reward Shaping

```
[INFO] LevelGeneration. Step: 20000. Time Elapsed: 123.998 s. Mean Reward: -0.446. Std of Reward: 0.403. Training.
[INFO] LevelGeneration. Step: 40000. Time Elapsed: 224.424 s. Mean Reward: -0.452. Std of Reward: 0.427. Training.
[INFO] LevelGeneration. Step: 60000. Time Elapsed: 339.300 s. Mean Reward: -0.327. Std of Reward: 0.450. Training.
[INFO] LevelGeneration. Step: 80000. Time Elapsed: 439.135 s. Mean Reward: -0.085. Std of Reward: 0.428. Training.
[INFO] LevelGeneration. Step: 100000. Time Elapsed: 554.404 s. Mean Reward: 0.184. Std of Reward: 0.264. Training.
[INFO] LevelGeneration. Step: 120000. Time Elapsed: 656.224 s. Mean Reward: 0.268. Std of Reward: 0.143. Training.
[INFO] LevelGeneration. Step: 140000. Time Elapsed: 776.597 s. Mean Reward: 0.284. Std of Reward: 0.102. Training.
[INFO] LevelGeneration. Step: 160000. Time Elapsed: 883.016 s. Mean Reward: 0.293. Std of Reward: 0.069. Training.
[INFO] LevelGeneration. Step: 180000. Time Elapsed: 1004.526 s. Mean Reward: 0.293. Std of Reward: 0.077. Training.
[INFO] LevelGeneration. Step: 200000. Time Elapsed: 1111.562 s. Mean Reward: 0.297. Std of Reward: 0.046. Training.
[INFO] LevelGeneration. Step: 220000. Time Elapsed: 1235.075 s. Mean Reward: 0.295. Std of Reward: 0.060. Training.
[INFO] LevelGeneration. Step: 240000. Time Elapsed: 1341.117 s. Mean Reward: 0.281. Std of Reward: 0.107. Training.
[INFO] LevelGeneration. Step: 260000. Time Elapsed: 1459.950 s. Mean Reward: 0.250. Std of Reward: 0.174. Training.
[INFO] LevelGeneration. Step: 280000. Time Elapsed: 1564.313 s. Mean Reward: 0.218. Std of Reward: 0.226. Training.
[INFO] LevelGeneration. Step: 300000. Time Elapsed: 1685.083 s. Mean Reward: 0.256. Std of Reward: 0.160. Training.
[INFO] LevelGeneration. Step: 320000. Time Elapsed: 1792.499 s. Mean Reward: 0.283. Std of Reward: 0.103. Training.
[INFO] LevelGeneration. Step: 340000. Time Elapsed: 1913.782 s. Mean Reward: 0.284. Std of Reward: 0.097. Training.
[INFO] LevelGeneration. Step: 360000. Time Elapsed: 2018.114 s. Mean Reward: 0.278. Std of Reward: 0.113. Training.
[INFO] LevelGeneration. Step: 380000. Time Elapsed: 2135.812 s. Mean Reward: 0.271. Std of Reward: 0.130. Training.
[INFO] LevelGeneration. Step: 400000. Time Elapsed: 2239.020 s. Mean Reward: 0.220. Std of Reward: 0.210. Training.
[INFO] LevelGeneration. Step: 420000. Time Elapsed: 2357.106 s. Mean Reward: 0.258. Std of Reward: 0.160. Training.
[INFO] LevelGeneration. Step: 440000. Time Elapsed: 2461.927 s. Mean Reward: 0.263. Std of Reward: 0.154. Training.
[INFO] LevelGeneration. Step: 460000. Time Elapsed: 2595.126 s. Mean Reward: 0.261. Std of Reward: 0.157. Training.
[INFO] LevelGeneration. Step: 480000. Time Elapsed: 2711.175 s. Mean Reward: 0.262. Std of Reward: 0.150. Training.
[INFO] LevelGeneration. Step: 500000. Time Elapsed: 2832.480 s. Mean Reward: 0.286. Std of Reward: 0.091. Training.
```

B.2 SAC Trainer with Reward Shaping

```
[INFO] LevelGeneration. Step: 20000. Time Elapsed: 109.669 s. Mean Reward: -0.436. Std of Reward: 0.412. Training.
[INFO] LevelGeneration. Step: 40000. Time Elapsed: 209.182 s. Mean Reward: -0.362. Std of Reward: 0.419. Training.
[INFO] LevelGeneration. Step: 60000. Time Elapsed: 308.366 s. Mean Reward: -0.249. Std of Reward: 0.414. Training.
[INFO] LevelGeneration. Step: 80000. Time Elapsed: 407.852 s. Mean Reward: -0.047. Std of Reward: 0.355. Training.
[INFO] LevelGeneration. Step: 100000. Time Elapsed: 516.217 s. Mean Reward: 0.061. Std of Reward: 0.301. Training.
[INFO] LevelGeneration. Step: 120000. Time Elapsed: 616.027 s. Mean Reward: 0.098. Std of Reward: 0.285. Training.
[INFO] LevelGeneration. Step: 140000. Time Elapsed: 710.398 s. Mean Reward: 0.131. Std of Reward: 0.270. Training.
[INFO] LevelGeneration. Step: 160000. Time Elapsed: 801.802 s. Mean Reward: 0.139. Std of Reward: 0.266. Training.
[INFO] LevelGeneration. Step: 180000. Time Elapsed: 895.290 s. Mean Reward: 0.155. Std of Reward: 0.257. Training.
[INFO] LevelGeneration. Step: 200000. Time Elapsed: 995.329 s. Mean Reward: 0.143. Std of Reward: 0.265. Training.
[INFO] LevelGeneration. Step: 220000. Time Elapsed: 1092.602 s. Mean Reward: 0.176. Std of Reward: 0.243. Training.
[INFO] LevelGeneration. Step: 240000. Time Elapsed: 1186.209 s. Mean Reward: 0.172. Std of Reward: 0.246. Training.
[INFO] LevelGeneration. Step: 260000. Time Elapsed: 1278.630 s. Mean Reward: 0.141. Std of Reward: 0.266. Training.
[INFO] LevelGeneration. Step: 280000. Time Elapsed: 1370.027 s. Mean Reward: 0.134. Std of Reward: 0.269. Training.
[INFO] LevelGeneration. Step: 300000. Time Elapsed: 1466.162 s. Mean Reward: 0.141. Std of Reward: 0.265. Training.
[INFO] LevelGeneration. Step: 320000. Time Elapsed: 1560.058 s. Mean Reward: 0.140. Std of Reward: 0.266. Training.
[INFO] LevelGeneration. Step: 340000. Time Elapsed: 1653.199 s. Mean Reward: 0.149. Std of Reward: 0.261. Training.
[INFO] LevelGeneration. Step: 360000. Time Elapsed: 1746.391 s. Mean Reward: 0.121. Std of Reward: 0.275. Training.
[INFO] LevelGeneration. Step: 380000. Time Elapsed: 1841.387 s. Mean Reward: 0.114. Std of Reward: 0.282. Training.
[INFO] LevelGeneration. Step: 400000. Time Elapsed: 1934.619 s. Mean Reward: 0.149. Std of Reward: 0.260. Training.
[INFO] LevelGeneration. Step: 420000. Time Elapsed: 2027.674 s. Mean Reward: 0.135. Std of Reward: 0.268. Training.
[INFO] LevelGeneration. Step: 440000. Time Elapsed: 2125.840 s. Mean Reward: -0.767. Std of Reward: 0.473. Training.
[INFO] LevelGeneration. Step: 460000. Time Elapsed: 2234.261 s. Mean Reward: -1.000. Std of Reward: 0.000. Training.
[INFO] LevelGeneration. Step: 480000. Time Elapsed: 2342.996 s. Mean Reward: -1.000. Std of Reward: 0.000. Training.
[INFO] LevelGeneration. Step: 500000. Time Elapsed: 2450.671 s. Mean Reward: -1.000. Std of Reward: 0.000. Training.
```

B.3 PPO Trainer with No Reward Shaping

```
[INFO] LevelGeneration. Step: 20000. Time Elapsed: 112.921 s. Mean Reward: -0.917. Std of Reward: 0.134. Training.
[INFO] LevelGeneration. Step: 40000. Time Elapsed: 215.328 s. Mean Reward: -0.861. Std of Reward: 0.150. Training.
[INFO] LevelGeneration. Step: 60000. Time Elapsed: 332.790 s. Mean Reward: -0.795. Std of Reward: 0.139. Training.
[INFO] LevelGeneration. Step: 80000. Time Elapsed: 439.220 s. Mean Reward: -0.737. Std of Reward: 0.098. Training.
[INFO] LevelGeneration. Step: 100000. Time Elapsed: 558.939 s. Mean Reward: -0.724. Std of Reward: 0.081. Training.
[INFO] LevelGeneration. Step: 120000. Time Elapsed: 662.897 s. Mean Reward: -0.716. Std of Reward: 0.067. Training.
[INFO] LevelGeneration. Step: 140000. Time Elapsed: 784.720 s. Mean Reward: -0.708. Std of Reward: 0.050. Training.
[INFO] LevelGeneration. Step: 160000. Time Elapsed: 890.718 s. Mean Reward: -0.710. Std of Reward: 0.055. Training.
[INFO] LevelGeneration. Step: 180000. Time Elapsed: 1013.213 s. Mean Reward: -0.706. Std of Reward: 0.043. Training.
[INFO] LevelGeneration. Step: 200000. Time Elapsed: 1117.796 s. Mean Reward: -0.715. Std of Reward: 0.066. Training.
[INFO] LevelGeneration. Step: 220000. Time Elapsed: 1241.989 s. Mean Reward: -0.707. Std of Reward: 0.044. Training.
[INFO] LevelGeneration. Step: 240000. Time Elapsed: 1350.367 s. Mean Reward: -0.704. Std of Reward: 0.035. Training.
[INFO] LevelGeneration. Step: 260000. Time Elapsed: 1473.606 s. Mean Reward: -0.709. Std of Reward: 0.051. Training.
[INFO] LevelGeneration. Step: 280000. Time Elapsed: 1581.512 s. Mean Reward: -0.711. Std of Reward: 0.056. Training.
[INFO] LevelGeneration. Step: 300000. Time Elapsed: 1704.801 s. Mean Reward: -0.709. Std of Reward: 0.051. Training.
[INFO] LevelGeneration. Step: 320000. Time Elapsed: 1814.931 s. Mean Reward: -0.706. Std of Reward: 0.042. Training.
[INFO] LevelGeneration. Step: 340000. Time Elapsed: 1939.063 s. Mean Reward: -0.703. Std of Reward: 0.031. Training.
[INFO] LevelGeneration. Step: 360000. Time Elapsed: 2054.105 s. Mean Reward: -0.701. Std of Reward: 0.020. Training.
[INFO] LevelGeneration. Step: 380000. Time Elapsed: 2178.382 s. Mean Reward: -0.703. Std of Reward: 0.029. Training.
[INFO] LevelGeneration. Step: 400000. Time Elapsed: 2287.094 s. Mean Reward: -0.703. Std of Reward: 0.029. Training.
[INFO] LevelGeneration. Step: 420000. Time Elapsed: 2410.508 s. Mean Reward: -0.701. Std of Reward: 0.017. Training.
[INFO] LevelGeneration. Step: 440000. Time Elapsed: 2517.301 s. Mean Reward: -0.702. Std of Reward: 0.022. Training.
[INFO] LevelGeneration. Step: 460000. Time Elapsed: 2638.504 s. Mean Reward: -0.701. Std of Reward: 0.017. Training.
[INFO] LevelGeneration. Step: 480000. Time Elapsed: 2746.454 s. Mean Reward: -0.709. Std of Reward: 0.050. Training.
[INFO] LevelGeneration. Step: 500000. Time Elapsed: 2872.324 s. Mean Reward: -0.711. Std of Reward: 0.056. Training.
```

B.4 SAC Trainer with No Reward Shaping

```
[INFO] LevelGeneration. Step: 20000. Time Elapsed: 110.320 s. Mean Reward: -0.926. Std of Reward: 0.130. Training.
[INFO] LevelGeneration. Step: 40000. Time Elapsed: 209.109 s. Mean Reward: -0.917. Std of Reward: 0.134. Training.
[INFO] LevelGeneration. Step: 60000. Time Elapsed: 309.083 s. Mean Reward: -0.910. Std of Reward: 0.138. Training.
[INFO] LevelGeneration. Step: 80000. Time Elapsed: 408.572 s. Mean Reward: -0.890. Std of Reward: 0.145. Training.
[INFO] LevelGeneration. Step: 100000. Time Elapsed: 509.064 s. Mean Reward: -0.861. Std of Reward: 0.150. Training.
[INFO] LevelGeneration. Step: 120000. Time Elapsed: 604.875 s. Mean Reward: -0.830. Std of Reward: 0.149. Training.
[INFO] LevelGeneration. Step: 140000. Time Elapsed: 693.567 s. Mean Reward: -0.795. Std of Reward: 0.140. Training.
[INFO] LevelGeneration. Step: 160000. Time Elapsed: 792.568 s. Mean Reward: -0.765. Std of Reward: 0.123. Training.
[INFO] LevelGeneration. Step: 180000. Time Elapsed: 892.965 s. Mean Reward: -0.785. Std of Reward: 0.135. Training.
[INFO] LevelGeneration. Step: 200000. Time Elapsed: 988.809 s. Mean Reward: -0.775. Std of Reward: 0.130. Training.
[INFO] LevelGeneration. Step: 220000. Time Elapsed: 1080.344 s. Mean Reward: -0.758. Std of Reward: 0.119. Training.
[INFO] LevelGeneration. Step: 240000. Time Elapsed: 1172.526 s. Mean Reward: -0.741. Std of Reward: 0.103. Training.
[INFO] LevelGeneration. Step: 260000. Time Elapsed: 1267.629 s. Mean Reward: -0.754. Std of Reward: 0.115. Training.
[INFO] LevelGeneration. Step: 280000. Time Elapsed: 1362.532 s. Mean Reward: -0.906. Std of Reward: 0.139. Training.
[INFO] LevelGeneration. Step: 300000. Time Elapsed: 1452.340 s. Mean Reward: -0.945. Std of Reward: 0.116. Training.
[INFO] LevelGeneration. Step: 320000. Time Elapsed: 1547.258 s. Mean Reward: -0.979. Std of Reward: 0.076. Training.
[INFO] LevelGeneration. Step: 340000. Time Elapsed: 1641.891 s. Mean Reward: -0.959. Std of Reward: 0.103. Training.
[INFO] LevelGeneration. Step: 360000. Time Elapsed: 1729.651 s. Mean Reward: -0.941. Std of Reward: 0.119. Training.
[INFO] LevelGeneration. Step: 380000. Time Elapsed: 1824.261 s. Mean Reward: -0.893. Std of Reward: 0.144. Training.
[INFO] LevelGeneration. Step: 400000. Time Elapsed: 1917.534 s. Mean Reward: -0.820. Std of Reward: 0.147. Training.
[INFO] LevelGeneration. Step: 420000. Time Elapsed: 2008.878 s. Mean Reward: -0.833. Std of Reward: 0.149. Training.
[INFO] LevelGeneration. Step: 440000. Time Elapsed: 2099.843 s. Mean Reward: -0.836. Std of Reward: 0.149. Training.
[INFO] LevelGeneration. Step: 460000. Time Elapsed: 2190.694 s. Mean Reward: -0.853. Std of Reward: 0.150. Training.
[INFO] LevelGeneration. Step: 480000. Time Elapsed: 2281.661 s. Mean Reward: -0.875. Std of Reward: 0.148. Training.
[INFO] LevelGeneration. Step: 500000. Time Elapsed: 2372.378 s. Mean Reward: -0.870. Std of Reward: 0.149. Training.
```