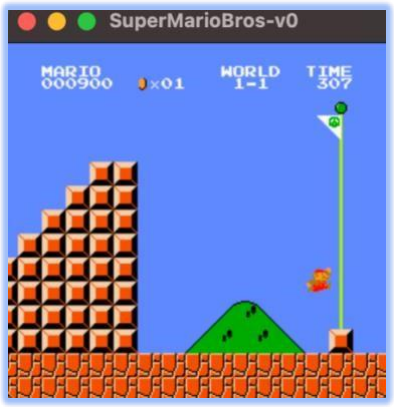
CITS3001

Super Mario Bros AI Agents Report



1. Introduction:

The integration of machine learning in game AI has experienced a surge in popularity, with reinforcement learning and rule-based agent's being two prominent approaches to machine learning. These models are explored for their potential to enhance the adaptability and intelligence of game characters, such as that of the classic Mario in Super Mario Bros.

This project is focused on the examination and comparative analysis of reinforcement learning and rule-based agents in their ability to navigate the Super Mario Bros game, through Open AI's gym environment. This project seeks to unveil how variations in training algorithms and hyperparameters influence the agents' performance, adaptability, and overall success in reaching the game's objective – the flag, in the fastest time possible.

2. Machine Learning in Gaming:

Past studies have highlighted the adaptability of reinforcement learning in diverse and complex gaming environments, where AI agents learn and evolve strategies to optimise game outcomes.

In contrast, rule-based approached have been valued for their reliability and ease of implementation. Although these systems are deterministic and less adaptable, their precision and predictability have made them effective in gaming AI applications.

This project study aims to delve into a side-by-side comparison of reinforcement learning and rule-based systems, offering an insight into the performance metrics and efficiency of both agents used on the Super Mario Bros game.

3. Methodology:

The reinforcement learning agent's I have implemented, trained with the use of Stable Baselines 3 and utilising Proximal Policy Optimization (PPO) and Deep Q-Network (DQN), underwent a series of training episodes with variations in hyperparameters with the aim of optimising performance in both the training and gameplay. The learning trajectory, adaptability, and game outcomes of the agent have been recorded through several experiments to determine and highlight the differences between the algorithms.

PPO is a policy optimisation method, that aims to improve the policy whilst also ensuring the updates do not change the policy too drastically, avoiding training stability issues, resulting in more consistent and reliable outcomes. It learns from the data such as the states, actions, rewards, that is collected using the current policy.

Contrastingly, DQN is a value optimisation method, estimating the value of taking actions in states to optimise the policy indirectly, utilising deep learning to approximate the Q-value function – the total expected rewards, beginning from a state and taking an action.

It is stable and robust, by learning from previous experiences and data utilising a replay buffer and a target network to stabilize the Q-value updates.

The rule-based agents were written with a set of predefined rules, aimed at navigating the game environment efficiently through environmental detection. This agent's performance served as a benchmark to evaluate the adaptability and learning capabilities of the RL agents.

Metrics including the time required to train the agent, reward (in this case, the distance Mario reaches in an episode), in-game actions per episode, in-game time taken to reach the flag, and overall success in reaching the flag, coins collected, and Goomba kills, were used to assess and compare the in-game performance of the agents.

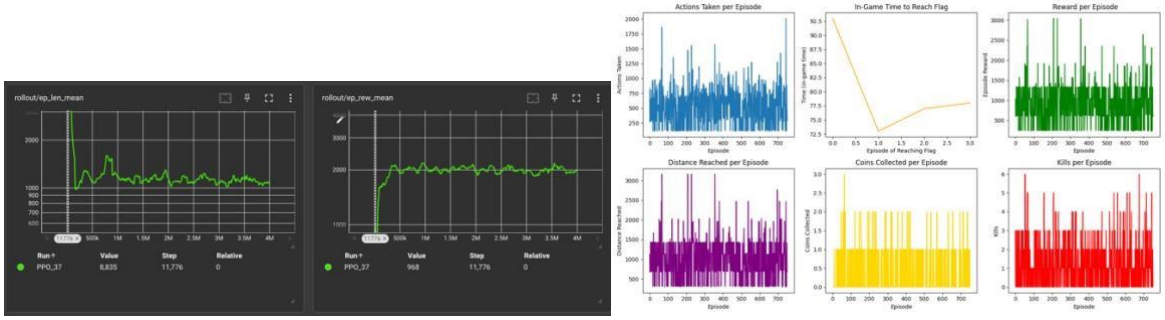
4. Analysis:

Through TensorBoard visualisations and extensive game renderings, the RL agent exhibited an intricate pattern of learning and adaptability. The variations in hyperparameters led to differing learning trajectories and game performance outcomes, underlining the complexity of RL training. One key point to make is a reward of around the 3000 mark generally resulted in frequent first level completions, while reaching a distance of over 3000 in game was indicative of the flag being reached.

The rule-based agent, while at times more consistent in performance compared to some reinforcement learning models, exhibited limitations in adaptability. Its deterministic nature was evident, providing both strengths in predictability and weaknesses in flexibility as well as adaptability during gameplay, as it is challenging to handcraft rules that cover every possible scenario optimally. One strong advantage for RB agents, is that they can be deployed immediately without the need for extensive training periods. They are often relatively simple or straightforward to implement, especially for deterministic tasks where rules can be clearly defined.

A thorough analysis revealed the RL agent’s superior adaptability and learning efficiency in diverse game scenarios. The training for all RL agents were performed on v0 of the Super Mario Bro’s game. The strengths that were revealed during this project was the efficiency, especially that of PPO and DQN when hyperparameters are set appropriately, as well as the versatility, adapting to wide range of environments when exposed to them. Once trained, RL agents can usually generalise their learned policies to new, unseen scenarios, provided they have been exposed to a diverse set of situations during training. Unfortunately, this was yet to be seen in the training completed during the timeframes set as the models often remained within the first level and only just reaching the flag of the first level by the end of the allocated training iterations. This leads on to the main weaknesses of RL agents, with training time being significant in order for the model to converge to an optimal policy, which is also a direct result of the computation complexity required, being quite expensive and intensive on computational resources.

The first RL agent I produced was trained with PPO, trained up to 4 million steps, with a learning rate of 0.00001 resulting in inconsistent performances, however, was able to reach the flag in the first level multiple times during testing. The reward across training progression is outlined in the graph below, demonstrating the visualisation testing outcomes. The reward value stabilised across the 2000 mark after 500,000 steps were reached, which, without improving much over the course of training which went up to 4 million steps. There were slight improvements towards the end which fluctuated and resulted in the model with the most effective in game results. The training graphs graphs reflect the game performance metrics, as Mario was able to reach the flag four times in 750 episodes, not often reaching a far enough distance consistently as well as taking varying in game times to reach the flag. The path Mario takes evidently varies across each episode with the number of coins collected being between 0 and 3 and reaching a maximum of 6 Goomba kills, however, often dying early in the game at the 1000 distance mark on most attempts, which is the section where 2 enemies fall off blocks following a gap. Interestingly, this was the first model to be able to reach the flag in the shortest training time, reaching it as 960 000 steps into the training.



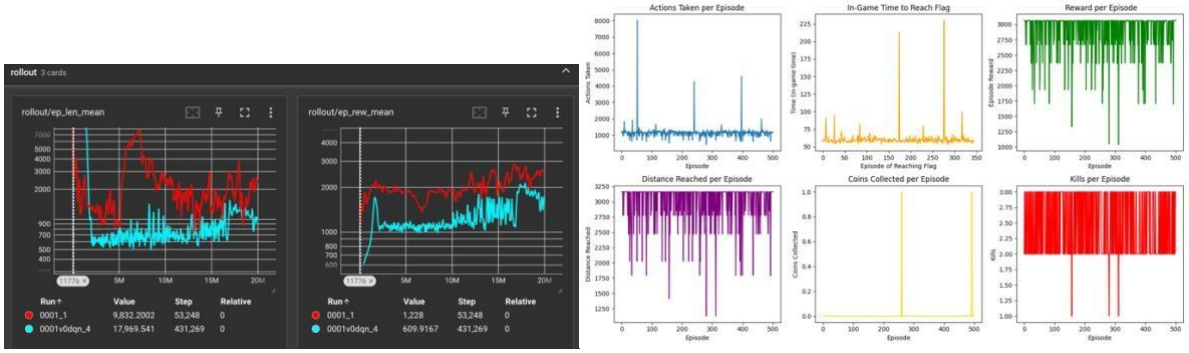
Following this model, I delved deeper into fine tuning hyperparameters, as well as looked to including additional pre-processing for the environment, such as grey-scaling, frameskipping, frame-observation scales, episode lengths, and frame stacking.
The various learning rate adjustments can be seen in the following graph, highlighting the fact that learning rates of 0.00001 and 0.000001 are never able to reach the same reward heights of other models, while also varying significantly in the episode lengths during training progression.

Through all of these adjustments, and fine tuning the models hyperparameters with a new learning rate of 0.0001, number of steps and batch size values, and increasing the training load to 20 million steps, a new, effective model was produced. In this model, as reflected by the red in both the training graphs and the visualisation metrics, Mario is able to reach the flag consistently and in consistent, fast in-game times. Over the 500 episodes, Mario reached the flag over 70% of the time, at an average of 60 in-game seconds.

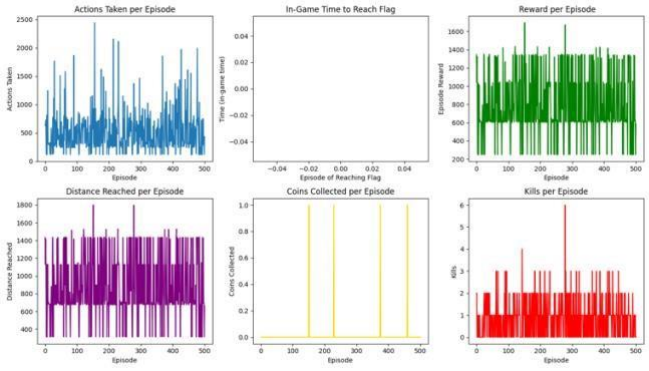
The consistency is evident as shown in the in-game time taken to reach the flag, in addition to the actions taken per episode. There are several outliers in the performance, in which cases it appears Mario gets stuck at an obstacle, running directly into it until either time runs out, or in one of the episodes eventually reaches the flag.

The path Mario takes is also evidently very consistent, collecting no coins in any run, indicating the route Mario takes each run is almost identical each episode.
The training graph reflects this, showing an increase over time peaking at 16 million steps, then fluctuating and gradually increasing again towards 20 million steps.

It is also evident that increasing the steps for RL agents dramatically improves the overall performance of the models, as reflected by the model jump of 4 million steps to 20 million steps, with the former only reaching the flag 0.4% of the time in comparison to the latter reaching it over 70% of the time.

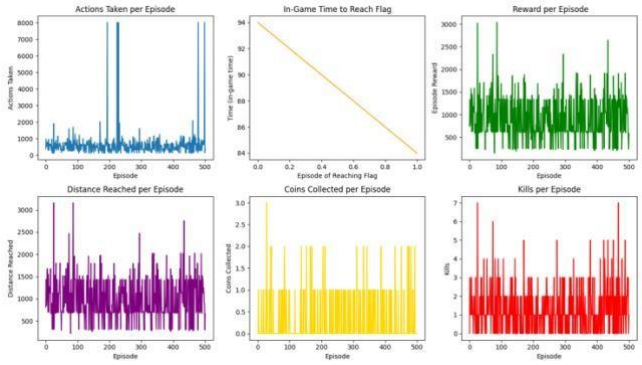


In contrast to this, I trained a DQN MLP Policy model to 20 million steps, as conveyed by the light blue data in the training graph above. In terms of reward over steps ratio, this model trained significantly slower compared to the PPO CNN Policy, gradually increasing but never quite reaching the same reward as the previous model. Over the 500 runs, this model never managed to reach the flag and was unable to consistently make it even halfway through the game, often getting stuck and dying at a distance of 800. The actions Mario takes each episode also varied each episode significantly, reflecting the inconsistent performance for this model.



To distinguish the performance differences between the MLP and CNN Algorithms, I have trained another DQN model using CNN. The DQN model using CNN Policy, performed

significantly better than its MLP counterpart. At only 4.5million steps into training, the model was able to reach the flag multiple times during testing as seen in the graphs below.



Convolutional Neural Networks (CNNs) are designed to process data with a grid-like topology, such as images. By using CNN Policy with PPO, the state input to the agent has spatial or grid-like structures, such as images from game environment's, meaning it would be an appropriate choice for Super Mario Bro's.
Whereas Multi-layer Perceptron's (MLPs) are feedforward neural networks that do not consider the spatial structure of data and treat input features independently. By using MLP Policy with DQN, it means that the agent's state input is best suited for flat vectors, as they look for relationships and patterns across the entire input.
In this project, using CNN Policy is more suitable, as it can make use of the spatial hierarchies in the data.
As was evident during the training process, CNNs generally can be more computationally expensive compared to MLPs for the same input size, reflecting the idea there is always a trade off in reinforcement learning. CNNs are also generally more robust and lead to better generalisation in environments such as Super Mario Bro's with visual inputs.

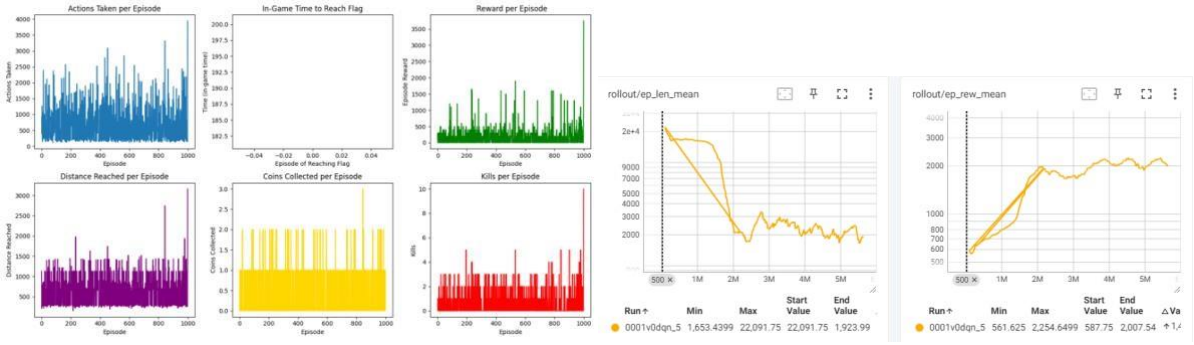
As seen from the data, the choice of policy can affect the learning dynamics. Using CNN Policy has evidently exhibited different convergence rates, reward dynamics, and episode lengths compared to that of MLP Policy, even when other parameters are kept constant. Thus, this significantly affects an agent's overall resultant performance.

Run	Value	Step	Relative
000001v0_1	540.1	20,000,768	2.084 day
00001_1	248.7	17,580,032	3.294 day
00001v0_1	1924	19,099,648	4.909 day
0001_1	2629	20,000,768	2.226 day
0001v0dqn_4	1636	19,999,523	1.101 day
PPO_37	1970	4,000,256	2.368 day

The above data reflects the time taken in real time for the training to complete for the respective models, DQN using MLP policy was by far the fastest to train for 20 million steps, however, this did not result in good performance. PPO using CNN took significantly more time to train without environmental pre-processing such as that displayed by PPO_37 at 2.3 days for only 4 million steps, and 00001v0_1 which took 4.9 days to reach 19 million steps, After applying the adjustments such as grey-scaling, frame stacking, frame skipping, and

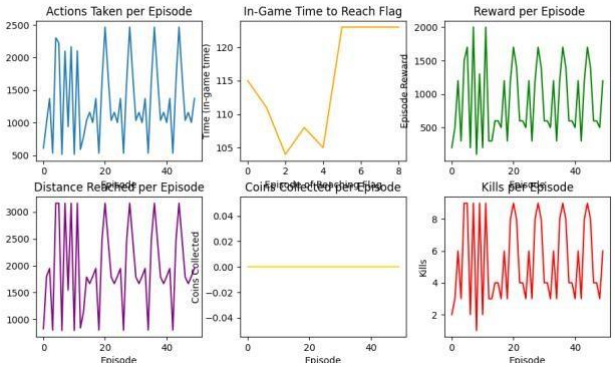
frame observation scale, the time to train was significantly improved, as displayed by 0001_1 which is was PPO model with these applied.

The first rule-based agent that was implemented used the info dictionary provided by the gym environment. This agent was not very consistent by any means, with inconsistent actions based on the specific game situations and scenarios. The result was that out of 1000 runs, it only reached the flag once, with significant variations in its actions taken per episode, but the distance reached each episode did remain consistent at the 500-1000 mark. This agent also managed to collect more coins than other agents along the way.



The second rule-based agent was implemented using open CV template matching code provided by Lauren Gee to strip information from the game with the basic goal to be moving as far right as possible while increasing score via enemy kills, it has four main rules followed in order of priority : The first two most important rules are: 1) Enemy encounters – the agent has subset checks using information such as speed travelled, jump path prediction, jump thresholds and enemy distance to work out the ideal course of action to be taken, as the agent also strives to maximise the score, the rule is set up to prioritise enemy kills rather than full avoidance. 2) Gap check works by scanning ahead of Mario for a certain threshold and checks that there are blocks in the estimated location and when a block is not in an expected position, it informs the agent to prepare for a jump. These two rules were higher priority than others since failure in these checks resulted in death for Mario which outweighs the inherent gain from moving right. The remaining rules are for pipe and stair detection, being stuck only makes the agent take longer to finish the stage so it was deemed less critical, and there is no real difference in performance in switching the order of the two checks.

The model was created through trial and error and fine tuning parameters to find the optimal distance to make actions, and although an initial run will result in a consistent flag reaching, which would lead you to believe it should have no problem to clear stage 1 consistently, within minimal discrepancies due to model altering and fine tuning parameters, mid-run the timing of actions has a significant impact on the performance with the first 8 episodes having a unique performance every time and agent winning more consistently however every set of 8 games after it will follow the exact same pattern of actions resulting in the model to clear stage 1 every 1 in 8 tries as shown below.



The Rule Based Agent underwent a series of transformations in order to reach its final state through extensive debugging especially through the use of print statements, the following main changes were implemented to fine tune parameters in order to reach better consistent performance were.

1) The choice of priority for the rules: Originally did not have enemies being number 1 priority resulting in sub-par results and frequent deaths.

2) More precise case checks: Such as early implementations had when enemy within certain distance Mario should jump, this was changed to return actions such as move left 3) Finer grain control on actions: Originally returned an action when a test condition is met, but by changing this to controlling the number of frames an action is repeated reduced the randomness of results of same test case and more healthy runs

4) Finer Control on variables: As Mario's actions are affected by the speed he is travelling e.g., having Mario jump 100 frames when he is standing still will result in a different travel position from having Mario jump 100 frames when he is sprinting. Implementing a momentum counter to change variables in real time influencing actions from test conditions resulted in a significant increase in score performance and overall better run performance in most metrics measured such as enemies killed, distance travelled etc.

Overall, comparing the completed RL PPO model with the completed RB agent there are noticeable differences. The first would be the consistency. The PPO model exhibits more consistent behaviour, suggesting that it has learned a near-optimal strategy. In contrast, the rule-based model displays cyclical patterns in various metrics, indicating it's following a set of predefined rules that lead to periodic changes in behaviour. Another is the efficiency, where the PPO model seems to stabilise at a faster gameplay (with a lower time to reach the flag) compared to the RB model. Adaptability of the agents is another point of difference, the PPO model's early episodes show exploration, which is absent in the rulebased model. This indicates the RL model's ability to adapt and learn from the environment, while the RB model strictly adheres to its predefined rules. The final comparison is of rewards and outcomes, in which, as intended, both models seem to prioritise reaching the flag and exhibit similar behaviours concerning coin collection and enemy kills. However, the PPO model's consistent high reward indicates it is significantly more successful in achieving the game's objectives compared to the RL model.

5. Discussion:

This study highlighted that hyperparameter variations play a pivotal role in shaping the RL agent's learning and adaptability, and thus performance. Different settings led to varied learning speeds, adaptabilities, and overall game performances, emphasizing the criticality of optimal hyperparameter tuning. In this case, it was found that a learning rate of 0.001 for the PPO algorithm was most effective, in addition to a gamma of 0.9 and number of steps being 512. It was also found best to use a buffer size of 100000 for the DQN model, given the computational resources used.

In addition to this, skipping every 4^{th} frame, frame stacking by 4, resizing the observation frame scale to 84x84, and converting the image to greyscale, significantly improved the training efficiency and time taken to process data. It also improved the memory allocation of models, from models being saved every set number of steps at almost 300MB, to under 30MB after these were implemented.

In scenarios of complexity and unpredictability, the RL agent exhibited superiority, learning, and adapting dynamically. The rule-based agent, while excelling in scenarios aligned with its predefined rules, faced challenges in more complex, unpredictable environments.

This implementation of the project was constrained by limitations such as time availability and computational resources, which could potentially impact the depth and breadth of insights derivable from the experimental setup we had in place.
Given the opportunity to use expensive computational resources with significantly more time, the models trained with both PPO and DQN algorithms could be potentially vastly more impressive in terms of the agent's performances. Given this, further experiments that would be ideal to analyse as performance metrics would include the generalisation across more levels. Unfortunately, with these trained models, they did not progress in other levels at all, and so including any such experiments was pointless. Furthermore, it would prove beneficial to explore additional algorithms for training and comparing their effectiveness with the tested algorithm's.

This comparative analysis emphasised the RL agent's robust adaptability and the rule-based agent's consistency. These findings offer invaluable perspectives for advancing game AI, with potential extensions to broader applications like robotics and simulation, where adaptability and consistency are critical.

References

Deeplyai (2023) *Super Mario Bros. with Stable-Baseline3 PPO*, *Kaggle*. Available at: https://www.kaggle.com/code/deeplyai/super-mario-bros-with-stable-baseline3-ppo

Enrik Yepes, B. (2023) *The Power of Reinforcement Learning Visiting Nostalgia Through Super Mario Bros*, *GitHub*. Available at: https://github.com/BJEnrik/reinforcementlearning-super-mario/blob/main/cpt1_byepes_ml3_indiv_project.ipynb).

jiseongHAN (21AD) *Super-Mario-RL /duel_dqn.py*, *GitHub*. Available at: https://github.com/jiseongHAN/Super-Mario-RL/blob/master/duel_dqn.py

OpenAI. (2023). *ChatGPT* (Mar 14 version) [Large language model]. Available at: https://chat.openai.com/chat

Schejba, Bc.O. (2022) *Deep Reinforcement Learning for Super Mario Bros.*, *Department of Applied Mathematics Supervisor: Ing. Daniel Vašata, Ph.D.* Available at: https://dspace.cvut.cz/bitstream/handle/10467/101068/F8-DP-2022-Schejbal-Ondrejthesis.pdf?sequence=-1

Schneider, N. (2021) *A simple guide to reinforcement learning with the super mario bros.. environment*, *Medium*. Available at: https://medium.com/geekculture/a-simple-guideto-reinforcement-learning-with-the-super-mario-bros-environment-495a13974a54

Stable Baselines 3 Callbacks (2021) *Callbacks - Stable Baselines3 2.2.0a7 documentation*. Available at: https://stable-baselines3.readthedocs.io/en/master/guide/callbacks.html

Stable Baselines3 Tutorial - Callbacks and hyperparameter tuning (no date) *Google Colab*. Available at: https://colab.research.google.com/github/araffin/rl-tutorialjnrr19/blob/sb3/4_callbacks_hyperparameter_tuning.ipynb#scrollTo=pXa8f6FsRUn8