Jump, Run and Learn: Reinforcement Learning Take on SuperMario Bros

CSxx46 Project Report

Parashara Ramesh	E1216292	A0285647M
Sriram Srikanth	E1132261	A0276528R
Xian He	E1101724	A0268425X
Grace Ngu Sook Ern	E1101580	A0268281X

Abstract

A comprehensive comparison of behavioural and quantitative performances of DQN, PPO and A2C on the classic "Super Mario Bros" video game. PPO demonstrated highest pass count over 1000 plays, A2C is the most risk-seeking achieving highest coin count over 1000 plays while DQN is the most conservative. Our proposed reward function encourages the agents to achieve higher game scores without hurting the agents' ability to pass the level. All 3 models demonstrate to some extent of generalizability on unseen levels but not enough to complete the unseen levels.

1 Introduction

In this project, we attempt to comprehensively analyze the behavioural and quantitative performances of different categories of Reinforcement Learning algorithms (Value based, Policy Based, Actor Critic based) on the classic "Super Mario Bros" video game. Additionally, we modify the reward function (to be discussed in the next section) and endeavor to correlate each agent's performance with the inherent traits of its respective algorithm, providing deeper insights into the requirements for success in overcoming a moderately complex game like Mario.

2 SuperMario Bros Environment & Scope

Considering that the original game has 32 levels (8 worlds x 4 stages) each with varying complexities in terrain, enemies & rewards, we simplified & standardized all our RL agents by ensuring that the:

- **Environment** used, corresponded to the blocky version #3 of the game. (refer to Figure 1)

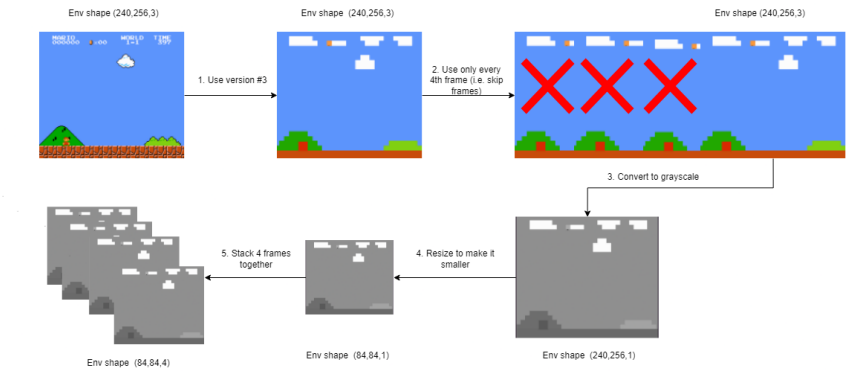

Figure 1: Simplified Environment for easier training

- **Action space** corresponds to that of "COMPLEX MOVEMENT" as defined by the gym environment which includes discrete actions like (left, right, up , down , sprint & NOOP)
- **Training** only occurs on (World 1 - Stage 1) while additionally **testing** the agent's generalizability on (World 1 - Stage 2) due to its similar (yet slightly more challenging) level design when compared with the level we trained on.
- **State** consists of pixels of the game after undergoing transformations of skip frame -> grayscale -> resize -> stackframe. (refer to Figure 1)
- **Reward** function used is modified from the original to incorporate the delta of mario's score between timesteps which we will henceforth refer to as the **CoinCollector model**.

3 Project Objectives and Training Setup

3.1 Project Objectives

We aim to study the Reinforcement Learning algorithms DQN (*Deep Q-Network*), A2C (*Synchronous Advantage Actor Critic*) and PPO (P*roximal Policy Optimization*) across 3 different depths/goals.

1. **GOAL 1:** *Battle of the Models*
   (a) Qualitative analysis of agent play-through videos
   (b) Quantitative analysis of various metrics captured via tensorboard logging
   (c) Evaluation analysis of pass count and coins collected rate
2. **GOAL 2:** *Impact of Reward shaping*
   (a) Evaluating the usage of the custom reward metric (Coin Collector) *vs* using the original reward metric of the environment.
3. **GOAL 3:** *Generalizability Test*
   (a) Analysis of subtle behavioural traits on a level the agents are not trained on.

3.2 Training Setup

We standardized the entire code by leveraging stable baselines 3 library for running the DQN, PPO and A2C algorithms for 5 million, 2.5 million & 600k training steps respectively (which is a result of running each algorithm 16-17 hours on Colab) on both the regular reward function & the Coin Collector reward function. The stable baselines package provides us with logging via tensorboard by capturing specific metrics w.r.t each algorithm (e.g. Exploration rate, Entropy Loss, Policy Loss and Value Loss) along with callbacks to plot the best/average reward. We evaluate the model quantitatively by assessing the pass count & coins collected per 1000 plays while also qualitatively assessing the performance by capturing 3 successful and 3 failure plays.

4 Models Intuition & Understanding

The models utilized in this project are: Deep Q-network (DQN), Proximal Policy Optimization (PPO) and Actor-Critic (A2C). Refer Table 1 for an overall comparison of the concept and implementation of DQN, PPO and A2C.

4.1 Deep Q-Network (DQN)

Deep Q-network (DQN) is a value-based reinforcement learning method that uses a neural network to approximate a state-value function within a Q-learning framework (Mnih et al., 2013). It was developed to address the challenges associated with training agents in environments characterized by high-dimensional and discrete action spaces (Plaat et al., 2020). Through end-to-end training with a deep neural network, DQN facilitates the direct learning of optimal policies from high-dimensional sensory inputs.

A standout innovation in DQN, compared to other reinforcement learning algorithms, is the implementation of experience replay (Mnih et al., 2013). This technique diversifies the training experiences by storing transitions in a replay buffer and randomly sampling from it. Experience replay not only disrupts the correlation between sequential training samples, thereby stabilizing the training process, but also improves data utilization efficiency, leading to shorter training times. This feature is particularly beneficial in complex environments where PPO and A2C might result in less efficient training.

Additionally, DQN employs a separate target network, a feature not utilized in PPO and A2C. This network, a clone of the original, is updated less frequently. Although it operates with slightly outdated parameters, this design helps mitigate the unstable learning dynamics often seen with continuously updated models, contributing to more stable training outcomes (Jia et al., 2021).

4.2 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a policy gradient method in reinforcement learning. Developed to enhance practical application, PPO utilizes a cost function similar to those in supervised learning, which allows for straightforward gradient descent with minimal hyperparameter adjustments (Schulman et al., 2017). This design helps PPO strike an effective balance between ease of implementation, sample efficiency, and tuning simplicity.

Before PPO's introduction, Trust Region Policy Optimization (TRPO) was used, but it struggled with computational efficiency in large and complex problems (Engstrom et al., 2020). PPO addresses this by optimizing a clipped version of the objective function, maintaining policy updates within a predefined "proximal" range. This clipping mechanism helps stabilize updates and curbs the tendency of agents to take excessively risky actions (Schulman et al., 2017).

A significant advantage of PPO over DQN is that PPO directly enhances policies instead of relying on action-value estimates, which helps reduce update variances and avoids issues like Q-value overestimation that happens commonly in DQN (Li and Hou, 2019). This allows PPO to support more complex policies than DQN can manage.

Compared to Actor-Critic methods like A2C, PPO offers more stable and consistent learning without the necessity for multiple parallel environments. PPO's clipping mechanism mitigates the risk of large policy updates that can lead to performance collapses, a vulnerability in A2C (Noawaroongroj and Rattagan, 2023). While A2C may excel in environments that benefit from diverse sample collection, PPO's combination of efficiency and stability often results in overall better performance across various settings.

4.3 (S)ynchronous Advantage Actor Critic (A2C)

The Advantage Actor-Critic (A2C) algorithm is a "synchronous" extension of the Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016), merging the Actor-Critic architecture with an advantage function trained in a distributed manner. A2C stands out for several reasons. Firstly, it leverages the Actor-Critic architecture, where an "Actor" suggests actions to take in a given state, and a "Critic" evaluates how good the state is. This dual approach allows for more efficient learning compared to single-network methods. Secondly, A2C introduces the advantage function, which estimates the advantage of taking a specific action over others which helps in improving the stability and speed of learning. Lastly, A2C is

Table 1: Concept & Implementation of DQN, PPO, & A2C

Model	DQN	PPO	A2C
Model Type	Value-based	Policy-based	Policy-based
Algorithm Concept	Estimate Q-value of taking an action	Directly learns policy mapping states to action	Estimates the advantage of taking certain action by learning an actor (policy) and a critic (value)
Sample Efficiency & Stability Technique	Experience replay & Target Network	Clipped surrogate objective & Entropy Regularization	Parallel environments & Advantage loss

parallelizable, enabling the training of multiple agents synchronously updating a shared network state which optimizes computational resources used.

Unlike DQN, which relies on experience replay, A2C updates its policy using fresh experiences collected in each training iteration. This on-policy learning approach eliminates the need for a replay buffer, reducing memory requirements and computational overhead. Consequently, A2C demonstrates greater sample efficiency, particularly in environments with high-dimensional state spaces.

However, A2C's performance can be sensitive to hyperparameters and network architectures which requires careful tuning is often required to achieve optimal results.

5 Model Behavior: Quantitative & Qualitative Analysis

5.1 Goal 1: Battle of the Models

5.1.1 Qualitative Analysis: Video Evaluation

We analyzed three successful and three unsuccessful runs from a set of 1,000 trials using DQN, PPO, and A2C models, focusing on four key aspects: risk level, movements, repetitive behaviors, and speed of stage completion. Video recordings of these trials are available here.

Regarding risk levels, DQN exhibited the most cautious approach, followed by PPO, with A2C showing the highest risk-seeking behaviour. In successful DQN scenarios, the agent often hesitated significantly near cliffs and stairs and pauses long before moving to the right, indicating its conservative strategy. Although its exploration rate sometimes facilitated overcoming obstacles, it also occasionally led the agent into hazards, such as monsters or cliffs. PPO's successful trials showed fewer instances of getting trapped in local optima compared to DQN, and it generally avoided monsters, positioning it as more explorative than DQN but less daring than A2C. A2C, in its successful runs, effectively maximized coin collection by engaging monsters and timing jumps adeptly. Notably, A2C was the sole model to discover a hidden passage rich in coins, offering a shortcut to complete the level.

In movement analysis, DQN's motions were the most jerky and awkward, whereas PPO achieved a balance between simplicity and complexity, resulting in smoother, more fluid movements. A2C executed the most intricate manoeuvres. DQN displayed more repetitive actions, aligning with its overall conservative nature, compared to PPO and A2C. As for the speed of the agents, PPO led the group, followed by A2C and then DQN. This observation correlates with the movement behaviours, where PPO's fluidity allowed it to navigate stages more swiftly. This agent avoided getting stuck before obstacles like pipes, a common problem in local optima, and it did not collide with platforms or bricks mid-air during jumps. This performance is partly due to PPO's clipping mechanism, which facilitates exploration while maintaining controlled policy updates.

5.1.2 Quantitative Analysis: Tensorboard Log Analysis

The TensorBoard log enabled us to assess specific metrics for each model, correlating them with similar concepts across different frameworks. Table 2 illustrates the metrics tracked for each model and their respective categories.

In exploring versus exploiting as shown in Figure 2, A2C demonstrates the greatest level of exploration, followed by PPO and DQN. DQN adjusts its exploration rate dynamically during training, starting high to allow comprehensive action assessment, then tapering to prioritize exploitation as the model converges. For PPO, key metrics include entropy_loss, approx_kl, and clip_fraction. Initially, entropy_loss rises sharply, then exhibits fluctuations with a general upward trend, indicating an increase in exploration

Table 2: Tensorboard Log Comparison between DQN, PPO & A2C

Model	DQN	PPO	A2C
Exploration vs. Exploitation	exploration_rate (Hyper-parameter)	entropy_loss, approx_kl & clip_fraction	entropy_loss
Policy Learning	-	policy_gradient_loss	policy_loss
Value Learning	loss (DQN is value-based model)	value_loss	value_loss
Sample Efficiency	High	Moderate	Low

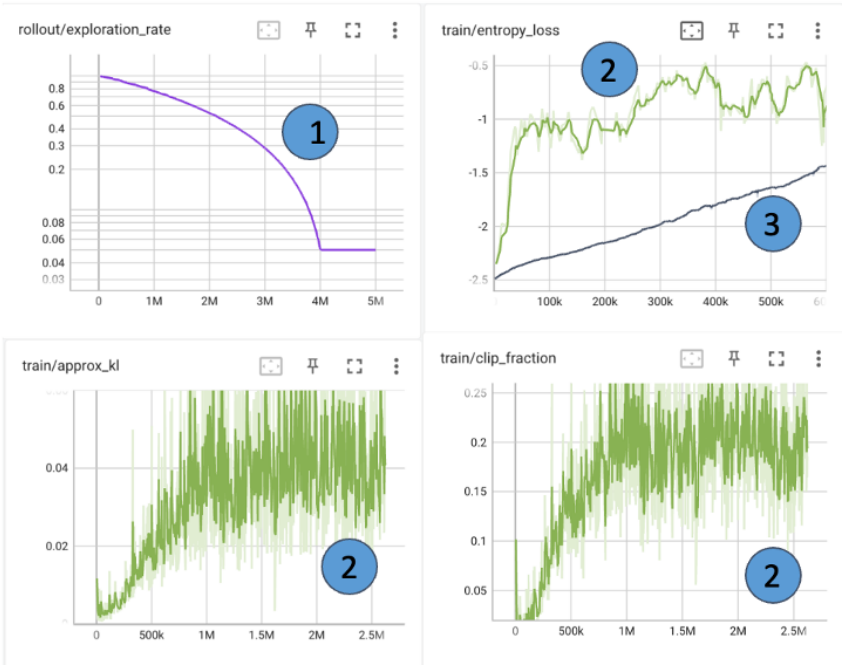

Figure 2: Exploration vs. Exploitation of DQN, PPO & A2C

without the tapering seen in DQN. The fluctuation remains within specific bounds, facilitated by the clipping mechanism. The clip_fraction also shows significant variation towards the training's end, addressing the increase in explorative policies. Conversely, A2C maintains a consistent exploration rate, gradually increasing only towards the end of the training period. It is noted that A2C has not yet achieved convergence and would benefit from extended training, a limitation imposed by our computational resources.

Regarding policy and value learning as shown in Figure 3, DQN monitors only value loss, whereas PPO and A2C track both policy and value loss. In DQN, value loss starts low during high exploration phases and begins to rise as the agent identifies a viable path, suggesting early-stage exploration deficiencies that might lead to later-stage inefficiencies. For PPO, fluctuations in policy gradient loss coupled with a decrease in value loss suggest overly aggressive policy updates, potentially deviating from optimal strategies. A2C shows an increase in both policy and value loss, indicative of divergence from the optimal policy.

In terms of sample efficiency, DQN is the most efficient, followed by PPO and then A2C. This efficiency is measured by the training duration required for the model to perform a set number of steps: DQN requires about 6 hours for 5 million steps, PPO takes approximately 3 days for 2.5 million steps, and A2C takes around 8 hours for 600,000 steps. DQN's efficiency stems from its use of experience replay, which recycles previous rounds to expedite learning. A2C's longer training time is due to its dual-model system involving both an actor and a critic, necessitating more extensive computation.

5.1.3 Evaluation Analysis: Pass Count and Coins Collected over 1000 plays

Besides tensorboard log analysis, we have also performed quantitative analysis based on 1000 plays as shown in Figure 4. We found that PPO completed the most rounds, followed by DQN and then A2C, consistent with our observations from video analyses and TensorBoard metrics. PPO demonstrated smoother movements, and its value loss metrics converged effectively. Conversely, the Actor-Critic (A2C) model, which pursued a more risk-seeking strategy, did not achieve convergence in value loss. Despite fewer successful plays, A2C collected the highest number of coins, largely due to its discovery of a secret passage rich in coins.

Furthermore, we plotted the best and average rewards for each model across the training steps as shown in Figure 4. Notably, the Actor-Critic model occasionally achieved higher peak rewards than DQN, which can be largely attributed to the significant coin collections. The average reward metrics reveal the general performance trend of the models, with PPO performing the best, followed by DQN and A2C. This aligns with our other analytical findings.

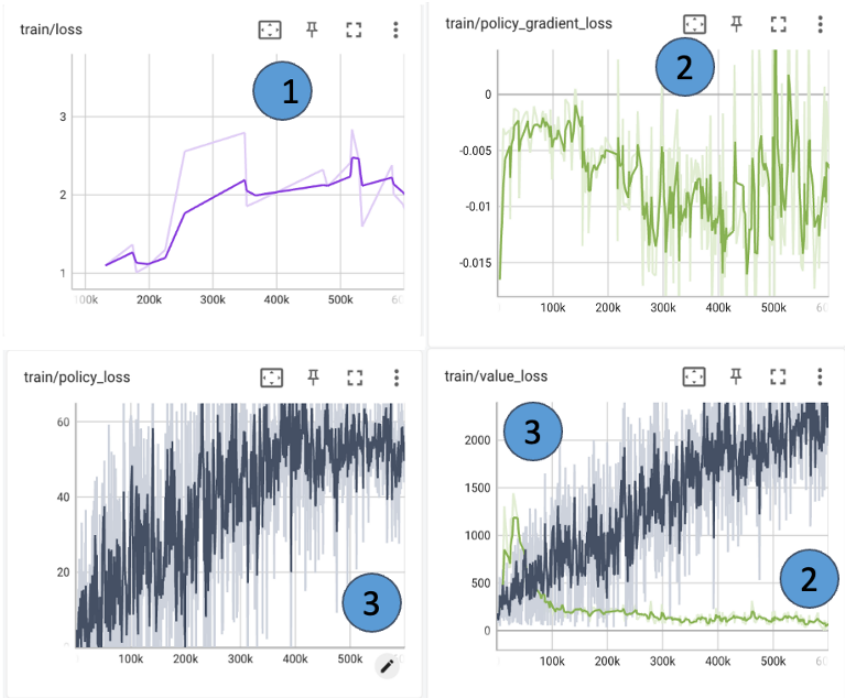

Figure 3: Policy & Value Loss of DQN, PPO & A2C

Overall, from the four plots, it is evident that PPO outperforms DQN in efficiency, achieving convergence in 2.5 million steps compared to DQN's 5 million steps, even though PPO required a longer overall training time. Meanwhile, A2C, despite its lower success rate in terms of completed rounds, excelled in coin collection.

5.2 Goal 2: Custom & Regular Reward

We studied the possibility of altering the agent's behavior by modifying the reward function. The reward function that comes with the gym-super-mario-bros package takes into account the position, speed, and death of the agent, which we refer to as the Regular reward function. Based on the assumption that providing the agent with the game score information would galvanize it to attempt for higher game score, we added the delta game score on top of the Regular reward function, which we call the Coin-Collector (CC) reward function as the game score is most prominently affected by the number of coin the player collects. Other events that affect the game score include defeating an enemy, collecting a mushroom, etc. We separately trained the three agents under the same setting for the same amount of time steps using the Regular and the CC reward functions respectively, and compared their behavioral differences.

The percentage increase of the number of coins collected by agents of the three algorithms is shown in the right of Figure 5, where the three bars correspond to the DQN, PPO, and A2C agents. It can be seen that the DQN and PPO agents trained with the CC reward function collect significantly more coins on average when compared to their Regular reward function counterparts, showing around 20% and 50% increase respectively, corroborating our hypothesis. The A2C agent collects fewer coins on average. However, the results of the A2C agent are arguably inconclusive as it was insufficiently trained (600k steps) as compared to the 2 other agents (5M and 2.5M steps), which calls for further investigation as a part of future work.

Analysis of the recorded playthrough of the agents also revealed that all three 3 algorithms trained with the CC function tend to be more active in defeating the enemies and even sometimes time their jumps to land onto them.

It can be concluded from the results of this experiment that adjustments to the reward function can result in behavioral changes of the trained agent. It also seems to show that by carefully designing the reward function, it's possible to encourage/discourage specific behaviors of the agents.

Furthermore, agents trained with the CC reward function have comparable pass rates when compared to regular agents (left of Figure 5). This may suggest that agents trained with the CC reward function can generate more interesting playthroughs as they would perform actions to achieve higher game scores

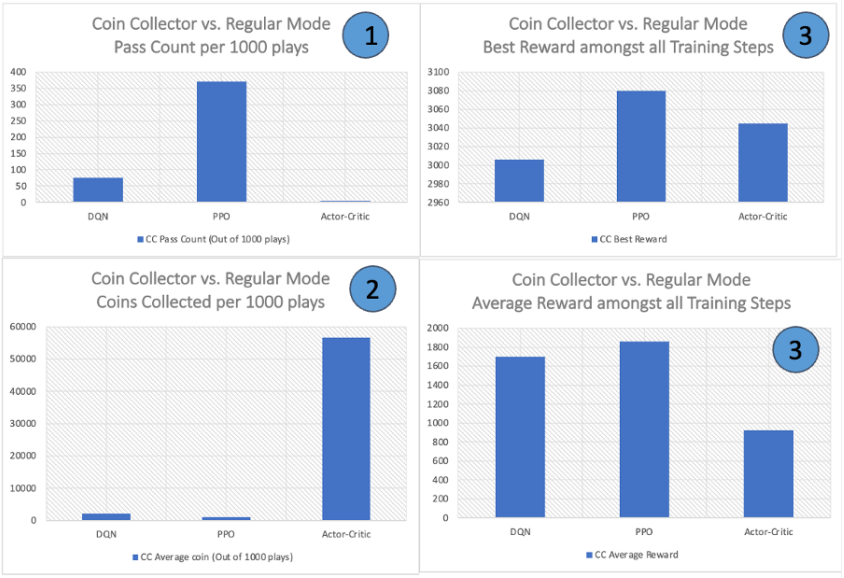

Figure 4: Pass Count & Coins Collected over 1000 plays (Left) & Best Reward & Average Reward of all Training Steps of DQN, PPO & A2C

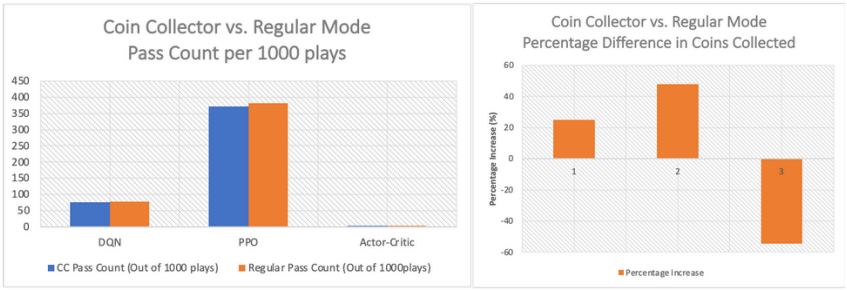

Figure 5: CC vs. Average Reward Function Pass Count (Left) & Percentage Difference in Coins Collected (Right)

without impeding the ability to finish the level.

5.3 Goal 3: Generalizability Test

We explored the generalizability of the agents by testing them in world 1-2, which is a game level that the agent has not seen before in the training as the agents are only trained in world 1-1.

Despite the fact that the colors of the background, terrain, and enemies of the environment are completely different as shown in Figure 6, the agents were observed to be able to navigate through the obstacles and enemies that they had seen in the training process, which demonstrates some extent of generalizability.

Nevertheless, the agents of all 3 algorithms ran into difficulties when encountering unseen enemies. They are observed to bash directly into the unseen enemy and fail the level. None of the agents managed to finish the level out of 1000 trials.

While the agents have shown limited generalizability, significant pieces of evidence that set the algorithms apart are yet to be collected. A possible future direction can be investigating the generalizability with agents trained in more worlds.

6 Conclusion

This project tackled three questions qualitatively and quantitatively in the context of Super Mario Bros: 1) the comparison among the DQN, PPO, and A2C algorithms, 2) the effectiveness of reward function engineering, and 3) the generalizability of the 3 algorithms.

DQN showed caution, PPO balanced exploration and exploitation, and A2C exhibited high risk-seeking behavior. PPO demonstrated the smoothest movements and fastest completion times, while A2C collected the most coins and discovered a hidden passage, despite not achieving convergence in value loss. DQN was the most sample-efficient due to experience replay.

By adjusting the reward function, we have successfully altered the behavior of the agents. Meanwhile, our proposed CC reward function may be a superior choice compared to the Regular reward function as it encourages the agents to achieve higher game scores without hurting the agents' ability to pass the level.

Agents of all 3 algorithms showed some extent of generalizability when being put into a game level that they had never seen before, but their demonstrated generalizability was not sufficient to help them pass the level.

In the future, the A2C model can be further trained to make a more fair comparison with the DQN and PPO models. It is also worth investigating the differences of the algorithms when trained at a greater scale using more levels of the game.

7 Work split

Parashara and Sriram: Environment setup, code standardization, and A2C.
Xian and Grace: DQN, PPO, and results analysis.

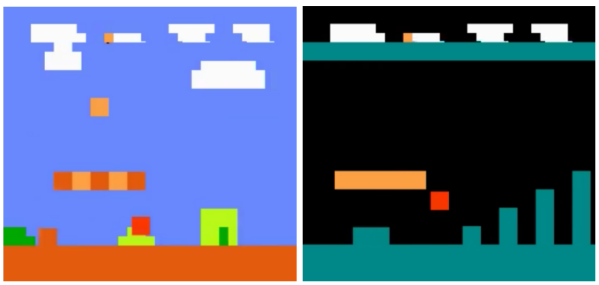

Figure 6: World 1-1 (Left) and World 1-2 (Right)

References

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2020). Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*.

Jia, W., Li, J., and Zhao, Y. (2021). Dqn algorithm based on target value network parameter dynamic update. In *2021 IEEE 4th International Conference on Computer and Communication Engineering Technology (CCET)*, pages 285–289.

Li, Z. and Hou, X. (2019). Mixing update q-value for deep reinforcement learning. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Noawaroongroj, P. and Rattagan, E. (2023). Comparing ppo and a2c algorithms for game levels generation using reinforcement learning.

Plaat, A., Kosters, W. A., and Preuss, M. (2020). Model-based deep reinforcement learning for high-dimensional problems, a survey. *CoRR*, abs/2008.05598.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.