Human level performance of an AI playing Donkey Kong

Jakub Grzywaczewski, Anna Ostrowska,
Igor Rudolf, Marta Szuwarska

June 2024

Abstract

This project explores the challenge of training an AI to play the classic Donkey Kong game. Deep Q-Networks (DQN) and SARSA algorithms were implemented, with initial attempts focused on standard rewards and penalties. Early results were not promising, leading to the introduction of several modifications, including penalties for needless jumping, incentives for climbing ladders and reaching higher levels, and heuristic-based action selection. Despite not fully achieving the primary goal of completing the first level, the AI demonstrated significant progress by navigating complex game elements and outperforming half of our team members. This underscores the potential of reinforcement learning for mastering intricate gameplay mechanics. Further development and optimization could eventually lead to the full completion of the game's objectives.

We strongly encourage one to visit RL Donkey Kong - GitHub,
where all files and information are available.

Keywords: Reinforcement Learning, Donkey Kong, Q-Learning, DQN, SARSA

Contents

1 Introduction

The main goal of our project was to train a reinforcement learning model for the Donkey Kong video game so that the agent would be able to reach the first level of this game. A side goal was for the agent to perform at the human level in this game.
In Donkey Kong, the player takes on the role of Mario and tries to save his girlfriend from a giant gorilla named Donkey Kong. To get to his girlfriend, Mario must climb unbroken ladders and omit the barrels.
The barrels fall between floors in a random and unpredictable pattern; they can fall down one of the ladders or the end edge of a floor.
In addition, if Mario grabs the hammer located on the 4th floor, he can hit the barrels, which are then destroyed. The hammer, however, has a time limit - it lasts 32 swings and then disappears.

We have been trying many techniques to solve this game and are finally sticking to Deep Q-Networks (DQN) and SARSA, which we will describe in the next section of this report. There was a little idea of using standard PPO and learning from human experience. It would be pretty helpful during the final section of beating the game. By these words, we mean the floor where Mario can find the hammer. Despite trying to encourage our agent with many techniques, the performance of this level was relatively not the best. Also, human experience could be helpful, but due to time limits, we were not able to implement this solution. In the end, DQN was quite a natural choice because DQN was originally developed and tested on Atari games, to which Donkey Kong is similar. We could approximate complex Q functions, which is beneficial in complex game environments. We also used SARSA because updates are based on the actual actions of the agent, which can lead to safer learning (less risky actions).

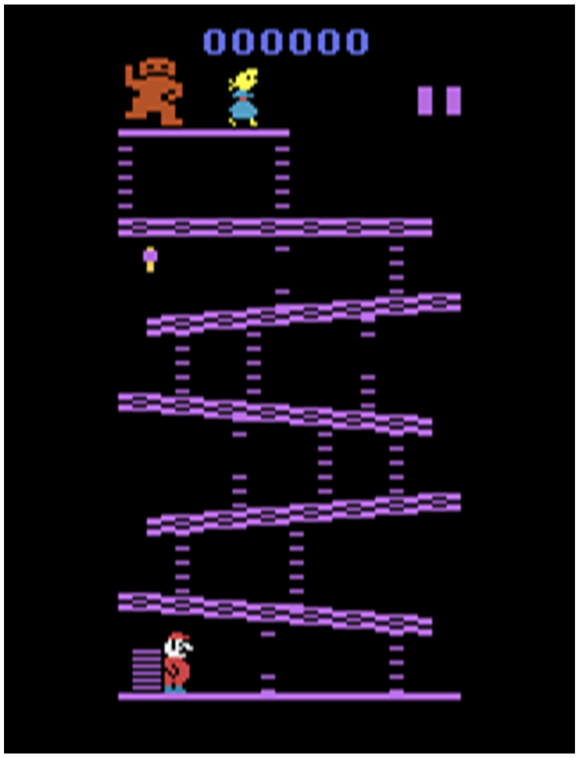
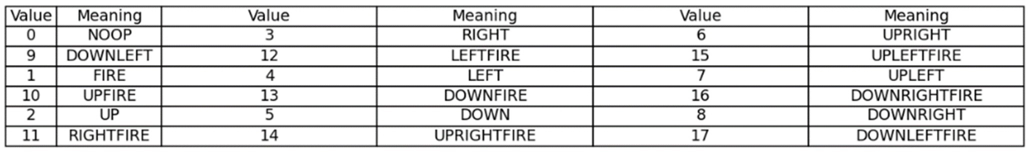


Figure 1: Starting point in Donkey Kong game.

2 Donkey Kong on Gymnasium

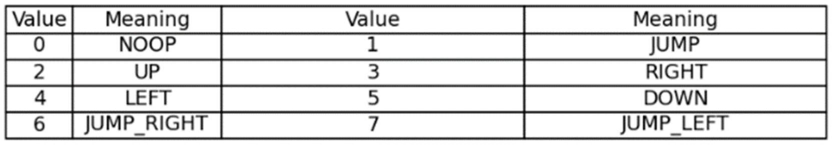
Gymnasium [8] is a maintained fork of OpenAI's Gym [1] library. It is a comprehensive library that is frequently used in the context of Machine Learning and Data Science, allowing users to create, train, and test machine learning algorithms on various simulation environments.
Gymnasium provides a diverse collection of simulation environments (e.g., Atari, MuJoCo, Box2D) to train reinforcement learning algorithms.
Donkey Kong environment is part of the Atari environments: a set of Atari 2600 environments simulated through Stella and the Arcade Learning Environment. Donkey Kong has the action space `Discrete(18)` with the table below listing the meaning of each action's meanings.



Value	Meaning	Value	Meaning	Value	Meaning
0	NOOP	3	RIGHT	6	UPRIGHT
9	DOWNLEFT	12	LEFTFIRE	15	UPLEFTFIRE
1	FIRE	4	LEFT	7	UPLEFT
10	UPFIRE	13	DOWNFIRE	16	DOWNRIGHTFIRE
2	UP	5	DOWN	8	DOWNRIGHT
11	RIGHTFIRE	14	UPRIGHTFIRE	17	DOWNLEFTFIRE

Figure 2: The action space for Donkey Kong game on Gymnasium.

Since some of the actions appeared to do nothing or do the same thing as other actions after launching the game, we reduced the action space to 8 possible actions.



Value	Meaning	Value	Meaning
0	NOOP	1	JUMP
2	UP	3	RIGHT
4	LEFT	5	DOWN
6	JUMP_RIGHT	7	JUMP_LEFT

Figure 3: The reduced list of actions.

We used the game variant `env_id=ALE/DonkeyKong-v5, obs_type=\rgb", observation_space=Box(0, 255, (210, 160, 3), np.uint8)`.

3 Algorithm

To train the AI agent, we have implemented Deep Q-Networks (DQN) [3] and SARSA [7] algorithms. The following sections describe the algorithmic framework of those algorithms, training procedures, and reward modifications that were instrumental in shaping the agent's performance.

3.1 DQN

One of the key algorithms that we have used is DQN, which stands for Deep Q-Networks. It basically combines Q-Learning [7] with deep neural networks. Q-Learning is an off-policy, model-free reinforcement learning algorithm that aims to find the optimal action-selection policy using a Q-value function. The Q-value function $Q(s, a)$ represents the expected return (future rewards) when taking action a in the state s and following the current policy thereafter. Essentially, we aim to estimate the value of following the current policy and iteratively improve it to get closer to the optimal policy.

The true Q-value function can be written as:

$$Q(s,a) = \mathbb{E}[R + \gamma \max_{a'} Q(S', a')],$$

where R and S' are random variables representing the reward and next state provided by the environment given the current state s and action a. Here, the expectation is over the distribution of possible rewards and next states.

In practice, we iteratively update the estimated Q-values. This update rule, used in the Q-Learning algorithm, may be written by us as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right], \tag{1}$$

where:

- s is the current state,
- a is the action taken,
- r is the reward received,
- s' is the next state,
- α is the learning rate,
- γ is the discount factor ($\gamma \in [0,1]$).

We must understand that this update rule is an approximation used in tabular Q-Learning, where the states and actions are discrete and the Q-values can be stored in a table. In the true Q-value function, if we set $\alpha = 1$ and calculate r and s' as expected values, we get:

$$Q(s,a) = \mathbb{E}[R + \gamma \max_{a'} Q(S', a')],$$

which represents the expected value of the reward plus the discounted maximum future reward, given the current state s and action a.

But how is the $Q(s,a)$ calculated? Well, it is approximated using a deep neural network, which is referred to as the Q-network. What we want to do here is, after training the neural network, to minimize the difference between the predicted Q-values and the target Q-values. This can be written as trying to minimize function $L(\theta)$. Below is the exact form of this function

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s,a;\theta) \right)^2 \right], \tag{2}$$

where:

- $L(\theta)$ is the loss function that we aim to minimize during the training of the neural network. It represents the difference between predicted and target Q-values.
- $\mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}$ denotes the expected value (average) over samples (s, a, r, s') drawn from the replay buffer $\mathcal{D}$.
- s is the current state, describing the present situation of the agent in the environment.
- a is the action taken by the agent in state s.
- r is the reward received after taking action a in state s.
- s' is the next state that the agent transitions to after taking action a in state s.
- γ is the discount factor, which determines how much future rewards are valued compared to immediate rewards. γ takes values in the range [0, 1].
- $\max_{a'} Q(s', a'; \theta^-)$ is the maximum Q-value for all possible actions a' in the next state s'. θ^- is the set of parameters of the target network, which is periodically updated to stabilize training.
- $Q(s, a; \theta)$ is the predicted Q-value for the current state s and action a, calculated using the current parameters θ of the neural network.

Annotation: Target network and its parameters θ^- is a copy of the main network and its parameters θ, but target network parameters are updated less frequently (at a certain fixed interval) to stabilize the learning process.

Implementation

Our implementation uses the neural network architecture presented in the *CleanRL* [2] repository and utilizes the *PyTorch* [5] library for tensor operations. In brief, we utilize a convolutional neural network with 3 convolution layers and 2 fully connected layers.

3.2 SARSA

We also tried to use SARSA. SARSA stands for State-Action-Reward-State-Action. Basically, it is just a sequence of next steps made by an algorithm. If we talk about an algorithm, we need to describe how it works. So, SARSA is an on-policy algorithm. It updates its Q-values based on the actions taken by the current policy. At first, we initialize the Q-values for all state-action pairs. Then, we choose an action based on the current policy, for example, the epsilon-greedy one. Epsilon-greedy means that we take a random action with the probability ϵ and with the probability $1 - \epsilon$ the action with the highest Q-value. Then, we observe the reward and go to the next step. In other words, the process continues. The formula for updates looks as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]. \tag{3}$$

Generally, the markings are the same, but additional ones have been introduced:

- α is the learning rate,
- γ is the discount factor,
- s is the current state,
- a is the current action,
- r is the reward received,
- s' is the next state,
- a' is the next action.

There is a very interesting theorem about the convergence of SARSA. Given sufficient exploration (i.e., all state-action pairs are visited infinitely often) and under certain conditions on the learning rate, SARSA is guaranteed to converge to the optimal Q-values. Below is an excerpt from the book: "Reinforcement Learning: An Introduction" by Richard S. Sutton and Andrew G. Barto, which discusses the topic of SARSA convergence.

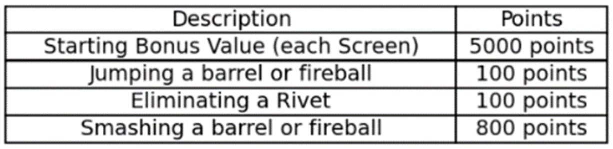
Fragment [6]:

> *The convergence properties of the Sarsa algorithm depend on the nature of the policy's dependence on q. For example, one could use ε-greedy or ε-soft policies. According to Satinder Singh (personal communication), Sarsa converges with probability 1 to an optimal policy and action-value function as long as all state–action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy (which can be arranged, for example, with ε-greedy policies by setting $\varepsilon = 1/t$), but this result has not yet been published in the literature.*

However, **we decided to stick with DQN rather than with SARSA because of DQN's better results**.

3.3 Initial rewards/penalties

The description of the initial rewards is in the table below. Rewards include Bonus Value (which gets lower every second of the game; it is added when Mario reaches his girlfriend for doing it quickly), jumping over a barrel or fireball, eliminating a Rivet(metal pieces that Mario must remove to complete the level), and smashing a barrel with a hammer. However, at the first level, we will not consider Rivets as well as fireballs. Also, we added the penalty for death (-200), which was double the negative reward for jumping over a barrel.



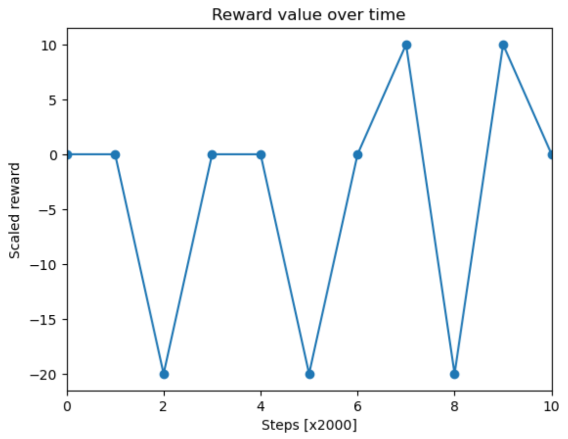
Description	Points
Starting Bonus Value (each Screen)	5000 points
Jumping a barrel or fireball	100 points
Eliminating a Rivet	100 points
Smashing a barrel or fireball	800 points

Figure 4: Initial rewards for actions in Donkey Kong game.

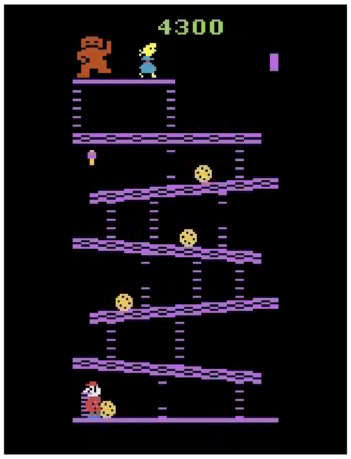
4 First results

Our first algorithm was DQN with initial rewards and an added penalty (negative reward -200) for death. Our first results after training the player on this algorithm were not satisfactory.
After the first 5 hours of training, the agent's actions appeared random. He managed to enter the 2nd floor, but he was walking alternately right and left and did

not jump over barrels.



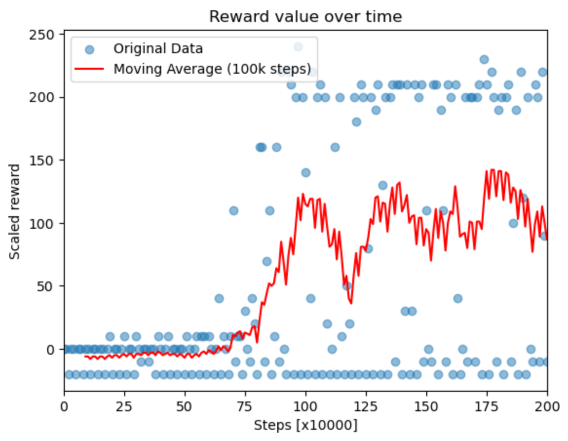
(a) Plot showing the value of reward during first training.



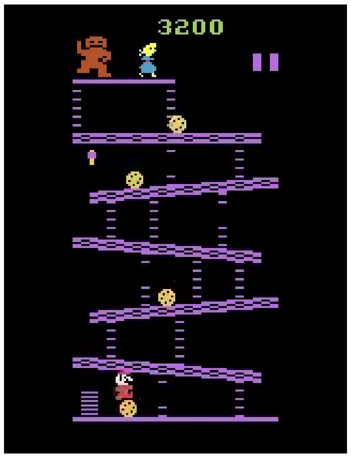
(b) One of the frames from the first simulation.

Figure 5: Results of training the agent with the first algorithm with initial rewards and death punishment.

In the chart, we divided the value of the rewards by 10, so the value of -20 means that Mario died during this period without jumping over any barrels.
With another 5 hours of training on this algorithm, the agent learned to jump over barrels, but he chose jumping as his preferred action - he did not enter higher floors than the ground floor, and rarely moved to the right or left. He took the jumping action all the time, even if there were no barrels in his surroundings. After another 5h, he didn't learn anything else - he continued mostly jumping on the same floor.



(a) Plot showing the value of reward during training with the first algorithm for a longer period of time.



(b) One of the frames from the second simulation.

Figure 6: Results of training the agent with the first algorithm with initial rewards and death punishment for a longer period of time.

As one can see on the chart, Mario, after a while, started getting some significant rewards for jumping over barrels. However, this was not our goal, so we decided to make some changes to the algorithm.

5 Modifications

When wondering why the only thing Mario has learned so far is jumping and how to improve it, it is important to pay attention to the rewards given to him and their value.
Knowing that the available rewards are: Bonus for winning the game quickly, jumping over a barrel, getting to his girlfriend, smashing a barrel with a hammer, and a penalty for dying, it is not hard to see that the only thing Mario is capable of is jumping over the barrels or omitting them in order to avoid penalty for death. If he is unable to reach even the 2nd floor of the board, he will never receive bonus points or rewards for reaching the princess or using the hammer. In the next steps, additional rewards were added to encourage him to achieve higher floors and take actions other than jumping.

5.1 Punishing needless jumping (PNJ)

Since we wanted to encourage the agent to take actions other than jumping and discourage him from needless jumping all the time, we decided to give him a penalty (negative reward) for needless jumping with the value of -20. We found this idea while researching the topic of training reinforcment learning agents for playing DonkeyKong [4]. We added a function that checks if there is a barrel near Mario. Then, we manually found pixels corresponding to ladders and tested whether they actually coincided with the field where Mario can go up. Every time Mario was not near the barrel or close to the ladder, and he jumped up, we gave him a negative reward (-20).

5.2 Ladders incentive (LAD)

Using found pixels corresponding to the ladders, we added a reward (+10) for Mario for moving up a ladder. We also added a higher negative reward (-15) for Mario for moving down the ladders, as we did not want him to go up and down the same ladder just to collect rewards for moving upwards.

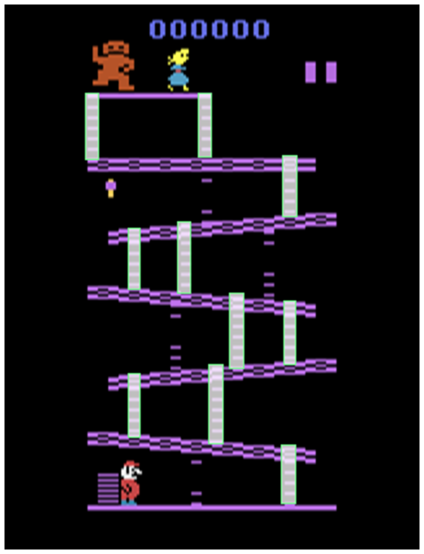


Figure 7: Found pixels that correspond to the ladders.

We hoped this modification would encourage him to move to the higher floors rather than staying on the ground floor and omitting barrels.

5.3 Level incentive (LVL)

For the same reason as adding the ladder incentive modification, we decided to add another reward (+200) for Mario for reaching the higher level (floor). We checked if his current level was higher than the level he was on in the previous position, and if so, we gave him +200 as his reward.

5.4 Magic stars incentive (MS)

In order to encourage Mario to move towards the end of the game, we chose a path that Mario would have to take anyway to reach his girlfriend and marked several points on it (”magic stars”).
The moment Mario reached the area of the star, we gave him a reward (+20) for it and removed the star from the available to get in a given game (so that he does not come back for the stars he has already reached).

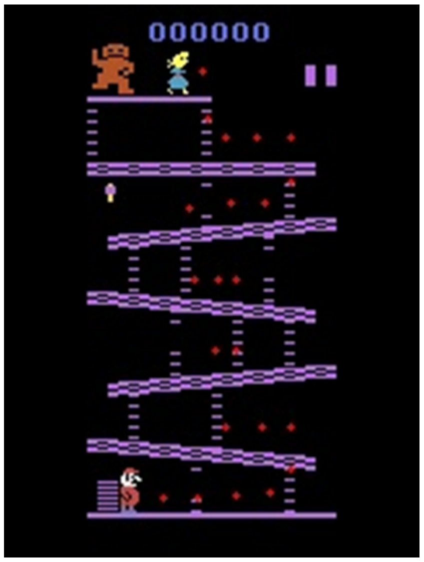


Figure 8: Pixels on which magic stars were placed.

5.5 Action heuristics (AH)

In order to speed up learning, we decided to check whether, after changing the initial probabilities of selecting a given action from random to depending on Mario's position, it would produce any results.
We declared a probability function from which Mario should choose his actions, depending on the floor he was currently on: if it is an even floor number (taking the ground floor as 0), he should go right with the probability of 3/7, up with the probability of 2/7 and go right, down or stay in place with probabilities of 2/21. If he is on an odd floor number, he should go left with the probability of 3/7, right with the probability of 2/21, and the rest stays the same as on even floors.

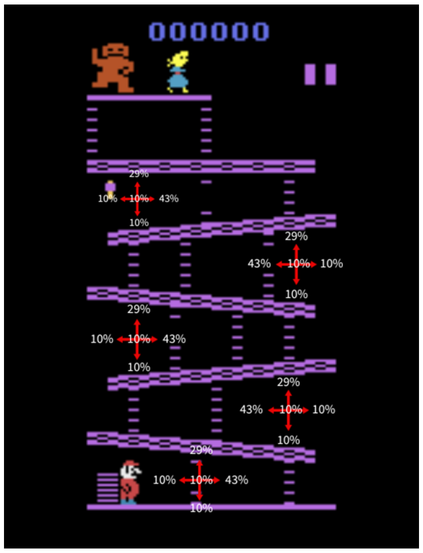


Figure 9: Approximate probabilities of taking different actions at the beginning of training with AH depending on Mario's position. **Percentages do not add up to 100 due to rounding.**

5.6 Summary of modifications

Modification	**Type**	**Value**
Punish needless jumping (PNJ)	Reward modifier	**-20** for jumping if not close to any barrel or ladder
Ladders incentive (LAD)	Reward modifier	**+10** for moving up the ladder and **-15** for moving down the ladder
Level incentive (LVL)	Reward modifier	**+200** for reaching a higher level (floor)
Magic stars incentive (MS)	Reward modifier	**+20** for reaching a star
Action heuristics (AH)	Action probabilities modifier	**even floor:** 3/7 right, 2/7 up, 2/21: noop, left, down; **odd floor:** 3/7 left, 2/7 up, 2/21: noop, right, down

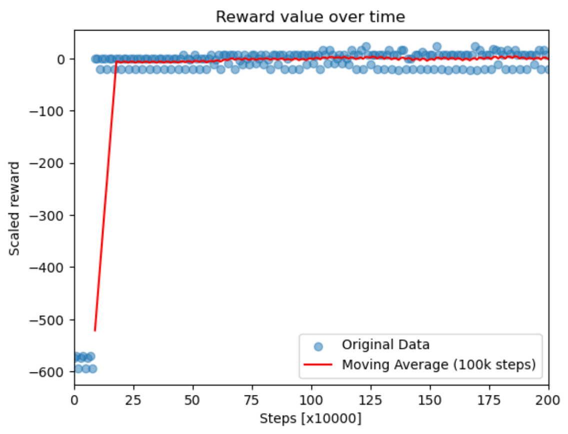
Table 1: Summary of newly introduced modifications in the algorithm.

6 Results

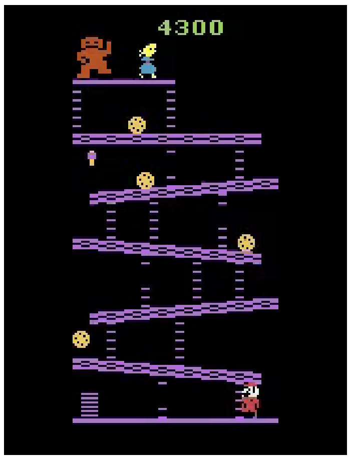
We tested our model with different combinations of modifications. Removing punishment for death never helped, so we present here only results that include punishing death. For every training, we saved video simulations and reward values after every evaluation. For every combination of modifications, we trained the AI model with parameter gamma equal to 0.99 and linear schedule with epsilon starting at 1 and ending up at 0.05 at 30% of steps.

6.1 PNJ

At first, we added only punishing needless jumping modification. At the beginning of training, Mario was jumping too much, so he was getting punished. We can see that on the plot 10a. Then, he learned to avoid jumping, so he was mostly going back and forth, which resulted in a reward of around 0.



(a) Plot showing the value of reward over 2 million steps.

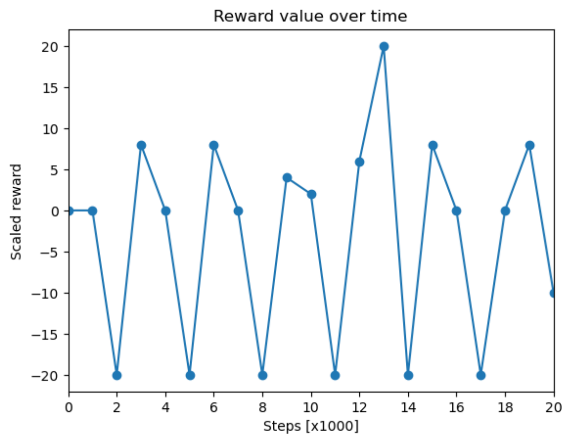


(b) One of the frames in the final evaluation.

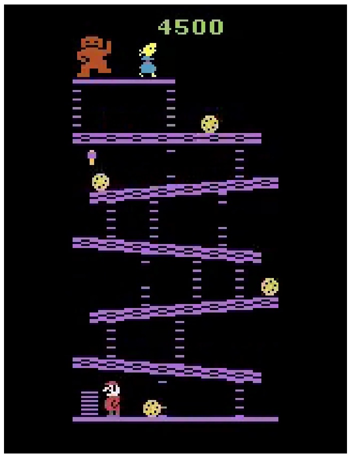
Figure 10: Results of training the agent with modification punishing needless jumping (PNJ) only.

6.2 MS

Next, we did some tests with only the magic stars incentive added. Occasionally, Mario collected some stars; however, mostly he was walking in circles, which usually ended with a quick death. It can be noticed while looking at the reward plot 11a below.



(a) Plot showing the value of reward over 200k steps.

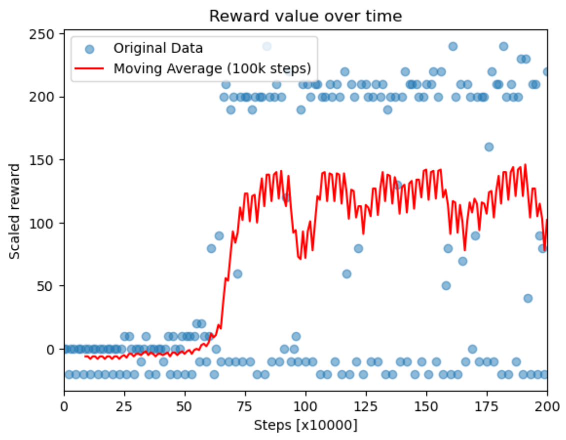


(b) One of the frames in the final evaluation.

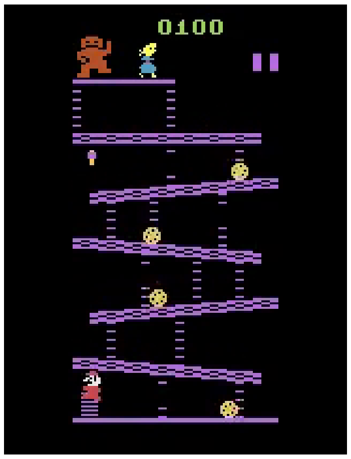
Figure 11: Results of training the agent with modification magic stars incentive (MS) only.

6.3 LVL

Adding the level incentive modification only forced our agent to reach a higher altitude. We hoped it would result in Mario climbing the ladders and reaching the next floors. Unfortunately, Mario was focused on the short-term reward. Therefore, as soon as he discovered this new reward, he started to jump constantly. That is why on the plot 12a at the beginning, the reward is around 0 and then increases with jumping.



(a) Plot showing the value of reward over 2 million steps.

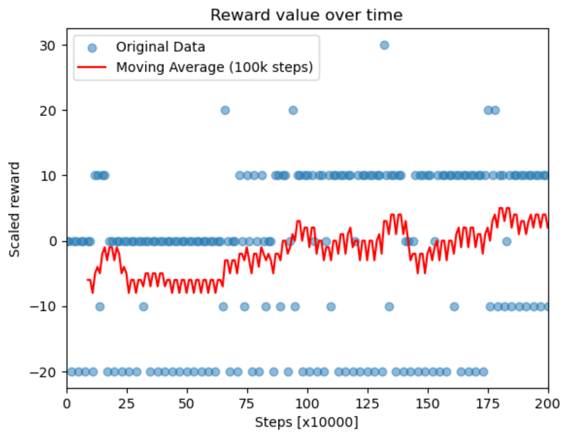


(b) One of the frames in the final evaluation.

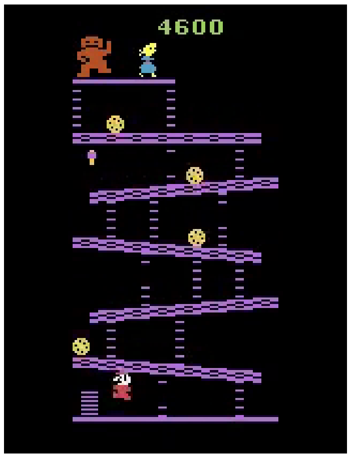
Figure 12: Results of training the agent with modification level incentive (LVL) only.

6.4 AH

Adding action heuristics modification only did not change much compared to the baseline algorithm. The agent was just picking actions randomly. Sometimes, he managed to jump over a barrel, which gave him points. But usually, he died quickly with a negative reward, as one can see on the plot 13a.



(a) Plot showing the value of reward over 2 million steps.

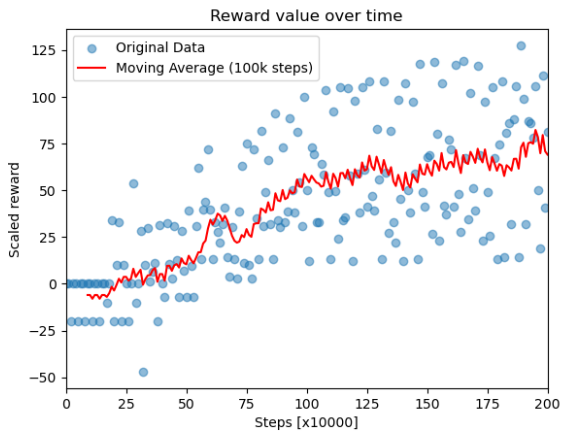


(b) One of the frames in the final evaluation.

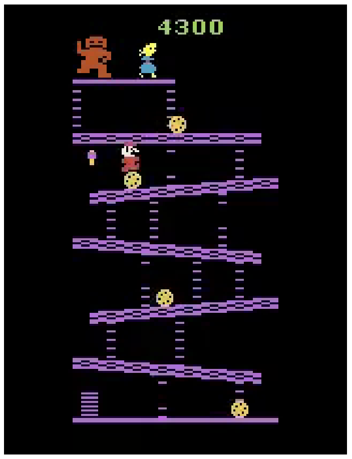
Figure 13: Results of training the agent with modification action heuristics (AH) only.

6.5 LAD+LVL

Our agent made great progress only when we combined the ladders and level incentives. Then, he finally learned to climb ladders. While training, he was learning how to get to higher and higher floors, and in the end, he managed to get to as high as 4th floor. Since there is a high reward for getting to the next floor, the reward value increased with every learned floor. This is visible on the plot 14a.



(a) Plot showing the value of reward across 2 million steps.

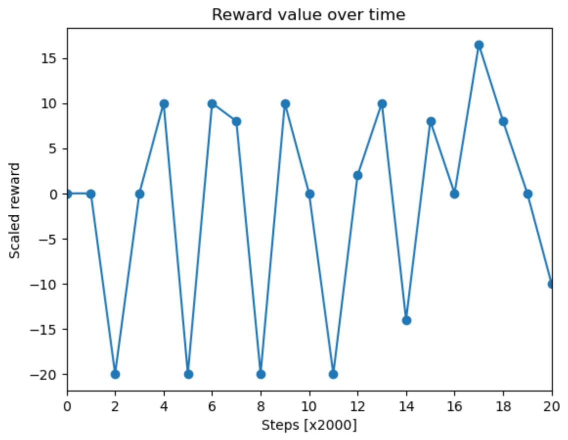


(b) The last frame before Mario's death in the final evaluation.

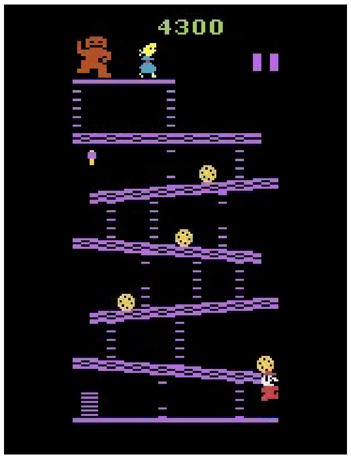
Figure 14: Results of training the agent with the combination of modifications ladders incentive and level incentive (LAD+LVL).

6.6 LAD+LVL+MS

The combination of ladders incentive, level incentive, and magic stars incentive modifications did not give excellent results. Mario was staying on the ground floor the whole time. It might be due to a shorter training time though (200k steps, not 2 million as before). Last time, after so little steps, Mario did not learn anything either. On the plot 15a, some rewards can be noticed. Those are mostly for jumping over barrels and collecting stars on the ground floor.



(a) Plot showing the value of reward across 200k steps.

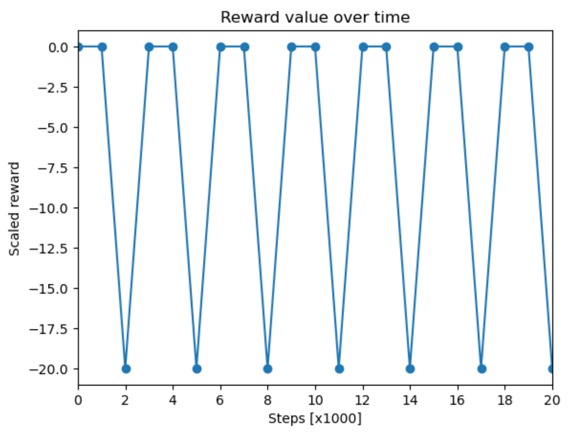


(b) One of the frames in the final evaluation.

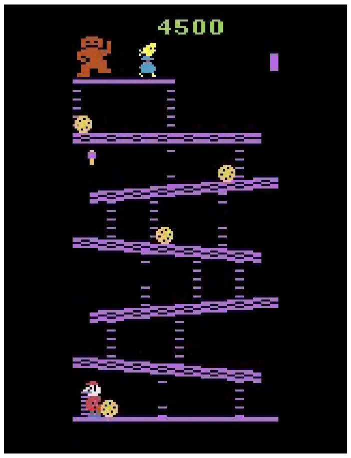
Figure 15: Results of training the agent with the combination of modifications ladders incentive, level incentive, and magic stars incentive (LAD+LVL+MS).

6.7 PNJ+AH

We also tested the combination of punishing needless jumping and action heuristics. As shown on the plot 16a, Mario was mostly punished for death and not rewarded for anything. He was just standing still or going in circles. It is probably because he did not have any incentive to climb.



(a) Plot showing the value of reward across 200k steps.

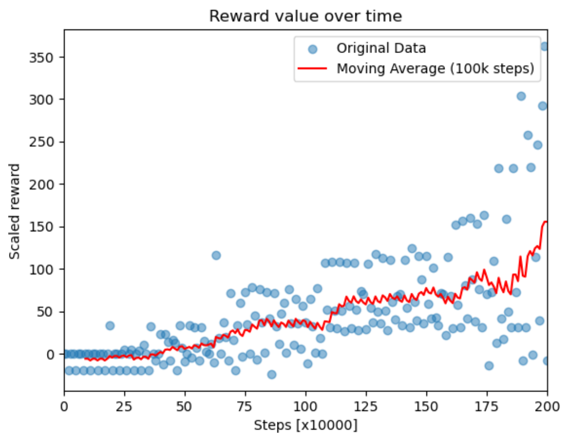


(b) One of the frames in the final evaluation.

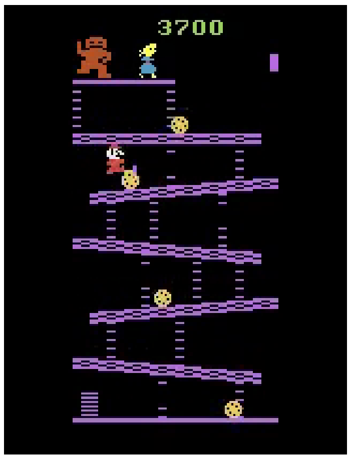
Figure 16: Results of training the agent with the combination of modifications punishing needless jumping and action heuristics (PNJ+AH).

6.8 LAD+LVL+PNJ

The combination of ladders incentive, level incentive, and punishing needless jumping worked wonders. Similarly to LAD+LVL, the reward value increased with every learned floor, which can be observed on the plot 17a. In addition, Mario caught the hammer and was able to hit the barrels with it, which scored him a great deal of points.



(a) Plot showing the value of reward across 2 million steps.

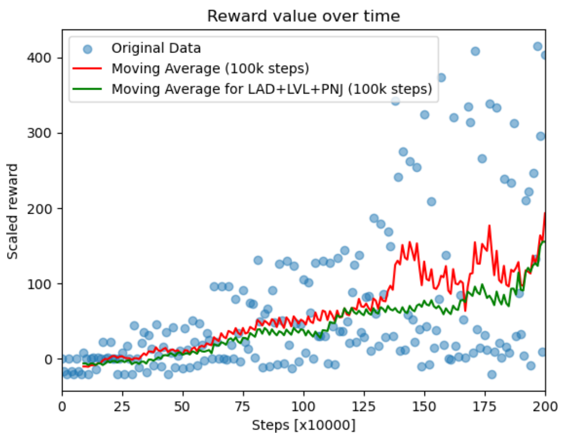


(b) The last frame before Mario's death in the final evaluation.

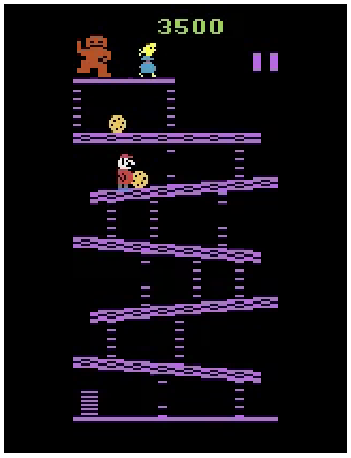
Figure 17: Results of training the agent with the combination of modifications ladders incentive, level incentive, and punishing needless jumping (LAD+LVL+PNJ).

6.9 LAD+LVL+MS+PNJ

We got similar results after adding the magic stars incentive to the mix (combining ladders incentive, level incentive, magic stars incentive, and punishing needless jumping). However, it is evident on the plot 18a that our agent learned to climb higher faster and scored even more points in the end.



(a) Plot showing the value of reward across 2 million steps.

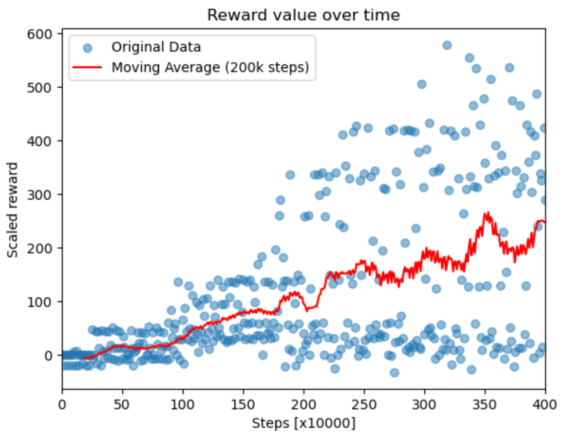


(b) The last frame before Mario's death in the final evaluation.

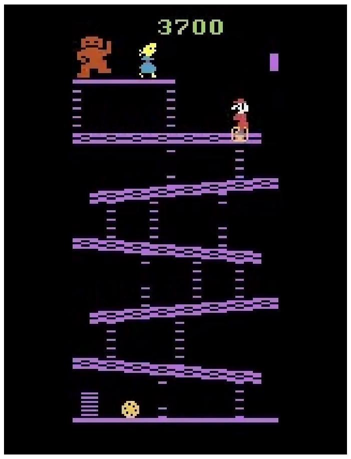
Figure 18: Results of training the agent with the combination of modifications ladders incentive, level incentive, magic stars incentive, and punishing needless jumping (LAD+LVL+MS+PNJ).

6.10 Final results

Our final results include all the modifications: ladders incentive, level incentive, magic stars incentive, punishing death, punishing needless jumping, and action heuristics. We trained our agent across 4 million steps with parameter gamma equal to 0.99 and linear schedule with epsilon starting at 1 and ending up at 0.05 at 30% of steps. The training took 12 hours on GPU. In the final simulation, Mario managed to reach the hammer and destroy some barrels on the 4th floor and then even get to the 5th floor. However, he died on that floor and did not rescue his girlfriend. Furthermore, it is clear on the plot 19a that the reward value stopped steadily increasing, which indicates that it became very difficult for the agent to continue learning.



(a) Plot showing the value of reward across 4 million steps.



(b) The last frame before Mario's death in the final evaluation.

Figure 19: One of the frames from the final simulation resulting from training the agent with the combination of ladders incentive, level incentive, magic stars incentive, punishing death, punishing needless jumping, and action heuristics (LAD+LVL+MS+PD+PNJ+AH).

7 Summary

The goal of the project was to develop an AI capable of performing at a human level in Donkey Kong by completing the first level. The approach involved using reinforcement learning techniques, specifically Deep Q-Networks (DQN) and SARSA algorithms, to train the AI agent.

The DQN algorithm was chosen for its ability to combine Q-Learning with deep neural networks, while SARSA was tested for its on-policy learning characteristics. The initial reward system included basic incentives such as points for jumping over barrels and penalties for dying, aiming to guide the agent's learning process. Despite these efforts, early results showed limited success, with the AI failing to progress effectively beyond random actions and frequent deaths.

To address the initial shortcomings, several modifications were made to the reward system to encourage more effective gameplay behaviors. These included punishing needless jumping (PNJ) to discourage excessive and unnecessary jumps and introducing rewards for climbing ladders (LAD) and reaching higher levels (LVL) to promote vertical movement. Additional incentives, such as "magic stars" (MS), were placed along the path to encourage progression toward the game's goal. Action heuristics (AH) were also implemented to guide the agent's action selection based on its current position, further refining the AI's decision-making process.

Initial training sessions revealed that the AI struggled to progress effectively, often resulting in the agent either standing still or engaging in ineffective actions. By combining various reward-modification strategies, we observed improved outcomes, such as better ladder climbing and point scoring. Although the AI model consistently reached higher floors up to the fourth floor and even managed to grab the hammer, it ultimately failed to complete the first level and save Mario's girlfriend, thus not fully meeting the main goal.

Despite not achieving the primary objective, the AI still managed to get through most of the first level and plays better than half of our team, fulfilling our secondary goal. This performance showcases the potential of reinforcement learning in mastering intricate gameplay mechanics, indicating that with further refinement and optimization, the AI could eventually achieve the main goal of completing the first level and saving Mario's girlfriend. The achievements thus far validate the effectiveness of the implemented reward modifications and highlight the promise of continued development in this domain.

8 Possibilities for development

Since we did not manage to achieve our goal, there is obviously some room for improvement. We had some ideas that we did not have time to implement. We will list them and explain why we think they might prove to be beneficial.

Ideas to implement:

- There is one idea we started implementing but abandoned it. This idea is learning from human feedback. What we mean by that is recording some games played by humans and post-training the model with that data. We wrote some scripts for that but realized we would need much more time to collect proper data. At this point, our agent tends to die on the 4th floor. Therefore, showing how to climb to the sixth floor and save the girl would probably be really beneficial. However, the game turned out to be very difficult for us, so it would take us a great deal of time to save Mario's girlfriend enough times.

- The algorithms we used were DQN and SARSA, but there might be better choices. We were also considering Proximal Policy Optimization (PPO) for a moment. While trying to improve the agent's performance other algorithms should also be explored.

- Another promising area for development is enhancing the robustness of our training environments by incorporating domain randomization techniques. By varying environmental factors such as barrel patterns, ladder placements, and obstacle speeds, the agent could learn to adapt to a broader range of scenarios. This approach could significantly improve the agent's generalization capabilities, making it more adept at handling the unpredictability inherent in Donkey Kong gameplay.

- Additionally, integrating advanced exploration strategies like curiosity-driven learning could further refine the agent's ability to discover optimal paths and strategies autonomously, potentially accelerating the learning process and improving overall performance.

- Finally, our reward system is rather still not perfect. Refining the reward system with more granular rewards for specific actions or sequences could lead to more nuanced learning and improved strategies.

References

[1] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[2] Shengyi Huang et al. “CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms”. In: *Journal of Machine Learning Research* 23.274 (2022), pp. 1–18. URL: http://jmlr.org/papers/v23/21-1342.html.

[3] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[4] Paul Ozkohen et al. “Learning to Play Donkey Kong Using Neural Networks and Reinforcement Learning”. In: Nov. 2017.

[5] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[6] Richard S. Sutton and Andrew G. Barto. “Chapter 6.4 Sarsa: On-Policy TD Control”. In: *Reinforcement Learning: An Introduction*. 1st. Cambridge, MA: MIT Press. URL: http://incompleteideas.net/book/ebook/node64.html.

[7] Richard S. Sutton and Andrew G. Barto. “Q-learning and SARSA: Chapters 6.4, 6.5”. In: *Reinforcement Learning: An Introduction*. 2nd. The MIT Press, 2018, pp. 129–132. URL: http://incompleteideas.net/book/RLbook2020.pdf.

[8] Mark Towers et al. *Gymnasium*. Version 1.2.0. 2023. DOI: 10.5281/zenodo.8127025. URL: https://gymnasium.farama.org/.