REPORT TITLE HERE

Connor Wallis (22506057) & Reilly Evans (23615971)

**Export This As A PDF Before Submitting**

**(and delete this)**

Introduction

One of the primary purposes of computers over their history has been to automate tasks that humans normally perform. Many different methods of creating agents for this purpose have been designed, from simple rule-based programs to complex machine learning algorithms. In this report we will construct two agents for the task of playing levels from Super Mario Bros: one rule-based agent implemented by hand, and one Proximal Policy Optimisation (PPO) agent trained using stable baselines. We will compare the performance of these two agents in a variety of areas, demonstrating the strengths and weaknesses of each approach. … [add more detail]

Analysis

The two agents we will be comparing are a rule-based agent and a PPO agent. These agents will interact with Super Mario Bros. using the methods in the gym-super-mario-bros package (Kauten, 2018). The rule-based agent is a simple Python script which reads in data corresponding to the game screen, alongside other information recorded by the package such as Mario’s coordinates and the number of lives he has., then chooses what combination of buttons to input based on that data. The rules used to determine what inputs are made were coded by hand over the course of this project’s development, with it being tweaked manually in response to testing it starting from World 1-1 with 3 lives. Identifying the different platforms and enemies was done in part with code developed by Lauren Gee (2023), with some modifications to increase the range of enemies and blocks the agent could identify. This agent uses the Super-Mario-Bros-v0 environment from the gym package, enabling the full graphics of the NES game rather than simplifying them like the other available environments do.

For our second agent, we used the Proximal Policy Optimization (PPO) reinforcement learning algorithm alongside the Convolutional Neural Network (CNN) policy from stable-baselines3. We decided on this algorithm as it was in line with the projects scope, being a more simple, stable, and efficient algorithm compared to others. We elected to use the CNN policy as it is suited towards grid-like data – in our case, images of the game screen. Like the Rule-based agent, our PPO agent read in the game screen observation and additional information. However, in contrast to the rule-based agent, the game screen observation data is reduced to limit the amount of data fed to the agent (to reduce computational load). This was achieved via grayscale of the three RGB channels into a single channel. The agent will then iterate through steps or ‘frames’ of the game analysing four game screens at a time. This was achieved through the frame-stacking technique, layering four frames into one giving the agent game screens with a sense of motion. After the agent analysed, it would select a decision based off its frame of understanding (model). The decisions were also restricted to 7 possible simple actions, reducing the computational stress and experimentation of the agent. Over millions of iterations of the analyse, predict, learn loop, the agent refined its model attempting to maximise the reward achieved by its predictions.

Selecting the optimal PPO hyperparameters and environment settings proved to be a challenge for our project. As we used newer versions of stable-baselines3 and PyTorch, we believe there were some incompatibilities and lack of publicly available information regarding hyperparameters for a super-mario-bros-gym environment. After 70 iterations of adjusting hyperparameters such as entropy coefficient, learning rate, discount factor, the learning policy, and the environment itself (we tried using super-mario-bros-gym-v3), we had to accept the underwhelming results of occasionally beating 1-1.

The reward value used for training the PPO agent is the one that is included in the gym-super-mario-bros. package. This reward is calculated for each step as

where r is the reward value, x0 and x1 are Mario’s x position before and after the step, c0 and c1 are the value of the in-game timer before and after the step, and d is -15 if Mario died during the step or 0 otherwise. The reward cannot be outside of the range -15 to 15 (Kauten, 2018). This results in the agent prioritising going right as quickly as possible while avoiding death. Starting from level 1-1 with 3 lives, the rule-based agent earns a total reward of 4126. Early in the PPO agent’s training it would achieve anywhere between [XXXX] – [XXXX] reward. After 4 million steps, the PPO agent would plateau at roughly 2000 average reward per 3 lives. It would go on to occasionally beat 1-1.

Besides the differences in total reward, there are several other differences in the agents that affect their usefulness. The rule based agent benefits from consistency; since neither the agent nor *Super Mario Bros.* itself have any random elements, it will always perform the exact same with no chance of random elements affecting its performance. Another notable benefit is being easy and quick to understand and tweak; if a change is desired the code can be directly edited and reran to see the changes, while the PPO agent requires hours of training just to see the results of any changes made. The obvious downside of the rule based agent is that it is incapable of learning from its training directly. Our agent was primarily designed for beating World 1-1, a level which uses the standard overworld colour palette; if made to try and play a differently themed level like 1-2 it will not recognise any of the blocks or enemies due to them being different colours. This will not change unless we directly program in the ability for it to recognise these objects. In contrast, the PPO agent will eventually learn to beat these levels if given enough time to train, with no intervention from the programmer.

With the PPO algorithm and the above reward function, we noticed many strange behaviours. For example, after millions of iterations, sometimes the agent would develop a local optimum such as running into the second pipe indefinitely. This led to us finding a time limit wrapper function for our environment from ClarityCoders (2022). This time limit wrapper indirectly modified the reward function, associating a death penalty for the agent if it did not complete the level in time.

[Strengths and Weaknesses of Rule-Based agent]

Although the PPO agent had an underwhelming outcome, it still posed some strengths in comparison to the rule-based agent. The first of which being that the agent displayed some level of adaptability. The agent would go on to seemingly understand that enemies need to be avoided and pipes/holes need to be jumped over. Given more mechanics, the model would adapt and develop strategies to overcome these if given sufficient training.

Another example of an advantage of the PPO agent is its property of optimisation. As the agent is trying to maximise its reward, and the reward function considers the remaining time, the agent is trying to minimise the time it takes to complete the stage. Therefore, the agent will attempt to create the fastest route possible for completing the stage.

A final example of an advantage from the PPO agent is that it has a degree of generalisation. When posed with newer unseen levels, the agent can extrapolate the mechanics of the stage from previous knowledge. This can be observed when the agent completes 1-1. Upon entering the new blue themed level, the agent will jump over the goombas and the tall pillars as it learned previously from 1-1.

However, not all is a positive with the PPO agent. Training time and resources are a considerable downside. To see any results from the PPO agent, it needs to be given a pre-trained model or train its own model. In our case, we trained the PPO agent on a Nvidia RTX 3070ti using its CUDA cores via PyTorch. Although we had one of the best graphics cards on the market, training still took many days to complete and provided us with underwhelming results.

Additionally, the stochastic nature of the PPO algorithm’s model makes it hard to create tangible results. This is reduced as training progresses but depending on the entropy coefficient, sometimes we were left questioning if the agent was learning at all.

A final disadvantage of the PPO algorithm is with its debugging and interpretability. Due to the cryptic nature and performance metrics of neural networks, its hard to decide if modifying a hyperparameter is improving the agent or not. At a minimum, it requires many hours of training to determine if a modification is beneficial or not. Compounding this is the stochasticity of the agent, which can cause great variations early in the agents learning.

Analyze and contrast the performance of the chosen AI methods.

• Discuss their respective strengths, weaknesses, and suitability for playing Super Mario Bros.

* Rule based is easy to understand; parameters can be tweaked easily to make it jump shorter, etc.,
* Rule based cannot adapt on its own; new palletes like underground or new enemies like Lakitus require adding them to the enemy recognition program, coding new actions to avoid them, etc.
* Experiments could include testing speed of level completion (be it time, frames or actions) or the amount of memory used; many of the listed examples don’t really work with a rule based agent that doesn’t learn
* Rule based takes 2374 steps total; 1618 to beat 1-1

Performance Metrics

While the gym-super-mario-bros package by default uses rightward progression as its primary metric of performance, this is not the only factor on which we can compare the agents. One potential alternate metric is to directly reward completing levels as quickly as possible. Under this metric we would provide large reward boosts at the end of levels based on their world/stage numbers, the time remaining on the in-game timer, and the number of steps since the agent was initialised, while falling back on rewarding moving right quickly within individual levels. While this at first seems to be not very distinct from the existing reward function, it benefits in rewarding the use of subareas to skip parts of a level. Many levels in the game, including World 1-1, have pipes or vines that lead to bonus rooms then return Mario to a later part of the level, skipping a large portion of platforming. While these shortcuts make for faster level completion, they result in less rightward movement overall and thus are discouraged by the current reward function; rewarding quick stage completion allows these shortcuts to be utilised by the agent. This benefit is emphasised even more with the Warp Zones, rare rooms that can allow the agent to skip entire worlds; rewarding agents who manage to locate these will promote an agent that gets to the end of the game as fast as possible. However, this metric’s usefulness is limited on our agents due to them not getting very far into the game; the rule-based agent barely makes it past the start of 1-2, while PPO only occasionally completes 1-1.

Another potential metric for our agents is points. Like most games of the time, *Super Mario Bros*. includes a points system that rewards various beneficial actions, like collecting coins and having lots of time left on the in-game timer, with points. By tying the reward function to how many points a given action earns, we will be measuring our agent’s performance in a unique way. Rewarding points encourages many unique behaviours, such as kicking Koopa Troopa shells into other enemies or hitting the top of the end-of-level flagpole. The points given for time remaining also ensure reaching the level’s end is still a priority, so the agents should still progress through the game. However, one change that must be made for this metric is to either restrict the agent to 1 life for each attempt or apply a heavy penalty to dying. The reason for this restriction is that *Super Mario Bros.* has a method through which a player can gain infinite extra lives by bouncing on Koopa shells on a staircase endlessly. Without one of these changes an agent could be incentivised to stall on one of the levels where this is possible, endlessly gaining points and lives but never progressing through the game. Additionally, the reward function will need to provide some positive reward to moving right, albeit less than the default function, in order to ensure Mario moves far enough into the level to start earning points and give the agents some direction for how to improve. Without this change, untrained agents will earn no reward as they move erratically at the start of the level (where there is nothing that can earn them points), making early training progress very slow. From 1-1 the rule-based agent earns a total of 21400 points, while the PPO agent[describe how models fit points]

You will notice that gym-super-mario-bros reward function assumes the objective of the game is to move as far right as possible. You are encouraged to come up with other performance and evaluation metrics for your agents. Novel and interesting metrics that you come up with will be rewarded.

* Points as a metric (collect coins/powerups gives lives and increases survivability; death penalty prevents infinite lives trick from causing problems
* Progression as a metric (get as far into the game as possible in terms of levels, or alternatively beat 8-4 from 1-1 as fast as possible; rewards finding the Warp Zones)

Visualisation/Debugging

Many techniques were used to help visualise the decision process of our agents. For the rule based agent, statements were added to the code that printed statistics to the terminal that we could use to identify what information the agent was accessing and using to make decisions. Gee’s code (2023) already included some commands of this nature, printing lists of locations for each identified entity to the terminal with each step it made. We added statements to print when the agent considered itself to not be mid-air, as well as directly stating what action it performed each frame. Another trick used to help visualise the decision making process was to freeze the game the instant the agent decided to jump, giving us time to analyse the game’s state and see what conditions made it decide to do so. This was achieved through printing thousands of lines to the terminal, forcing the agent to wait until they were printed while we analysed the game.

-PPO printing (tenserboard, other visualisation techniques)

Includewhatvisualizationtechniquesyouusedtogaininsightsintotheagent’sdecision- making process.

Include what debugging/profiling tools you utilised to optimize the algorithms and enhance performance.

* Besides what was already in Lauren’s code, printing to the terminal when decisions are made helps identify actions
* Freezing the game when a decision is made (via spamming the terminal with 250,000 messages) helps to identify exactly what constitutes a scenario where said decision is made

References

* Kauten, C. (2018). *Super Mario Bros for OpenAI Gym*. GitHub. Retrieved October 10, 2023, from <https://github.com/Kautenja/gym-super-mario-bros>
* Gee, L. (2023). *mario\_locate\_objects.py*. UWA Learning Management System. Retrieved October 5, 2023, from <https://lms.uwa.edu.au/bbcswebdav/pid-3405777-dt-content-rid-43562900_1/xid-43562900_1>
* [other references; use APA7 style; pytorch tutorial?]
* https://github.com/ClarityCoders/MarioPPO/tree/master

Misc. notes [DELETE BEFORE SUBMITTING]:

Our agents are Hand Implemented Rule based agent and PPO from Stable Baselines

When we make changes to existing code (Laurens, pytorch tutorial) document it!

‘poetry run nes\_py --rom super-mario-bros.nes --mode human’ for human controlled Mario