Exploring Reinforcement Learning through Super Mario Bros.

Hussain Zainal (hzainal@gmu.edu)
George Mason University

***Abstract* - This report examines the application of Reinforcement Learning (RL) in the context of a Super Mario Bros. environment, utilizing the Double DQN framework to train an agent to navigate a classic 2D platformer game. Through a multi-stage development process, four progressively enhanced models are engineered to investigate how reward shaping, heuristic logic, and computer vision affect agent behaviour and performance. Each model introduces specific mechanics to analyze how incremental adjustments impact the results. In addition to evaluating these models, the report also examines broader RL limitations revealed by the Super Mario Bros environment, such as sparse rewards, exploration inefficiencies, and failure to adapt to changing environments. By comparing these findings to challenges present in more complex video game environments, the scalability and generalization of RL in dynamic and abstract environments are highlighted. The results highlight the need for more thoughtful reward design, temporal reasoning, and adaptive policy learning to overcome suboptimal strategies.**

***Index Terms* - Reinforcement learning, Double DQN, Super Mario bros., Reward Shaping, Computer Vision, Exploration vs. Exploitation, Multi-Agent Learning.**

I. Introduction

The application of reinforcement learning (RL) to classic video games has served as a testing ground for the model, utilizing behavioral understanding in artificial intelligence. Classic video games, such as Super Mario Bros., take place in a tightly bound environment with clearly defined rules, visual cues, and objectives, making the game an ideal testing ground for modeling a decision-making agent. This project investigates how a progressively refined RL agent, based on the Double Deep Q-Network (Double DQN), can be trained to solve an early-stage Super Mario Bros level using various strategies, incorporating traditional reward-based approaches, heuristic punishments, and computer vision.

The approach applied in this paper is iterative, beginning with a simple Double DQN model that encourages movement towards the goal while penalizing failure. Subsequent models build upon this simple model by incorporating in reward shaping, integrating computer vision as an additional parameter through simple template matching, and ultimately adjusting the rewards based on the agent's interactions with the emphasized object detection. Though all models were trained in the same core environment, each model represents the impact that cascading implementations have on the RL agent.

The experiments not only serve to evaluate the effectiveness of specific RL configurations, but also highlight how RL approaches environments that contain both implicit and explicit objectives. As this report will explore, challenges such as balancing exploration and exploitation, emergent gameplay mechanics, and complex systems found in other games. Understanding where current models succeed and fail can help inform the development of more innovative, more adaptable, and impactful agents in more complex domains.

II. Reinforcement Learning Algorithms Applied

Designing an algorithm to complete and solve levels in Super Mario Bros. requires more than just hardcoded behaviour; it requires the ability to adapt, explore, and improve. Reinforcement Learning (RL) provides a framework for teaching agents to make decisions in interactive environments by rewarding it when it succeed at tasks, and penalizing failed tasks. This section provides a high-level overview of reinforcement learning (RL), followed by a focus on the Deep Q-Network and its enhanced variant, Double DQN, which serves as the foundation for this project.

A. *Reinforcement Learning Overview*

In an RL setup, an agent interacts with an environment by observing its current state, choosing an action, receiving a reward, and then transitioning into a new state. Over time, the agent learns a strategy that maps states to actions in a way that maximizes its reward; this strategy is referred to as a policy. This process is typically modeled as a Markov Decision Process (MDP)[1], which comprises a set of states, a set of actions A, a reward function R, and a discount factor (γ) that determines the agent's valuation of future rewards.

Q-Learning is one of the fundamental RL algorithms, which aims to learn the optimal action-value function Q(s, a), representing the expected return of taking action in state s and following the best policy thereafter. However, for complex environments with high-dimensional inputs, such as pixel-based game screens, traditional Q-learning becomes infeasible due to the massive number of possible states.

$$MDP = (S, A, R, \gamma)$$

Defines a Markov Decision Process (MDP) composed of states, actions, a reward function, and a discount factor for future rewards.

B. *Deep Q Networks and Epsilon-Greedy Exploration*

To address this scalability issue, Deep Q-Networks (DQNs) use deep neural networks to approximate the Q-function. Instead of storing Q-values in a table, the agent trains a convolutional neural network (CNN) to predict Q-values from a visual input. DQNs enable the agent to handle raw pixel data from Super Mario Bros., as each frame of gameplay is a high-dimensional image.

$$Q(s, a) = E[r + \gamma \cdot max_{a}{}' Q(s', a')]$$

The Bellman equation describes the expected return for taking an action in a given state and following the optimal policy thereafter.

A crucial aspect of training an RL agent is determining how it explores the environment it interacts with. In this project, epsilon-greedy exploration is utilized: with probability ϵ, the agent selects a random action (exploration); it selects the action with the highest predicted Q-value (exploitation) with probability 1- ϵ. Initially, ϵ is set to a high value, such as 1.0, to encourage exploration. Still, it decays over time to a lower bound (0.1), allowing the agent to exploit its behavior while still having room to explore. This decay process is a crucial parameter to consider. If epsilon decays too quickly, the agent may stop exploring before discovering helpful strategies, but if it decays too slowly, learning may take unnecessarily long.

C. *Double Deep Q-Networks (DQN)*

While DQNs are powerful, they tend to overestimate action values during training. This occurs because the same system used to select the best action is also employed to estimate its value, which can result in overly optimistic value updates and instability in the learning process. Double SQN addresses this issue by decoupling the selection and evaluation process of actions. Instead of relying on a single network, Double DQN maintains two separate networks:

- The online network is used to select the action that appears to have the highest value
- The target network is used to evaluate the value of the action.

$$Q\ \{\{target\}\} = r + \gamma \cdot Q\ \{\{target\}\}(s',\ \arg \backslash max\ a\ Q\ \{\{online\}\}(s',\ a))$$

The Double DQN target update formula separates action selection and evaluation to reduce overestimation bias during training.

During training, whenever a Q-Value is updated for a given state-action pair, the online network picks the best action during the next state, and the target provides the Q-Value of that action. This reduces bias in Q-values and leads to relatively more stable and accurate learning[2]. In this project, Double SQN forms the backbone of the agent in all the models.

III. Models Implemented

The following section evaluates the four reinforcement learning models engineered to play Super Mario Bros. Each model reflects a progressive evolution in agent complexity, state awareness, and behavioural feedback. From

simple rewards to incorporating computer vision for obstacle detection, the goal was not just to solve the first level in a game, but to experiment with how specific changes and strategies improve the RL results. All models were run using the same default parameters listed in Tables 1 and 2, aiming to achieve a result of 3000, which is the reward required for completing a level.

Parameter	**Value**	**Description**
Epsilon_start	1.0	Defines how random the agent is at the start of training.
Epsilon_min	0.4	Controls the lower bound of randomness. Cutoff value since training ends at 60,000 episodes
ϵ Decay Rate	0.99999975	Affects how fast the agent transitions from exploration to exploitation

Table 1, Exploration Parameters. These parameters control the agent's exploration behavior during training. Determines how often the agent tries new actions versus exploiting known successful ones.

Parameter	**Value**	**Description**
Number of Episodes per Model	100,000	Controls how long the agent trains.
Learning Rate	0.00025	Speed and stability of learning
Gamma (Discount factor)	0.9	Weighs future rewards that encourage long-term planning
Batch_size	32	Number of experiences sampled per training step. Affects generalization and update quality
Replay_buffer_capacity	100,000	Max size of experience memory. Controls how much history the agent remembers

Table 2, Training Parameters. These parameters control the learning process of the agent.

A. Model 1 (Basic Reward-driven Agent)

Model 1 serves as the foundation of this process. It operates with a straightforward reward structure. The agent is rewarded for moving right, reaching the goal, and completing the level, and is negatively reinforced if it dies. No vision, memory, or adaptive mechanism is included. Its sole learning objective is to shape a policy that forces its way to the goal.

While this model demonstrates adequate early-stage learning, its limitations become apparent as soon as it encounters certain obstacles, which introduce challenges to its approach. The agent struggles to jump over enemies and obstacles because no feedback is given to the agent. Still, this model is valuable as a baseline to highlight what could be achieved with iterative tweaks to the algorithm.

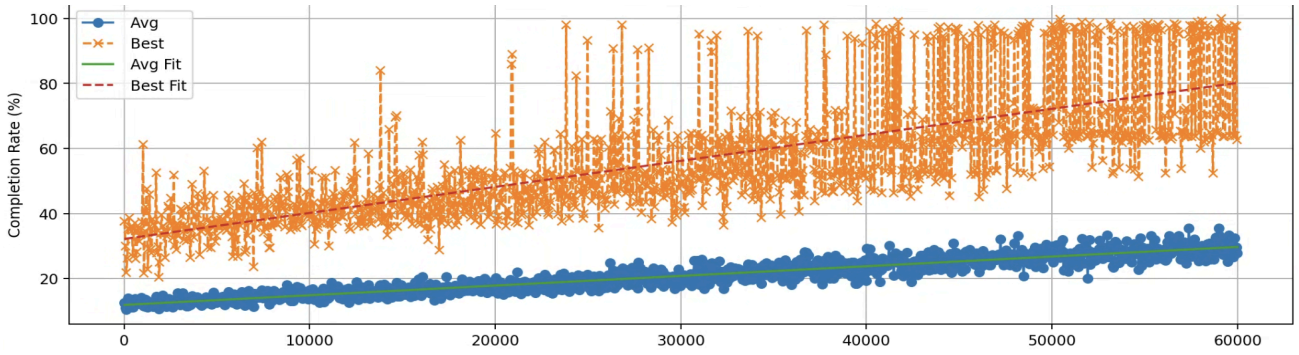

Figure 1, Model 1 results

B. *Model 2 (Stuck Detection & Negative Reinforcement)*

Model 2 builds upon the foundation established by Model 1 (Basic Reward-driven agent) by adding a heuristic to detect when the agent is "stuck". If no meaningful forward progress is made after a defined number of frames, the model is penalized. This modification addresses one of the key issues observed in Model 1, namely the tendency for the model to become trapped in repeating motion loops or to freeze in place when unable to overcome obstacles.

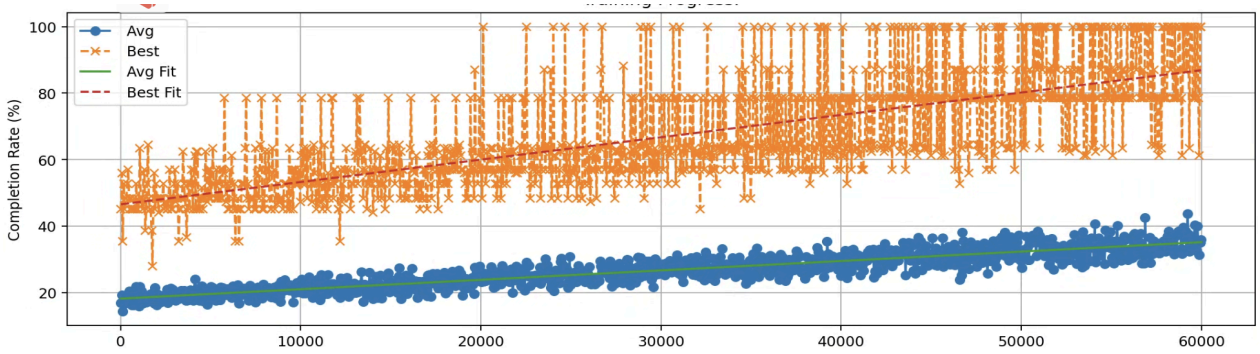

Figure 2, Model 2 results

The stuck-detection mechanism strengthened the agent's awareness of time and positional change, encouraging the agent to try out different actions (e.g, jumping/running) when it hits dead ends. This leads to a higher likelihood of completing the level, especially in episodes where the agent is stuck between obstacles or too afraid to jump over pits and enemies. While still operating within a limited reward space, the model exhibits slightly more exploratory behavior and yields an increased reward rate per episode compared to Model 1.

Despite the improvement, the model remains reactive and lacks a genuine understanding of the game environment. It learns to "panic" and applies whatever strategy shows a positive reward. The decision to jump or change is often made out of desperation, rather than insight. Nevertheless, the model highlights how simple rule-based modifications to reward systems can improve behaviour.

C. *Model 3 (Reward-aware Visual Enemy Detection)*

Model 3 of the reinforcement learning agent introduces a critical enhancement by implementing computer vision (CV) into the reward and observation space. OpenCV template matching was utilized to detect enemies within the rendered frame [3]. These visual cues are predefined through a folder of templates, and each frame is scanned for matches with every other frame. This information informs the logic of what constitutes a meaningful interaction, based on the agent's response to the enemy.

To incentivize smarter choice-making, the agent was not only rewarded for performing a jump action near an enemy, but also had to have its y coordinates surpass the enemy's, then its x coordinates surpass the enemy's, and survive upon landing. This reward pattern ensures that the agent is only rewarded for surviving an interaction with an enemy, eliminating false positives where Mario might jump near an enemy and die mid-air or immediately upon landing. By tying rewards to both action and outcome, the agent is encouraged to develop behaviour that not only reacts to the presence of enemies but actively tries to overcome them.

This dual condition shapes the reinforcement signal to become much more informative, rather than just rewarding jumping, as seen in earlier models. Instead of rewarding isolated actions, successful sequences are rewarded, which is a step closer to modelling high-level skills like obstacle evasion. Over time, this should enhance the agent's tactical performance by enabling it to jump when necessary and taking into account the spatial distance between Mario and enemy locations.

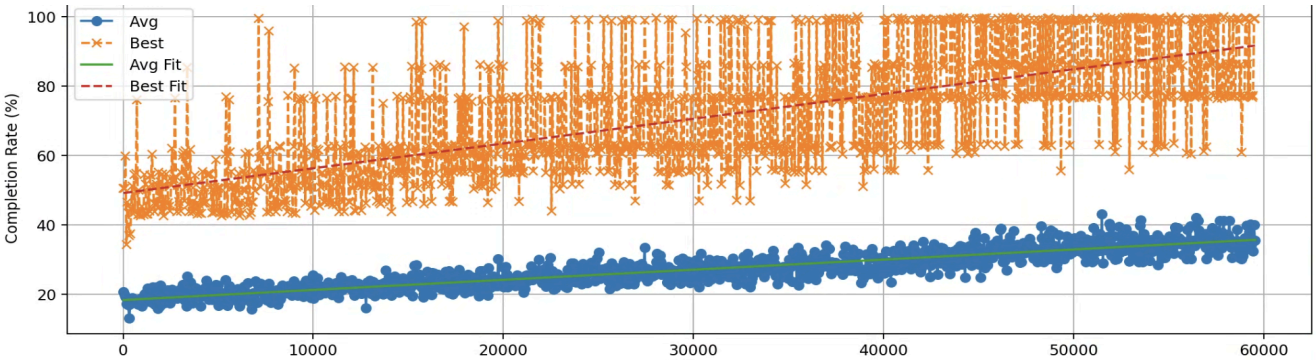

Figure 3, Model 3 results

D. *Model Evaluation*

Model	**Rate of Average Learning (Rewards / Episode)**	**Average completion at ϵ = 0.2**
Model 1 (Basic Reward-driven Agent)	0.022	60.1
Model 2 (Stuck Detection & Negative Reinforcement)	0.024	62.3
Model 3 (Reward-aware Enemy Detection)	0.022	66.8

Table 3, model results

Table 3 summarizes the effectiveness findings of all three reinforcement learning models, based on their average learning rate, the average reward at the minimum decay value, and their progress toward a "perfect" model under the provided conditions. These values were calculated by compressing the rewards from each 50-concurrent episode into an average value, a line of best fit indicating the trendline, and a separate line showcasing the highest reward of each 50 episodes. To evaluate the models, the average reward of $\epsilon = 0.2$ was used to determine a model's performance when randomness is not a significant factor, while still allowing for its potential growth to be a factor.

Model 1, the basic reward-driven agent, and Model 3, the reward-based enemy detection agent, both showcased identical learning rates of 0.022 rewards per episode. However, Model 3 had a head start due to its early reward values, and is closer to its maximum theoretical result average.

While Model 2 exhibits the highest learning rate, its "completion" model reward is only slightly above that of Model 1 and Model 3, highlighting that Model 1 would have reached the same conclusion as Model 2, albeit at a slower pace.

These results demonstrate that different approaches to modelling an RL solution could be geared towards either speeding up training, enhancing final performance, or balancing both. While accelerated learning may not favor

long-term benefits, it is essential to note that it can still be valuable in identifying implementation issues earlier in model training if AI engineers are attentive to the episode outputs.

VI. REINFORCEMENT LEARNING LIMITATIONS

While Reinforcement Learning has seen great success in controlled environments, such as Super Mario Bros., its real-world performance often breaks down under more nuanced or unconstrained conditions, which RL models must adapt to. Many RL models function well in clearly defined reward-rich tasks and environments, but often fail when learning in environments with hidden dynamics, ambiguous feedback, or complex dependencies. Despite being a simple game on the surface, Super Mario Bros. contains hidden mechanics that enable highly optimized gameplay routes. Traditional RL approaches often fail to exploit these nuances because they are unable to interact directly with the game's internal mechanisms.

This section examines several core limitations that emerged during experimentation with the Super Mario Bros. RL agent, as well as how these issues scale up in more complex environments, such as the games “Super Smash Bros.” and “Overwatch”. Each subsection highlights a different dimension of difficulty, ranging from inefficient exploration and feature blindness to abstract skill valuation and multi-agent coordination. These challenges illustrate that even when an agent performs moderately well, it often does so without understanding the deep strategic space of a game. These observations have broader implications for RL developments, especially when the systems are applied to domains where success depends on abstraction, knowledge, foresight, and collaboration.

A. Exploration vs Exploitation

One of the primary challenges with conventional reinforcement learning (RL) in Super Mario Bros. is the sparsity of rewards. In its simplest model (Model 1), the agent receives a reward only when moving right, reaching a flagpole, and is penalized for dying. This creates a highly localized reward gradient that overfits to moving right, often ignoring vertical exploration or alternate paths that do not yield immediate gains. This becomes particularly problematic in levels 1-2, where hidden shortcuts exist and can only be discovered by exploring vertical spaces or uncovering secrets. These are not part of the agent's reward system and would require significant trial-and-error exploration. Given the nature of the greedy exploration aspect implemented in the agent, further exploration becomes statistically rare, especially after the agent has been heavily rewarded for completing the first level.

This demonstrates a key weakness that showcases how exploration vs exploitation trade-offs are environment-biased. If the environment has hidden mechanics and the reward system is naive, the agent will execute based on its suboptimal RL algorithm, which has been rewarded. In Super Mario Bros., this results in predictable, linear gameplay with little curiosity beyond the earlier levels. Unlike humans who might jump around idly and discover secrets out of curiosity, the agent does not have any incentive to behave “creatively” based on its epsilon values once it is far enough into its training. While solutions such as explicit reward shaping or intrinsic motivation could encourage more creative behavior [4], these additional solutions require a deeper integration into the agent’s learning architecture. Moreover, the issue compounds over time due to how agents use experience replay and gradient updates based on the reward history, where bias towards a particular path becomes amplified. This is exacerbated by replay buffer sampling, where the dominance of “safe” trajectories becomes more pronounced than that of more optimal policies, which is an inherent and unavoidable downside of RL agents; the only solution is an increased memory size.

B. Feature Blindness: The Frame Rule Dilemma

Many games and real-world tasks appear simple on the surface, yet they conceal intricate systems that are not immediately apparent to the agent. In Super Mario Bros., players can optimize their movement by manipulating the game system in ways that are never explicitly revealed on-screen. These strategies rely on a deep understanding of the game's inner workings, including collision timing, spawn patterns, and enemy manipulation. Humans have learned these strategies by experimentation, communities, and developer insight. Conversely, most RL agents operate purely based on observations and rewards, without access to the internal mechanics and debug-level information. Without access to resources that humans have, agents may plateau at suboptimal strategies, unable to even recognize that better solutions exist. This key divide between what is visible and what is meaningful makes it difficult for agents to optimize in environments truly. Designers can explicitly expose hidden features through engineered rewards or additional inputs.

An especially revealing example of RL's limitations in Super Mario Bros is the frame rule system. The frame rule system is a hidden timing mechanic that checks level completion every 0.35 seconds. This means that if an agent finishes a level slightly faster, it may experience the same delay as a somewhat less optimal solution. For an RL agent, it would struggle to decide which solution is more optimal because the system rewards both equally. Because this timing quirk has no visible indicator within the game, an agent would have a significantly harder time detecting it.

C. Abstract Skill Discovery

In contrast to the linear and goal-directed design of Super Mario Bros., multiplayer games present a complex gameplay system where the value of an action is often abstract and context-dependent. In the case of Super Smash Bros. Melee, a multiplayer fighting game, specific techniques are present that are not part of the intended game mechanics, nor do they yield any direct in-game rewards. A prominent technique that is both unintended in design and abstract, and is employed at the pinnacle of optimized play sessions, is "wavedashing". Wavedashing is essential for metrics that are not easily measurable, such as spacing and mind games. An RL agent only trained on win/loss signals or damage dealt would struggle to assign appropriate values unless explicitly and deeply embedded in the reward structure.

Unlike in Super Mario Bros, where progress is visibly mapped to objective rewards, Super Smash Bros. Melee requires the agent to understand why something is valuable, not just what directly produces a reward. As mentioned earlier, techniques do not always correlate with measurable values, creating ambiguity. This kind of skill challenges traditional RL algorithms because their value is deeply tied to psychological pressure and probabilistic outcomes, which are not strategies that RLs traditionally encode.

D. The Scale Challenge: Multi-Agent RLs

Where Super Mario Bros. presents a controlled environment, some games introduce a completely different scale of complexity, which RL models struggle to scale to. Overwatch is a 10-player game that takes place in various environments, with diverse objectives, and features multiple characters, each with their unique options. An optimal decision in Overwatch is not just "good for the player" but often depends on information that other RL agents hold. These interdependencies expand the agent's input space, making single-agent RL approaches ineffective without extensive multi-agent modeling, shared strategies, and opponent modeling. This represents a highly exaggerated version of the same issue encountered in Super Mario Bros., where the transition between simple levels can break an agent's policy if it overfits to the first environment.

Moreover, Overwatch lacks a single dominant path to success as there are no easily quantifiable measurements that can be directly attributed to rewards, and the measurements that can be tied to rewards are heavily context-dependent. Multiple in-game strategies can be employed, but they may not be effective under a slightly different environment or a slight deviation in other agent models. The sheer number of possible state combinations and cooperative dependencies suggests that large-scale training may fail to generalize effectively.

V. Final Remarks

A. Challenges

Developing Model 3 introduced several practical and technical challenges due to the need to incorporate computer vision for enemy detection and recognition. Template matching using OpenCV proved to be time-consuming because several templates were not recognized. While the core concept of using visual templates to identify enemies by matching parts on the screen is straightforward, its execution proved to be more challenging. The challenges stemmed from the inconsistencies in resolution, compression, and in-game noise. Resizing the templates consumed a sizable amount of time, as did toggling between grayscale and color input, and applying a Gaussian blur to mimic the in-game noise of the games. These modifications to the CV functionality proved effective for specific templates but began to fail for previously identifiable templates. Only the most necessary templates were prioritized for functionality, and those became the foundation for effective detection. The remaining templates are still present in the templates folder and could prove helpful if the need to evolve the RL agent arises.

Debugging also became more difficult during Model 3 due to the integration of OpenCV. The CV implementation required the display parameter to be set to false. With the display not being present, progress could not be visibly observed, and debugging had to involve a flood of print statements to infer what is going on

behind the scenes. This slowed down the iteration and made it harder to isolate and resolve bugs. Despite these setbacks, the key components of the CV model are operational, and the groundwork has been laid down for future enhancement.

Another major challenge was the duration of model training. Each agent required nearly 24 hours to reach a reasonable exploration-exploitation threshold, with epsilon decaying to around 0.4 throughout 60,000 episodes. Attempting to parallelize the training process through multithreading seemed like a viable way to optimize runtime, but it ultimately did not yield the desired results. The large memory footprint of each model and the replay buffer capacity caused RAM exhaustion, resulting in the system crashing before any of the models made significant progress in their learning. This forced a return to serial and individual training.

B. Future Improvements

I. Combined Agents:

While this project focused on training the agent to complete a specific Super Mario Bros level efficiently, the current approach could be extended into a modular framework. Rather than training a single agent to handle all levels, future work could train one agent per level. This would allow each agent to overfit to its respective level without worrying about affecting its performance in later levels within different environments. A simple level-to-agent lookup function could be used to load the corresponding model to the level being entered, effectively making a larger and more capable system.

II. Enemy Logic:

Another area that could use some improvement lies within how the model interacts with enemies. The current implementation primarily focuses on enemies encountered on early levels and lacks additional logic to handle later enemies. Adding other enemies could enrich the agent's situational awareness. More importantly, an enemy-specific reward mechanism could be implemented. In the case of the enemy "Koopa", it could employ the same jumping over strategy as it would for a goomba, but landing directly on it provides a shell that can be used offensively. Enemies like the "Hammer Bros" introduce project-based attacks, which may require Mario to wait for an opportunity, duck, or backtrack, actions for which the model currently has no reward system.

III. Limited Controls

The current agent is heavily incentivized to sprint forward at all times, and while this strategy is viable for early levels, it might not be so for later levels. Later levels introduce platforming segments, certain enemies, and hazards that require more nuanced navigation. The most theorized optimal strategies for playing Super Mario Bros require specific actions, which are currently disincentivized in the current model. Enabling backward movement and other actions could facilitate more intelligent decision-making. Future versions of the model could support more action spaces.

C. Conclusion

This project explored the application of reinforcement learning through the lens of the classic, yet deceptively complex game known as Super Mario Bros. By iteratively developing three distinct models, each building upon the previous one, this report evaluated how reward shaping, heuristic logic, and computer vision impact model performance. Through this progress, it became evident that even simple changes to the reward structure or input processing can have a substantial positive impact on the learning trajectory of an RL agent.

Despite the improvements seen in each successive model, several challenges were encountered. Most of these challenges centered on reward sparsity and environment overfitting, underscoring the current limitations of RL in dynamic game environments. Nonetheless, the work done in this project lays the foundation for more robust frameworks. The development of enemy-specific logic, modular agent assignment, and expanded action spaces could provide promising results for future developments. As RL systems evolve, so will the complexity of the environments in which they operate. Classic games like Super Mario Bros., with their mix of predictable rules and challenges, remain a suitable testing ground for pushing the boundaries of Reinforcement Learning.

V. Acknowledgements

The base environment and code structure for this project were adapted from the open-source repository **Super-Mario-Bros-RL by Sourish07**. I would like to thank the original author for providing a solid framework that enabled deeper experimentation with reinforcement learning, reward shaping, and computer vision.

VI. References

- [1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.
- [2] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," *Proc. AAAI Conf. Artif. Intell.*, vol. 30, no. 1, 2016.
- [3] G. Bradski, "The OpenCV library," *Dr. Dobb's Journal of Software Tools*, 2000.
- [4] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proc. 16th Int. Conf. Mach. Learn.*, 1999, pp. 278–287.