Super Mario Bros Agent Analysis

Rhianna Hepburn: 23340238

Brittany Carlsson: 22752092

CITS3001: Agents, Algorithms and Artificial Intelligence

### Abstract:

This paper provides a comprehensive analysis of the performance, strengths and weaknesses of three agents: Rule-Based (Hand Implemented), Proximal Policy Optimisation (PPO) from Stable-Baselines3,” *a set of improved implementations of Reinforcement Learning (RL) algorithms based on OpenAI Baselines.”-* (StableBaselines, 2018), and a Deep-Q Network Reinforcement Learning agent from StableBaselines3, in the context of playing Nintendo’s 1983 video game, Super Mario Bros.

This paper analyses the implementations of rule-based, proximal policy optimization (PPO) and deep-q network (DQN) algorithms in the context of the Super Mario game. The rule-based algorithm was hand-written, while PPO and DQN were implemented using Stable Baselines (reference). The aim of this report is to determine the suitability of different machine learning algorithms in creating the most optimal game-playing agent.

# Analysis

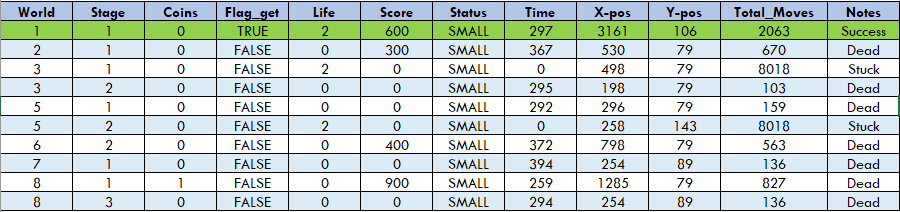
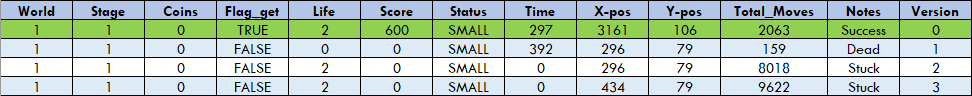
The rule-based agent employs a set of hand-written conditions that have allowed Mario to avoid all enemies and reach the flagpole as optimally as he can. This method ensures that time is not wasted on irrelevant moves such as moving left or jumping when it is not necessary. Overall, this implementation took Mario 103 seconds to complete level 1. The algorithm scans ten pixels in front of Mario (with the origin being the right most red pixel on Mario’s cap), and below Mario to determine upcoming obstacles such as an enemy, pipe or a hole in the ground. Currently, this algorithm has only been programmed for the first world that Mario is in, however, this algorithm does have the potential to be scaled and suitable enough where Mario could complete multiple levels, with the exclusion of boss fights, as this functionality has not yet been implemented.

CHANGE: add tools of analysis and optimization here, condensation of performance matrix, debugger etc into one paragraph

The strengths of the rule-based agent are that Mario completes the level very fast in comparison, unlike the PPO agent. Mario does not make mistakes such as falling into a hole or dying by touching an enemy and it is easy to tweak Mario’s movements to collect more coins, stomp on enemies and jump higher to receive a better score when capturing the end of level flag, which is not possible with the PPO. Not needing to train the agent is another strength as this can be strenuous on a computer’s processor and graphics card and it can take a long time to train the model depending on the number of timesteps have been set. Less timesteps can mean that the model has not had enough time to learn, thus resulting in models with low accuracy and are more mistake prone.

A weakness of the rule-based agent is its inability to adapt to unseen levels of the game. The agent requires code that takes into consideration every minor detail and aspect of each level, in order to hard-code rules for Mario to progress. For example, for an unseen level such as World 1-2, Mario dies instantly from an approaching enemy as the code has not yet analysed the new colour scheme (Table 1.1). Thus, creating an agent that could complete multiple worlds and stages of Super Mario Bros would be very time expensive, and potentially infeasible as the observation space grows. Hence, reinforcement learning agents, such as the PPO and DQN agent are more suitable algorithms for game playing agents such as mario.

**Table 1.1** Statistics for Overworld Selection Using Version 0

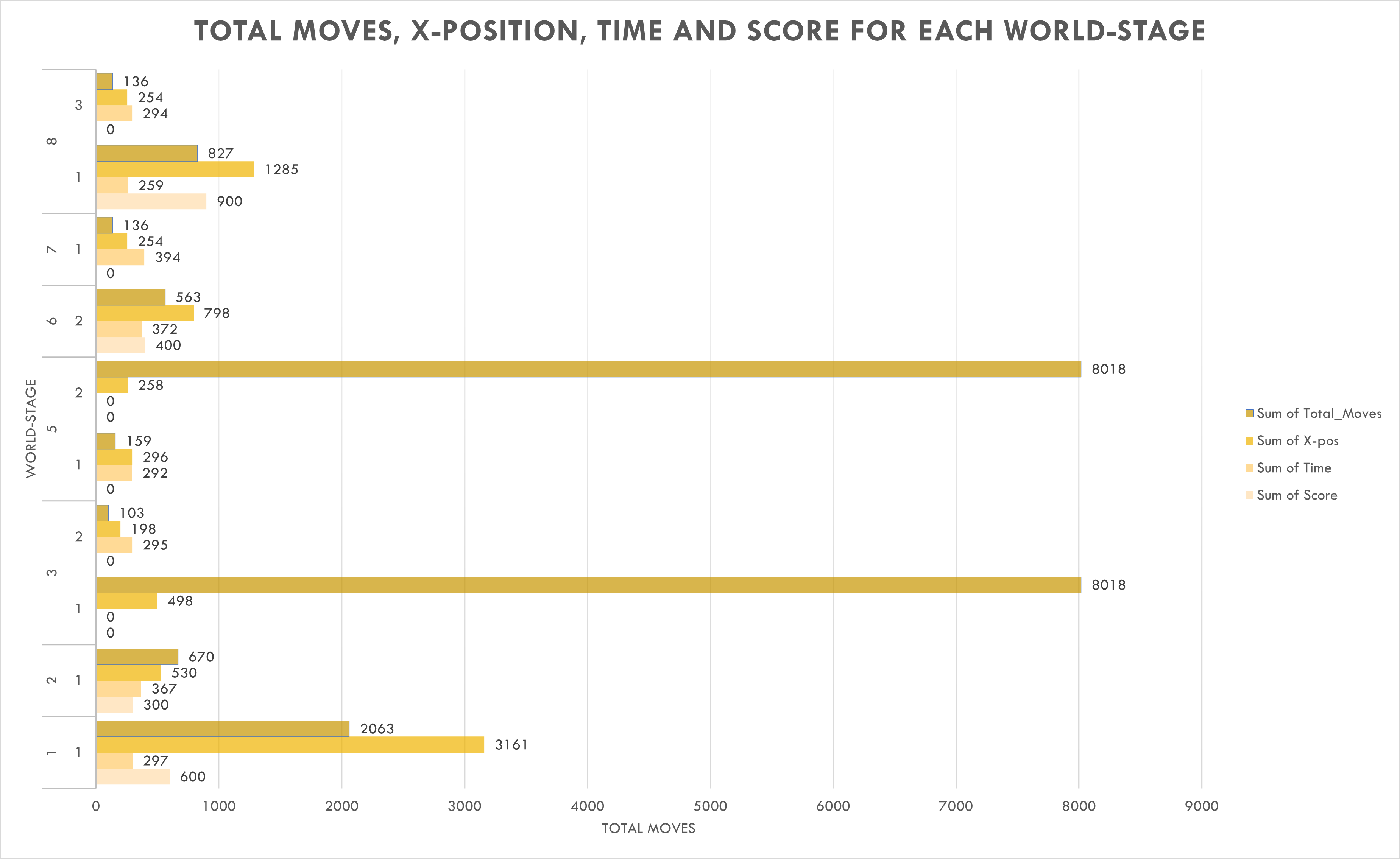
**Table 1.2** Statistics for World 1 Stage 1 Using Version 0 to 3

**Table 1.3** Averages the Overworld Levels Using Version 0

|  |  |  |  |
| --- | --- | --- | --- |
| Time | Score | X-position | Total Moves |
| 257 | 220 | 753.2 | 2069.3 |

**Table 1.4** Averages Across Version’s 0 to 3

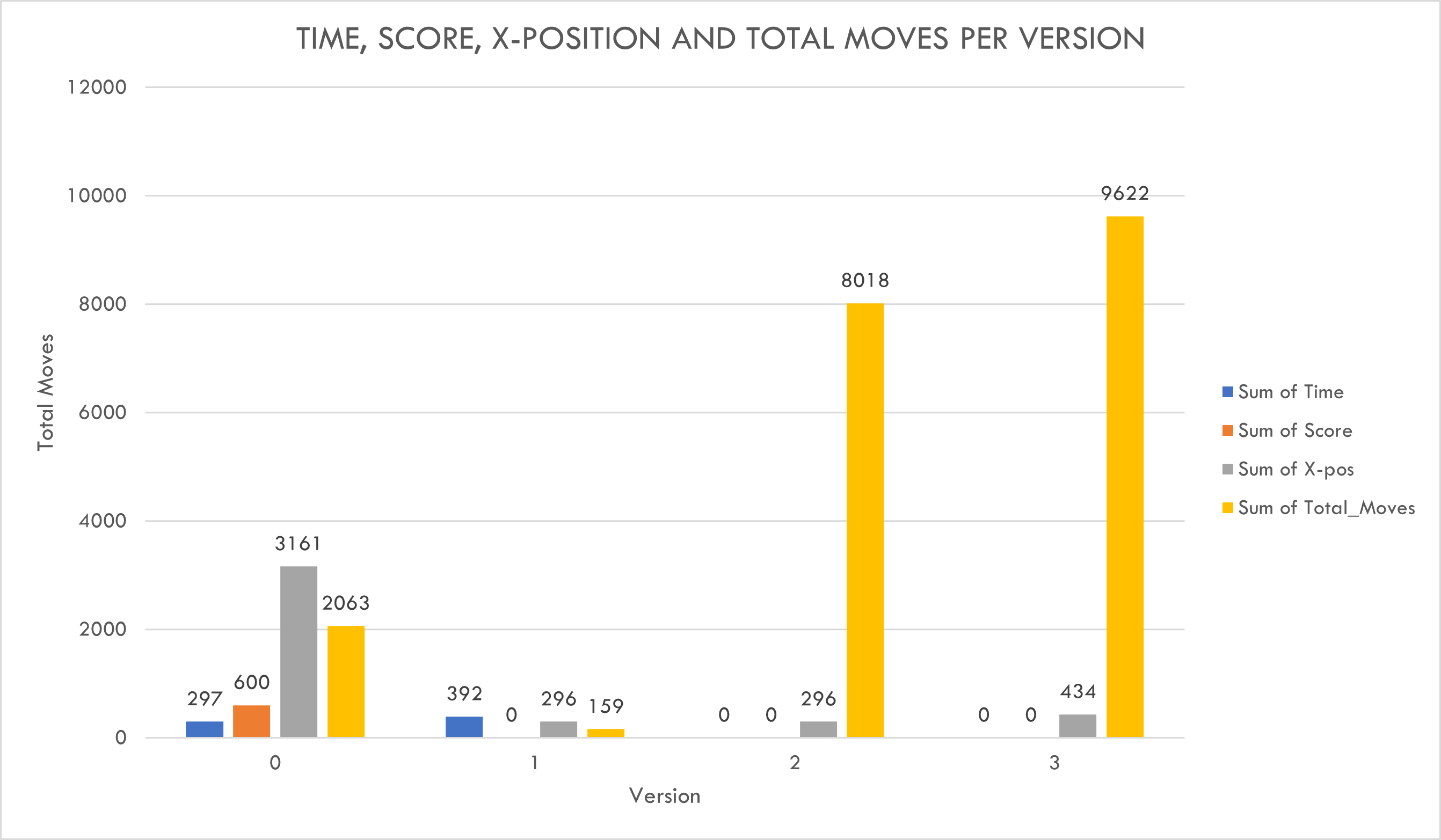
|  |  |  |  |
| --- | --- | --- | --- |
| Time | Score | X-position | Total Moves |
| 172.25 | 150 | 1046.75 | 4965.5 |

**Figure 1.1** Bar graph representing the sum of the total moves, x-position, time and score for the worlds and stages in Table 1.1.

A graph of different colored bars

Description automatically generated

**Figure 2.0** Column graph representing the percentage of level completed by the Mario agent based on x position at the time of death or completion

**Figure 1.2** Bar graph representing the sum of the total moves, x-position, time and score for the different versions (0 to 3) each using World 1 Stage 1, referenced in Table 1.2.

*Performance to Unseen Levels*

Generalisation tests were completed by using "Overworld" levels with similar features that were used to train the Rule-Based agent using version 0. Each test showed that the agent did have the potential to react to the different worlds and stages based on its predefined code for world-1, stage-1.

Referencing Table 1.1, Table 1.2, Figure 1.1 and Figure 1.2, some common trends are present, for example:

* With Mario becoming stuck in two out of the nine worlds, this can indicate that for the most part, Mario can react to the environment around him, but that an enemy that he has not yet encountered is causing his death. In world-1, stage-1, the only enemies were Koopa Troopas and Goombas, yet in later worlds and stages, newer enemies are introduced, which the Rule-Based agent is not yet equipped to deal with.
* A common trend in the data is in relation to Mario's score and x-position. If Mario is not stuck, then the higher the score he has obtained correlates to a higher x-position. This makes sense, as the further Mario progresses in a level, the more obstacles he comes across which can increase his overall score.
* Overall, the data for each version suggests that because the agent was trained primarily using version-0, the performance would decrease as the version number increased. This is attributed to the fact that the agent scans 10 pixels ahead of Mario and obtains the RGB values for each of those 10 pixels and matches it with preset data of each obstacle. With each version changing the environment slightly, this has led to mismatches and has caused the agent not to respond in many situations, thus resulting in Mario dying or becoming stuck in spaces where he would typically be able to pass with ease.
* Lastly, another common trend is in relation to Mario obtaining a higher score (primarily >500) which corresponds to him having a higher x-position and a lower total\_moves. This data suggests that the agent performs optimally when it can understand the given environment based on its data presets that have been manually programmed, hence it does not need to take unnecessary moves to keep moving right. This is clearly seen in world-8 stage-1, where the agent almost reaches the halfway point before dying from an enemy that has not been accounted for in the code. With this knowledge, it can suggest that if the enemy had been considered, then this agent did have the potential to pass this level.

Overall, this analysis provided insight into how this agent could be further modified to have the ability to pass non-boss, Overworld levels.

Note: potentially expand to other levels if time

In this section:

-integrate what visualization tools we used into our discussion, and debugging tools to optimize and enhance (to expand on this section, could potentially use tools to optimize rule-based and discuss our findings)

## *Performance metrics*

Eg.:

-time, distance, points collected, efficiency of moves

-explanation of utility functions used from stable baseline functions

### Rule-based implementation:

General structure of analysis:

-explanation of our implementation

-summary of performance, what metrics were hit

-explanation of the difference in result in comparison to the other algorithms

-strengths of the algorithm in comparison

-weaknesses

-aspects of suitability

*Proximal Policy Optimization implementation*

Research notes:

*Deep Q Network implementation*

Research notes:

* Strength: allows for benefits of Q learning (expand) while removing the infeasibility of storing a state-action space for all possible permutations of Marios world (problematic, especially when trained on multiple levels)

# Performance Metrics

When the Rule-Based agent stops executing, a performance matrix is generated and returned to the terminal. This matrix contains the information from the ‘info’ dictionary provided from the Super Mario gym. This dictionary contains values describing Mario’s journey until he captures the flag, dies or when the time runs out. The total amount of moves is another measurement that was implemented in addition to the ‘info’ dictionary, this helped provide insight into how optimal the moves Mario made were and helped determine if Mario had become stuck trying to complete a random level. Lastly, a message for each situation generates above the performance matrix depending on the circumstance, for example, the message being: “Mario has died” or “Mario got stuck trying to complete the level” or “Mario has captured the flag!”.

The rule-based agent currently has only been programmed to complete World 1-1 and because this agent was handwritten using logic, no training was required. Through debugging, Mario died constantly from all obstacles, however the logic implemented means that he will never die in the first level. Once Mario reaches the flagpole at the end of the level, the performance matrix data will be returned to the terminal.

# Visualisation and Debugging

The rule-based agent was visualised and debugged using three main methods:

1. Visual Studio Code Python3 debugger
2. Microsoft Paint
3. Super Mario Bros game via the Nintendo Switch Online game pass.

*Methodology 1:*

This debugging technique provided clear insight into lines of code where Mario would make incorrect moves that would result in him dying or getting stuck at pipes or blocks. Breakpoints were set and just before an obstacle would affect Mario, a screen shot would be taken, and Methodology 2 would commence. Print statements were also utilised to provide insight into what the next 10 pixels in front of Mario were and what Red, Green and Blue (RGB) values each pixel held. Originally this value was 5, but this led to Mario confusing the pipes and bushes, resulting in unnecessary jumping and lives lost. Through debugging, the optimal value was found to be 10, as this resulted in successful object identification by Mario.

*Methodology 2:*

The screen shot from Methodology 1 was pasted into Microsoft Paint with the grid option turned on. This provided insight and visualisation regarding the pixel distance (x,y) from the right-most red pixel on Mario’s cap to the nearest obstacle. Figure 1.3 Displays a screenshot of the debugging process.

*Methodology 3:*

This was mainly utilised to determine how high Mario was required to jump to get over pipes of all sizes, as well as the optimal distance away from the pipe to make the jump successfully.

The two Reinforcement Learning agents used the following visualisation and debugging techniques:

1. Callbacks
2. Generalisation to Unseen Levels
3. Tensorboard Logs and Wrappers

*Callbacks:*

A callback class sourced from {GET SOURCE <https://github.com/nicknochnack/MarioRL/blob/main/Mario%20Tutorial.ipynb>} aided in both debugging and visualising for each agent. This class is linked to StableBaselines3’s common callback module, which saved the current model every 10,000 timesteps it made before reaching the total number of timesteps to train the agent. This allowed models of 10,000 step increments to be loaded, making debugging easier, as it provided insight into how much longer the agent should train for in order to obtain a better result or to see the variance in success between the different model sets, in relation to the distance travelled across the level before Mario dies.

*Generalisation to Unseen Levels:*

After the PPO and DQN agents had finished their training, their latest model versions were loaded and tested on different versions of the world e.g., SuperMarioBros-v1, SuperMarioBros-v2 and SuperMarioBros-v3. Both models seemed to play as well as they did in SuperMarioBros-v0, their original training environment. To better reflect if this would continue for unseen levels, each agent had been trained

Tensorboard Logs and Wrappers

* Used to visualize the progress of the models training, and an insight into policy conversion, success of the model as it trained (reward function) ,loss function etc

**Conclusions**

Overall, the Rule Based agent proved to be the fastest and least error prone agent of the three when completing world-1, stage-1, in relation to the current testing states that both Reinforcement Logic agents are at. There are benefits to having a rule-based heuristic as an agent because if they are programmed perfectly, they will account for every situation and play without a fault. An implementation as such can be costly, time and resource wise to create such an agent.

References

Stable Baselines. (2018). Welcome to stable baselines docs! - RL baselines made easy¶. Welcome to Stable Baselines docs! - RL Baselines Made Easy - Stable Baselines 2.10.3a0 documentation. <https://stable-baselines.readthedocs.io/en/master/>

Experiments:

Averaged over 100 episodes for the AI agents

DQN trained on level 1 and tested on level 1:

X distanc %: 33.48623853211011

Mean rewards is 689.21 STD reward is : 482.8582047558061 Mean steps is: 49.67 Completed episodes: 0

Percentage of valuable actions: 78.74 Average number of coins per episode: 0.29

(689.21, 482.8582047558061, 49.67, 0, 78.73968190054359, 0.29)

Done.

DQN training on random stages and tested on level 1:

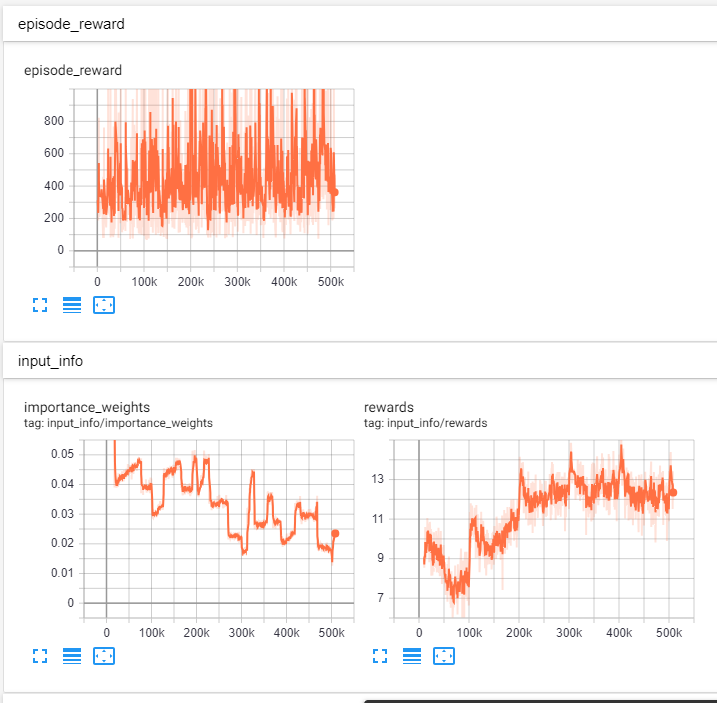
X distanc %: 23.84182220816198

Mean rewards is 524.98 STD reward is : 386.3190386196363 Mean steps is: 41.5 Completed episodes: 0

Percentage of valuable actions: 74.72 Average number of coins per episode: 0.48-Notes: Struggles with jumping over holes, often rewards gained by killing enemies is a fluke rather than an intentional action (eg. Stuck at a pole jumping up and down and happens to land on an enemy)

Tensorflow diagrams:

For DQN agent trained on random stages:



A screenshot of a graph

Description automatically generated

PPO training on random stages (yet to be finished)

A screenshot of a graph

Description automatically generated

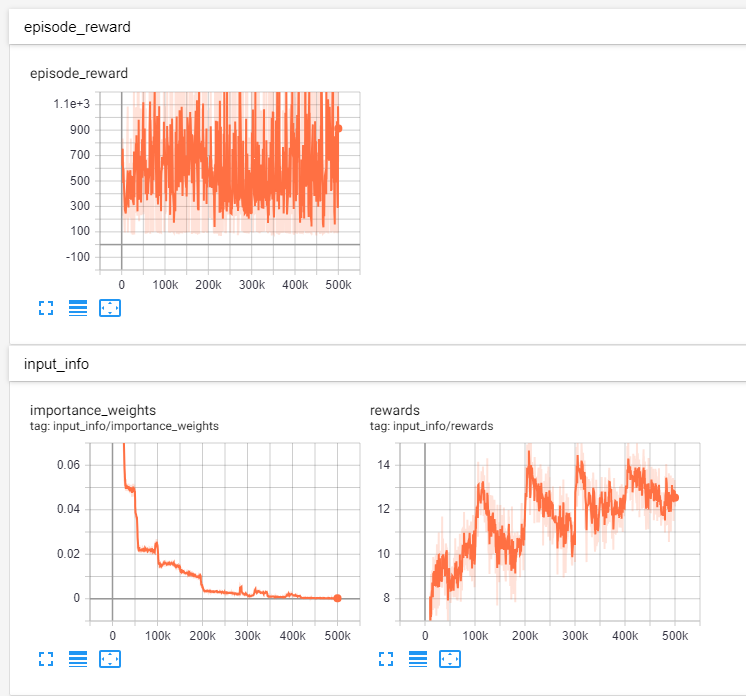
A screenshot of a computer

Description automatically generated

A graph with red lines

Description automatically generated

DQN agent trained on world 1, stage 1 only



A screenshot of a graph

Description automatically generated

Appendix

Figure 1.1 (IMAGE OF MS PAINT DEBUGGING)

*For additional information on APA Style formatting, please consult the* [*APA Style Manual, 7th*](https://apastyle.apa.org/style-grammar-guidelines) *Edition.*

A graph of a graph

Description automatically generated with medium confidence

A graph of a graph

Description automatically generated with medium confidence

A graph of a graph

Description automatically generated with medium confidence

A graph of a graph

Description automatically generated with medium confidence