Training an Agent to Play Mario Kart and Pong Using Deep Reinforcement Learning

Reginald McLean
Department of Computer Science
Ryerson University
Toronto, Canada
reginald.mclean@ryerson.ca

Bartosz Kosakowski
Department of Computer Science
Ryerson University
Toronto, Canada
bkosakowski@ryerson.ca

Albina Cako
Department of Biomedical Engineering
Ryerson University
Toronto, Canada
albina.cako@ryerson.ca

***Abstract*—This is the final report for CP8319: Reinforcement Learning at Ryerson University, Winter 2021. This report outlines the authors' attempt at training a reinforcement learning agent for playing Mario Kart, the deficiencies in the approach, and some areas for improvements. The authors then implemented Deep Q-Learning and Double Deep Q-Learning for Pong, where it was found that Double Deep Q-Learning performed better due to more accurate Q-Value estimations compared to Deep Q-Learning. The code can be found here: [1].**

I. Introduction

The code is in the repo here [1].

Super Mario Kart is a game that was developed and published by Nintendo for their console, the Super Nintendo Entertainment System (SNES) [2]. It was released in 1992 and it is one of the most popular SNES games in history. In addition, it has sparked the development of an entire franchise of games that continue to be released to this day. Super Mario Kart is a racing game where the player can play as one of eight characters from the Mario series of video games. In the single player mode, the player can select one of three difficulty levels and compete against computer controlled opponents. During the race, there are offensive powers that allow the players to give themselves an edge by impeding their opponents or by means to boost their speed [2].

This project intends to train a Super Mario Kart agent to play the game. This problem was selected due to the time constraint on this project, as the Mario Kart agent has very few actions to learn (mainly drive and turn). In addition, the agent's performance can be measured simply by the agent reaching the end of the game. Besides this, all the members of the team were fond of this game in their childhood, which was another reason why this game was chosen. However, as the project went along, the team ran out of GPU memory and was unable to complete the training of the Mario Kart agent. Given the time constraints and lack of adequate results, the main focus became to teach an agent to play the game Pong instead.

Pong is one of the first computer games created. It is a game similar to "tennis", where there are two players each with a paddle, which they use to reflect an incoming ball from reaching their side of the screen. Every time the player fails to reflect the ball, the opposite player gains a point. A total of 21 points is required to win the game [3]. The game Pong was chosen as it requires less memory and time to train, thus allowing the team to train it using different algorithms and compare their performance. Two deep reinforcement learning algorithms were implemented and studied: Deep Q Learning (DQN) and Double Deep Q Learning (DDQN).

There were several lessons learned from this project, one of them was the general approach in the solutions for these types of tasks. Intuitively, it was assumed that every frame would need to be processed and used as part of the training, however further research into the topic clarified that such an approach is unnecessary and very cumbersome. Only every few frames (e.g., every fourth frame) would need to be captured and parsed in order to allow for the agent to process the environment and learn, with highly effective results. Another lesson was the machinations of the algorithms themselves, as they were quite novel and not something the authors had studied in the past. Once the transition to Pong was made, this project also provided insight into an additional deep reinforcement learning algorithm, namely Double Deep Q-Learning. It is also important that prior to committing to a project, the team has enough GPU memory and time to complete a certain project.

Since the project began with its focus on Mario Kart, which then shifted to Pong, each section has a subsection to describe the objective with respect to these games individually. The body of the report begins with a problem statement that elaborates on the further considerations for developing the model. Furthermore, the report discusses the results for each project in the Results section. Next, the Discussion section provides more detail about what was learned from this project from each game, as well as possible future improvements on the Pong game. Lastly, there is a brief summary of the language and libraries used for the implementation.

II. Problem Statement

A. Mario Kart

Mario Kart is a game where the user races other characters around a Mario character-themed map. In this race, the user has multiple opportunities to hit blocks that provide them with an item that can be used to gain an advantage over the other racers in the game. There are two modes of gameplay in this game: a grand prix and a time trial. In the grand prix the user competes in a series of races to accumulate points to win the

cup. In the time trial the user races against the previous best time to set records for any of the available tracks.

There are many concepts in Mario Kart that can be decomposed and made sense of almost instantaneously with very little thought. For instance, when a player sees a box with a question mark, it is understood that it contains some item that can provide assistance to the player to potentially improve their position within a race. Knowing this fact, they drive towards it and anticipate the reward within. However, with some experience, one learns that there are deceptive item boxes that harm the player and cause their cart to temporarily stall out; these are distinguished from the beneficial boxes due to their slight colour difference and that the question mark on their faces are upside down. With this knowledge, within a fraction of a second, a player can usually identify whether the box ahead will provide help or harm, and make a decision to crash into it to get the reward or to avoid it. This is one small interaction of many that make up the plethora or rules and nuances within the game that can be very challenging for an agent to learn. Suppose the example is taken a step further and the agent learns to differentiate between the two types of boxes and can successfully navigate into the beneficial item boxes; it must now understand that it possesses an item and the rules associated with that. For instance, there is a green turtle shell that can be rewarded which can be used as a simple projectile that travels in the direction that the user aims and releases it. If it hits another racer, it disappears and the target is stalled for a few seconds, otherwise it ricochets off of barriers until it hits someone else or it makes several bounces. This simple item brings a complicated strategy with it: the wielder can choose to target another racer directly, or predict their future location and bounce it off of surfaces. They must also consider the possibility that if their aim holds false, then their throw may result in the shell ricocheting back into themselves, either immediately or after it bounces a few times. In the background, the race continues and the player must be cognizant of other racers with other items, whether their position is faltering or not, and so on. The utilization of such an item involves a complicated series of predictions that to a human player is calculated without thought and almost instantaneously. Knowing that this is only a small fraction of the game, it is plain to see the challenge an agent faces. In order to simply become human-level, let alone consistently attaining first place, there are a gargantuan amount of rules and subtleties that have to be learned.

However, some of these concepts are not necessarily essential to success, but are indeed a great aid. Racers are not required to collect and use items, they are merely tools to assist them. Given this, a potential area to streamline the complexity of learning to play the game is to ignore some of these rules. For instance, an agent could learn to ignore the item boxes by acting as if they are not there to begin with regardless of whether it is helpful or harmful. The efficacy of such a strategy may seem dubious, but consider this: the harmful boxes are intended to confuse racers who are targeting the helpful boxes. If a racer is not targeting any helpful boxes to begin with, then the effectiveness of the harmful boxes is reduced to a racer crashing into it by chance rather than intent. This approach would require careful consideration of which aspects to ignore and which aspects to keep; an aspect might be deemed as not necessary may prove to be very important. These changes can be reflected by modifying the action space and rewards. For example, to prevent the agent from learning to use items, the action space would consist of every other button press except for the one that throws items and the reward would not factor in successfully hitting an opponent with an item.

B. Pong

Pong is a simpler game than Mario Kart. The main problem facing the agent is to learn how to maneuver the paddle up, down, or keep its position. Also, the agent has to predict the next location of the ball. The agent does not necessarily need to know where its opponent is positioned; as long as it can at least keep returning balls that the opponent sends, it is in a good position (since maintaining a tie means that it is not losing).

Nonetheless, it is possible to devise a strategy to improve the odds of success in this game, beyond simply positioning the paddle in a way to return the ball to the opponent and hoping that they miss. As described in [4], the paddle is divided into sections that determine the angle in which the ball will return to the opponent. This can be leveraged in order to tactically return the ball far away from where the opponent is presently located, thus causing them to be in an unfavourable position. The game contains four zones, one in each corner of the screen, where it is impossible for the opponent to return the ball, so the agent may potentially pick up on a strategy where they can combine these factors into a highly effective technique. However, this is not a requirement for success, so it may be the case that the agent does not develop any strategy in particular besides the most basic one, which is to return the opponent's ball.

III. Environment

A. Mario Kart

For the Mario Kart project, the gym-retro library was used [5]. This environment comes pre-loaded with nearly 1000 games. Depending on the copyright protection, some of the games were included in the library and had their states, reward, actions and environment already predefined and ready to use. However, this was not the case with Mario Kart. The manual labour involved with setting up this data was obtained from [6]. This repository provided us with a few sample environments, the states, and a defined reward function.

The sample environments included three race tracks where the racing mode was set to "time trial", which means that the agent does not face any obstacles (besides the barriers defined within the race track), opponents, nor any items to pick up. The gym-retro library comes packaged with an integration tool that allows for users to select a ROM of a game and create environments and states by saving information directly from the game [5]. It also allows for users to search through

Variable	Definition
Lap	The current lap of the race
Position	Denotes what section of the track the user is in
Overall Speed	Speed of the agent

TABLE I
INTERNAL VARIABLES OF MARIO KART FOR REWARD FUNCTION

the list of variables within the game, as it loads, to identify key variables for the agent (such as the lap number) [5]. For instance, the user can select a specific race track, character, and game mode and export it as the environment.

$$Reward = (L * 1000) + (P * 10) + (O/10) \tag{1}$$

The reward function was devised by [6] and is defined in equation 1. It consists of the game variables that are defined in table 1. L is the lap, P is the position, and O is the overall speed. The first term of the summation is the weighted value of the lap. This involves first subtracting 127 from the lap number, since the game was designed such that the starting value is 128 instead of 0. Then, the value is multiplied by 1000 to incentivize the agent to try to complete laps. The second term is the weighted position, which incentives the agent to progress in the tracks, since a higher P value means the agent is on a further section of the track, with the finish line being in the last section. Lastly, is the weighted speed term, which is divided by 100 to reduce the size of this portion of the reward since it can be a high value (from 0 to 1000).

The state for Mario Kart was the current frame that the model observed. This would be processed by the network and its information would be extracted to provide some context for the agent about relevant information, such as the track boundaries.

The actions are simply representations of the possible inputs that are available on the SNES controller [7].

The final action space chosen to train Mario Kart is a subset of the possible inputs, because some of the inputs are not relevant for the agent to learn to play the game. The removed actions are START, SELECT and X button inputs. This left nine actions for the agent to learn.

B. Pong

The Pong environment comes predefined and loaded within the Gym library, so it can be simply accessed by specifying which environment to load at the start of the code. Pong contains only two important factors that the agent must consider when an action is selected: the position of the ball and the position of its paddle. By contrast, in Mario Kart there are many important aspects such as the agent's position in the race relative to other racers, locations of item boxes within the track, speed boosts on the ground for the agent to gain an edge, the shape of the track ahead of the agent, and so on. Furthermore, the rewards map directly to the game; the agent is rewarded for scoring a point and is penalized for having a point scored against it.

$$Reward = agentScore - opponentScore \tag{2}$$

The reward function for the Pong agent was simple, and is outlined in equation 2. The worst possible score for the agent is -21, since that is the most amount of points that can be scored before the game ends. Therefore, the best possible score is +21, in which case the agent does not let any points be scored against it. Given this, as the average score increases over time, the agent's progress can be very clearly interpreted.

As with Mario Kart, the state for the Pong implementation is simply the game frame that the model is currently observing. From this, the network can determine important information such as the ball's position and its paddle position.

The action space for the Pong agent is very small, and maps to the Atari 2600 controller as the Mario Kart action space maps to the SNES controller. The actions that were available are as follows: no operation, activate the "fire" button, move joystick left, move joystick right, move joystick left and fire simultaneously, and move joystick right and fire simultaneously.

IV. METHODS AND MODELS

A. Mario Kart

The solution that was pursued was to implement a Deep Q-Learning Network (DQN). The DQN reward function is defined in equation 3. A Deep Q-Learning algorithm uses two neural networks. It maps a state into (action, q-value) pairs [8].

$$Q(s_t, a_t) = r_t + \gamma \max_a Q'(s_{t+1}, a_{t+1}) \tag{3}$$

This approach had previously been successful for other games [8] and so became the target algorithm to implement. Preprocessing the input involved converting the image to grayscale; this would reduce the computations required by the network, as processing the raw colour image would be an intensive process. Furthermore, the detail at full colour is not necessary [8].

To measure the success of the agent we chose to compare how long it takes for the agent to complete the track with the authors' times. The time the agent takes to complete the track was selected because it also indicates the agent's performance level; the more time the agent spends on the track, the worse the agent performs. The agent's performance with respect to time can indicate improving control of the kart on the track.

The reward function has room for improvement, as there is no penalty to disincentive the agent from performing actions that are not conducive to its progress. For example, if it is turned around and racing in the opposite direction, its reward can increase even though this would cause it to lose. The game provides indicators that the player is moving in the wrong direction, which means that there are variables defined within the game's code that differentiate between correct and incorrect direction. Therefore, this could be leveraged to penalize the agent if it attempts to go in the wrong direction.

B. Pong

This implementation consisted of two approaches. The first approach used was the same as for Mario Kart, using DQN to train the agent to learn the game. This was selected because the approach has been thoroughly demonstrated to excel at learning how to play many Atari games, in some cases even better than expert players [8]. In particular, this game is one where the agent learns to play extremely well using DQN, therefore using this algorithm was the first solution chosen.

The other implementation followed the Double DQN (DDQN) approach. The benefit of DDQN is that it is more robust against maximization bias than the normal DQN mode [9]. The difference in the DDQN from the original DQN is in the implementation of the loss function. In DDQN, the best action to take in the next state is calculated using the main train network, however, the values corresponding to the action come from the target network [9]. The purpose of the DDQN is to reduce the chance of DQN overestimating values of Q, which can harm the training process. The DQN reward function is defined in equation 4.

$$Q(s_t, a_t) = r_t + \gamma \max_a Q'(s_{t+1}, \text{argmax}_a Q(s_{t+1}, a)) \quad (4)$$

The metrics used to assess the capabilities of the Pong agent was its average score difference. This refers to the difference between the agent's score when the game ends and its opponent's score. For a game such as Pong, where the objective is to score points against an opponent, this is an intuitive metric. The agent was trained three times for both DQN and DDQN. In this training setup the agent was trained using randomly initialized weights for the first experiment, and then subsequent experiments used the final weights from the previous experiment.

V. Results

A. Mario Kart

With limited remaining time to complete the Mario Kart implementation, there were few opportunities to tune the code for Mario Kart. Specifically there was a problem with loading the replay buffer into memory where a CUDA out of memory error was thrown. At this point, due to time constraints, Pong became the focus of the project.

B. Pong

The pong project demonstrated much more successful results. Both the DQN and DDQN led to an agent that learned how to successfully play the game by continually improving its score. The results for DQN can be seen in Figure 1, 2, and 3. As seen in these figures, the agent score improved over time. The training of the agent generated a final average reward of approximately -11.48, -1.7, and 4.18 for run 1, 2 and 3, respectively. The DDQN results can be seen in Figure 4, 5, 6. The average scores the agent obtained were around -11.22, -2.54 and 7.84 for run 1, 2 and 3, respectively.

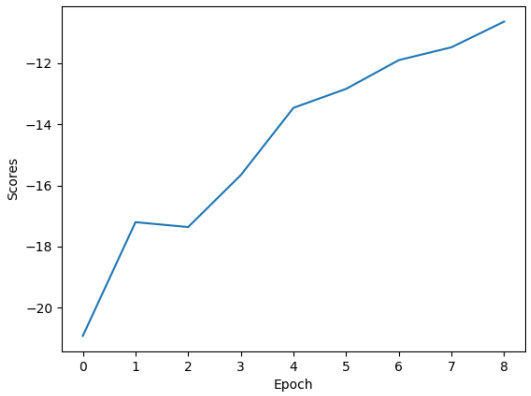

Fig. 1. DQN Run 1

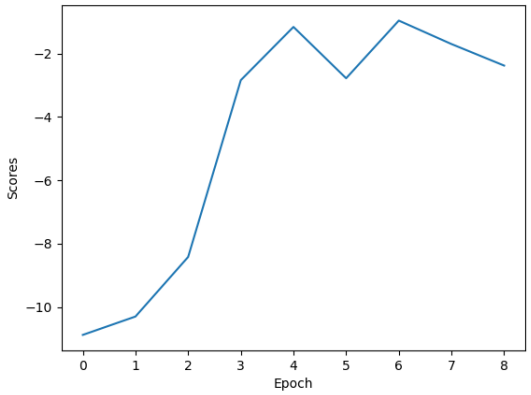

Fig. 2. DQN Run 2

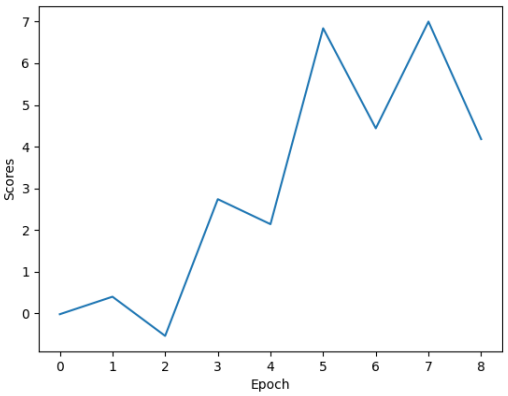

Fig. 3. DQN Run 3

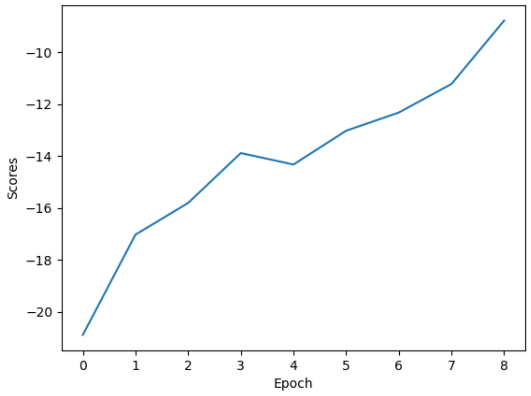

Fig. 4. DDQN Run 1

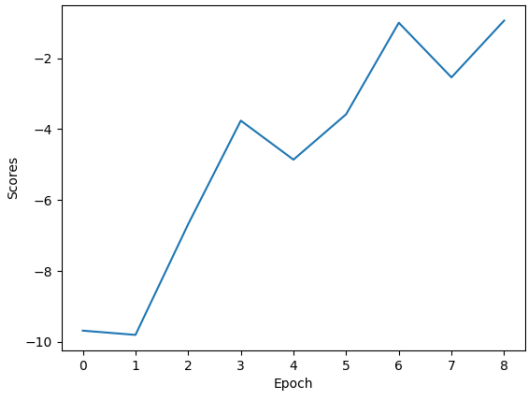

Fig. 5. DDQN Run 2

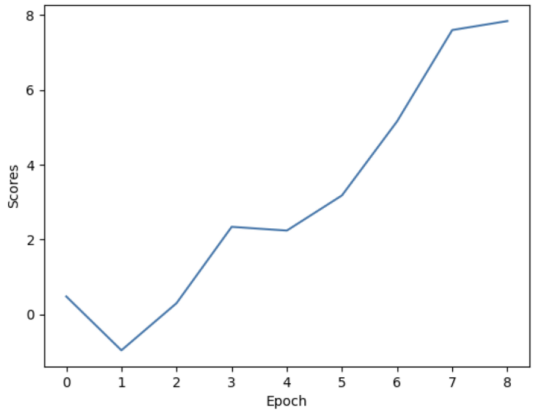

Fig. 6. DDQN Run 3

VI. Discussion

A. Mario Kart

One of the factors for the failure in the Mario Kart project was that the model that was selected could not reflect the complexity of the game. There were three approaches that were followed: the model from [10], the model from [6], and the model from [8]. The first two models did not provide any insight into their success, so their initial capabilities were unknown until their re-implementation was completed. However, once the agent was running it was difficult to know whether there was a fault in the re-implementation or whether the original code for both models performed poorly. There were adjustments made to some of the model's parameters in order to improve its learning ability, such as by adjusting the number of steps within an episode from 500 to 50,000. This is because 500 time steps within the game was too little time for the agent to figure out complex behaviour. For instance, 500 time steps may be the initial empty stretch of a track where the agent simply has to accelerate forward to succeed. Given the complexity of Mario Kart, this would not lead to a successful model if it was then placed onto a different track for a longer period of time. Despite this, the agent's success seemed to have plateaued, which led to the pursuit of [8]. This third approach had its own problems, the most challenging of which was the lack of available resources for the model to use. In particular, the system kept running out of memory on the graphics card (GPU) and would crash the program. The reason for this is potentially because the replay memory becomes too large for the GPU memory to store all of the data. Further investigation could be taken to see if there is a way to use a different approach for the memory buffer. After a number of iterations the memory buffer could be updated with new replays instead of pushing all of the data at once.

There were several attempts made to solve this problem: there was an additional stopping criteria added so that if the agent did not reach the in-game stopping criteria (i.e., reach the finish line after three laps), the environment would reset. Specifically, the agent was given two minutes to complete the three laps within an episode, which was approximately 15,000 time steps. The number of episodes was also changed to 50. The starting epsilon value was reduced to 0.1 in order to attempt to have the model reach the training stage. Furthermore, the batch size was reduced so that there was less data to fit onto the GPU memory. Another potential solution was to move the model to a different GPU that has more memory, but this did not resolve the issue. Additionally, there were efforts made to split the model across multiple GPU's but this resulted in different errors. In the end, time constraints prevented any further investigation into these faults, and the project shifted focus to teach an agent how to play Pong instead.

Given this, it is challenging to properly assess what can be done in future experiments. Had there been enough GPU memory and time to focus on the project, it would have been possible to study how the previous changes impacted the result

of the model. In this case, potential future experiments can be repetitions of these previous ones. Additionally, it would be interesting to see how changes to the reward function impact the agent's capabilities. At the latest iteration of the project, the reward function from [6] was used. One observation that was made was that there was no negative reward for undesired behaviours, as we can see in equation 1. For instance, if the agent were to be turned around and head in the wrong direction, then there would be no consequence for this, despite it being the exact opposite of the desired behaviour. Perhaps by factoring in a negative reward for such behaviour and adjusting the weight of the other factors in the reward function, the performance of the agent may be improved. Another future change could be to accommodate more frames. The approach that was followed looked at every four frames, but given the complexity of Mario Kart, it may be the case that more frames should be looked at. Lastly, an additional preprocessing step could have been added to shrink the image down to some smaller scale. This could have further improved the performance of the model and may have mitigated some of the memory issues. This could be pursued in future attempts.

B. Pong

The Pong agent was successfully trained with DQN and DDQN. The three different training stages showed improvement in the agent's performance over time. For the DQN model, the agent's average score between the first and second training sessions improved by 10.5 points. From the second training to the third (final) training session, the agent's score improved by 8 points. Furthermore, the DDQN model performed slightly better than the DQN model. The agent's average score improved by 8.5 points between the first and second training and by an average of 8.5 points from the second to third training. Generally, this agent also scored higher after each training session than the agent trained with the DQN model. The agent overall achieved a higher average score of 8 points in the DDQN model, beating the DQN agent's maximum average score of 7 points. The results show that DDQN is a better model to train the Pong game. Compared to human performance stated as an average of -3 [8], the agents trained with DQN and DDQN beat this score by 10 and 11 points, respectively. In addition, this showed that the approach used in this paper to train the agent on the previous training weights, was the correct approach to improve the agent's performance.

Comparing the DDQN results in Figures 4, 5, 6 to the DQN results in Figures 1, 2, 3, the DDQN has a much smoother rate of progression. This is because DDQN estimates the Q values more accurately than DQN, which overestimates the Q values. This overestimation leads to a worse generalization than DDQN.

To improve the agent's score, future projects can involve longer training time or further training sessions. As was seen, the more training sessions that were performed, the agent's score improved by 8 - 10.5 points. Thus, increasing training time could increase the performance of the agent. In addition, other algorithms could be used to train the model to achieve higher performance.

VII. Implementation

The implementations of both projects contained a lot of overlap; both projects were implemented using Python 3. The main libraries used were Pytorch, OpenAI Gym, NumPy, pyglet, Keras and Matplotlib. The DQN algorithm was implemented using both the third assignment and course notes from the Reinforcement Learning course at Ryerson University. The DDQN algorithm was coded following the instruction in this book [11]. Since the agent could not learn to play Mario Kart sufficiently well, nor could it do so in a timely manner, the original implementation of the algorithm for this game was not included with this report. However, the code for the Pong algorithm was included.

VIII. Conclusion

In conclusion, we attempted to train a Deep Q-Learning algorithm to play Mario Kart. This involved integrating the game into the gym retro environment using the included integration tool, defining the reward function, as well as the initial states. This attempt was unsuccessful due to time and hardware constraints. Further work could be done to investigate these limitations around the replay memory buffer. The next attempt was to use the same Deep Q-Learning algorithm to train an agent to play Pong. In addition to the traditional algorithm, Double Deep Q-Learning was also implemented and experimented with. In these experiments it was found that the Double Deep Q-Learning algorithm performed better when compared to traditional Deep Q-Learning as the Double variant reduces Q-Value overestimation which leads to better generalization performance. Further work could be done to explore the performance of other Deep Q-Learning algorithms on this task in comparison to the original algorithm.

References

[1] [Online]. Available: https://github.com/reginald-mclean/ReinforcementLearningProject

[2] "Nintendo power," 1992.

[3] Pong game. [Online]. Available: https://www.ponggame.org/

[4] D. Ahl. (1974) Playing pong to win. [Online]. Available: https://www.atariarchives.org/bcc1/showpage.php?page=141

[5] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, "Gotta learn fast: A new benchmark for generalization in rl," *arXiv preprint arXiv:1804.03720*, 2018.

[6] J. M. John Compitello, "Mariokartproject," https://github.com/johnc1231/MarioKartProject, 2019.

[7] N. of America, "Super mario kart instruction booklet," 1992.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[9] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[10] J. Han, "Super-mario-rl," https://github.com/jiseongHAN/Super-Mario-RL, 2020.

[11] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt, 2018.