UWA CITS3301 Semester 2 2023

Constructing AI Algorithms to play Super Mario Bros

Comparing Rule-Based Logic and Reinforcement Learning Models

Callum Brown

10-3-2023

1. Introduction

This report will examine the performance of two different artificial intelligence algorithms in a game playing context. The primitive environment established by a 1990's version of Super Mario Bros introduces a number of simple hazards and side goals alongside the main task – **navigate to the rightmost possible part of each level** (and retrieve Princess Peach from the clutches of Bowser). Each of the algorithms will be influenced by these goals, prioritising tasks hierarchically to complete this goal while keeping Mario out of harm's way. The first algorithm is a rule-based logic solution, which makes simple frame-by-frame decisions based on Mario's proximity to nearby objects. The second is a much more complex reinforcement learning model which makes use of simultaneous sampling and stochastic gradient ascent through iterative training.

2. Analysis

2.1 A Rule-Based Logic Model

Using the 'locate_mario_objects' code supplied by Lauren Gee [1], enemies, blocks and items of interest are identified at different coordinates in the frame for each timestep as the agent navigates the level. A hierarchy is established by the AI prioritising which objects Mario should deal with first as he moves. Enemies are deemed the greatest threat, and such are dealt with first. Mario will jump over, on or attempt to avoid each enemy until he has moved safely onwards, and only then will he worry about other obstructing objects. The first of which are pipes, which the algorithm will attempt to jump Mario over once he has enough forward momentum. Throughout the entire process the algorithm is scanning for gaps in the ground blocks so when gaps are found, Mario is instructed to jump over them, with jump height scaled based on the size of the gap. Finally, other obstructing blocks will be examined and jumped over.

Each timestep, the algorithm will return an action as soon as it finds a necessary one, moving to the next step without even looking at objects of lower priority. The hierarchy of action selection within the rule-based logic model is as follows:

1. Enemies
    - 1.1 Mario will jump over, adjust himself in the air, and on the other side when he lands if required.
    - 1.2 In cases where there are multiple enemies, the closest enemy is prioritised.
2. Objects
    - 2.1 Mario will jump over pipes given he has enough forward momentum. If not, he will move left until he has a long enough run up for the jump.
    - 2.2 Mario will jump over gaps in the ground in front of him.
    - 2.3 Mario will clear obstructing blocks with a jump.

The result of this hierarchy is a safe, relatively smart, but not very fast Mario. He toddles along to the end of the first level with ease, backtracking to make run ups to tall pipes, and skipping coin boxes and powerups. The issue with this algorithm is its lack of robustness. It isn't very adaptable, and it requires the pre-identification of all objects and blocks within the level. If the algorithm doesn't recognise a particular object, it won't do anything about the object. It has no memory between frames, but it is logical and will work the same way every time.

2.2. A PPO Reinforcement Learning Model

A Proximal Policy Optimisation algorithm is a strong performing example of a gradient ascent method on reinforcement learning, which works by making small, iterative changes to our agent, Mario's decision-making process as it goes. The algorithm alternates between sampling the environment by looking at frames of the game as Mario navigates the level and optimising a 'surrogate' decision-making function using stochastic gradient ascent.

Gradient ascent algorithms generally work by finding the best performing point of a function, and then moving in the direction of the gradient, or tweaking the function so the result moves closer to the best performing point. [2] While the algorithm is much more complex than our rule-based implementation, much of the work has been done for us as we implement a pre-built package called 'stable-baselines3' to train the model.

However, results from testing the model are not always consistent, where a model will sometimes easily find the solution, other times won't. To counteract this, we must find a suitable performance metric to determine when the best model has been obtained from training. In testing, Mario moves MUCH faster than in the rule-based implementation, but he dies a lot more, as the reward function is only concerned with moving right as optimally as it can.

2.3. Comparison

	Rule Based	PPO RL
Training Time	None	Varying but exponential for a strong performing model
Robustness	Only works for the first level.	Works on all levels (to varying success)
Complexity	Simple	Very complex
Performance on predefined levels	Perfect	Very Good
Performance on unseen levels	Null	Good
First to complete level	1 iteration	Approx 1.4 million iterations
Model Scaling	None	Non-linear, see Section 3.2
Play speed	Slow	Very fast

3. Performance Metrics

3.2. The Rule-Based Logic Model

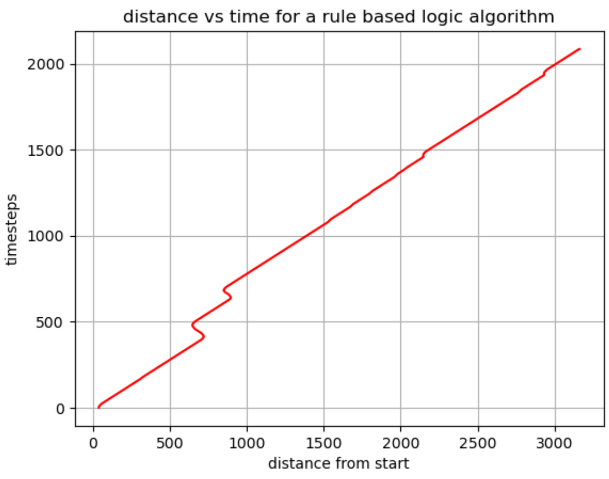

Figure 1 Mario's X path for rule based algorithm

We will first examine the primary goal of the algorithm – to navigate to the rightmost position in the level. Figure 1 takes x_pos from the info returned each step in the process and plots it against the timesteps of the algorithm, visualising where Mario had to backtrack to clear certain obstacles. A smooth line would indicate the algorithm is perfect – mario travels at the maximum speed for the entire duration of the level. The slightly jagged line here indicates Mario adjusted at three different points on the level to make sure he cleared an obstacle or remained safe.

3.3. The Reinforcement Learning Model

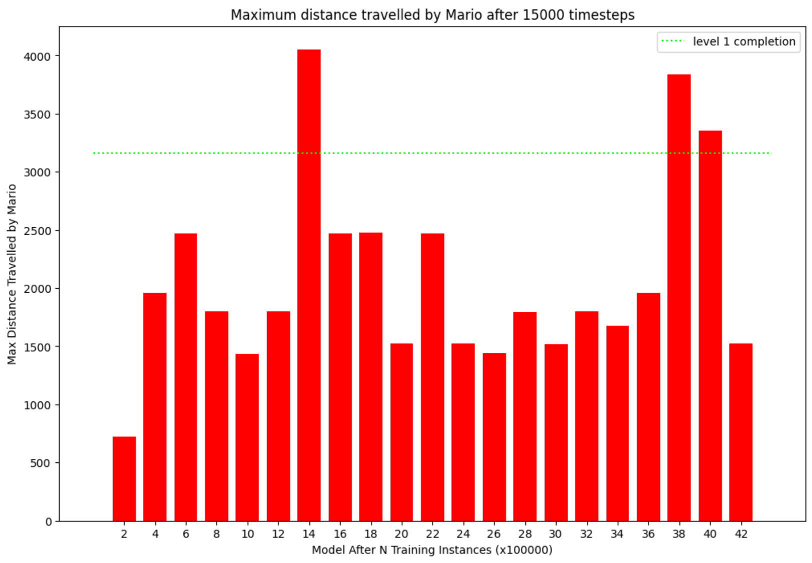

Figure 2 PPO model performance at different training intervals

Utilising a GPU (An NVIDIA GeForce RTX 3080 Laptop GPU) training that would have taken days to took just a few hours to train the model for five million timesteps. The model was trained on v3, stage one. [3]

Figure 2 shows a comparison of performance of the reinforcement learning model at different training intervals. The test is performed on stage one, the training stage. For each training interval, the model is given 15000 timesteps to record its best performance. Three instances completed the level with the 1.4 million model reaching nearly 900 units into the second level, the best performing instance. With 15000 timesteps, models are given ample time to demonstrate capability, and such, this test is a strong indicator of the agent's ability to 'move as far right on the screen as possible'.

The non-linear relationship between model performance and time spent training could be due to a number of factors. The complexity of the Mario environment is the probable cause, in that the level does not exhibit patterns or repeated configurations, and such the same action in different areas of the level will produce different results, which can confuse the algorithm as it explores.

Next, we test the model's robustness by looking at its performance across unseen environments in figure three, or the other stages in the first world. The model at 1.4 million training instances is the best performing (even reaching bowser in the final level) outperforming the 4 million and primitive 20 thousand instance models. The different block styles and changes in layout have a varying effect on performance, with all models performing poorly on stage three.

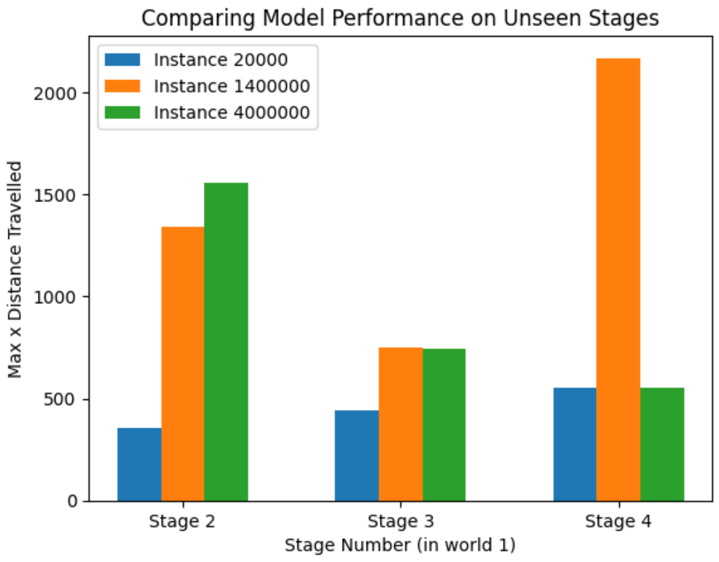

Figure 3 PPO Model performance on unseen environments

Comparing this with our rule-based model, all models automatically win as the agent is confused by differing block colours.

4. Visualisation and Debugging

4.1 The Rule-Based Logic Model

Laying out the algorithm began with by writing out a logical structure of the rules Mario should follow. Initial intuitive decisions were made for each logical situation Mario would find himself in, an example being 'if Mario is on the same y level as an enemy, and the x distance between them is below a certain value, jump!' After implementing this simple rule, we can immediately watch Mario follow this rule by rendering the environment of the game frame by frame. Mario will die if he jumps too late and will have less time to make decisions on the other side of the enemy if he jumps too high. So, tweaks are made to the x distance value changing when, and for how long exactly he jumps, optimising his movement. In development, exact positions of objects and Mario were accessed and printed for fine-tuning changes. Once a rule is optimised, we move to the next most important rule, and implement that.

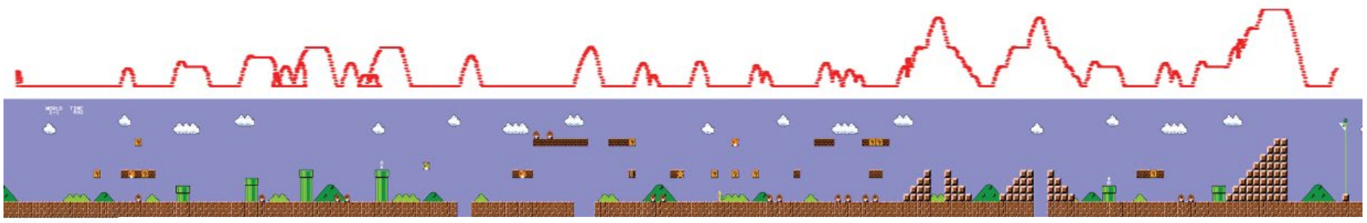

Figure 4 Mario's optimised rule-based path through level 1

The process continues until all the necessary rules are implemented and Mario reaches the end of the level without issue. Figure four shows the final optimised path taken by Mario as the agent uses the final algorithm. See line 20 in the code to enable debug for the algorithm that prints the action Mario takes in each timestep based on the nearest object of interest. Note that if Mario encounters no nearby objects, he will default to walking directly right.

4.2.The Reinforcement Learning Model

The complex nature of the PPO algorithm made it difficult to visualise the progress training was having on Mario's performance. Writing a class to store copies of the model at various intervals of computation, we could now compare the performance of the algorithm in its primitive stages versus after a huge number of iterations. By loading each saved model and running it, we can track how far Mario gets through the level at each stage of computation, and how the model improves upon, or the opposite, compared to previous iterations.

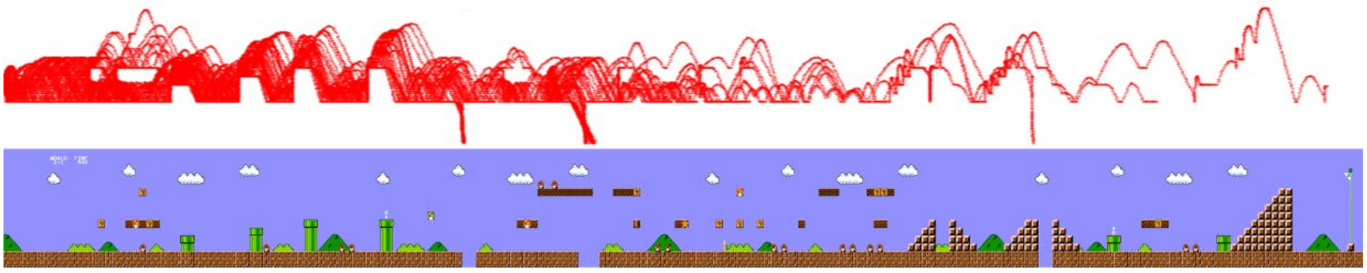
Figure 5 Mario's PPO path through level 1

Figure five is an example of the path the agent takes on its journey to completing level one. Specifically, our 1.4 million iteration trained model has been tested here to demonstrate the variability of the decision making of the algorithm. The red lines that terminate before the end of the level indicate deaths, where Mario has been sent back to the start of the stage. We also see a nice gradient in line saturation indicating Mario does not complete the level instantly, perfectly, but rather navigates the stage until he finds a solution. We can also see he doesn't follow one strict path, indicating the policy is adaptable and not trained to fit the stage directly.

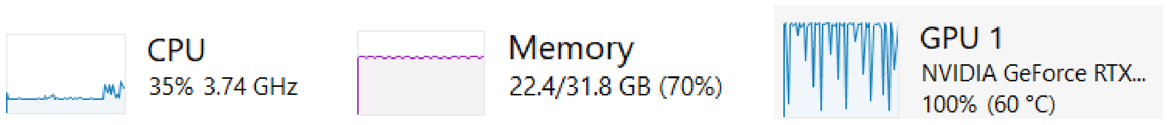

Figure 6 PC Performance during training

Task manager was employed to track GPU and memory usage during training. Figure six shows my PC's usages with just the jupyter notebook running. When training on v3 of the game, training progressed at between 110-135 fps. This was slightly better than training on v0 on average, while training both versions on the CPU alone progressed at roughly 15-30 fps.

5. Conclusion

Comparing the performance of two different artificial intelligence algorithms is a complex task. In the context of Super Mario Bros, our rule-based logic model performed well on the first level, completing the stage without death, while our Proximal Policy Optimisation model, after much training, didn't perform as perfectly, dying a lot, but ran much faster and demonstrated an ability to perform on other unseen levels. After much visualisation and debugging we can comfortably say the two models achieved the same goal 'move mario as far right of the screen as possible', though using completely different decision-making policies.

6. References

[1] Gee, Lauren (2023, September) *code for locating objects on the screen in super mario bros.*[Course Supplied Code] CITS3301, The university of Western Australia. https://lms.uwa.edu.au/ultra/courses/_79122_1/cl/outline

[2] Schulman, John (2017, August 28) *Proximal Policy Optimisation Algorithms* [Academic Article] Cornell University. https://arxiv.org/abs/1707.06347

[3] Renotte, Nicholas (2022, August) *Build an Mario AI Model with Python | Gaming Reinforcement Learning.* [Youtube Video]. https://www.youtube.com/watch?v=2eeYqJ0uBKE

[4] AurelianTactics (2018, December 14) *Understanding PPO Plots in Tensorboard* [Article]. https://medium.com/aureliantactics/understanding-ppo-plots-in-tensorboard-cbc3199b9ba2

[5] Brooks, Ruth (N.D.) *What is Reinforcement Learning* [Academic Article] University of York. https://online.york.ac.uk/what-is-reinforcement-learning/