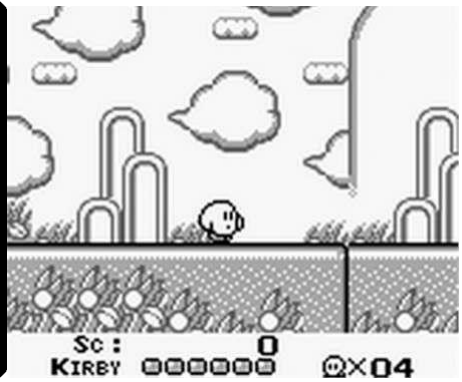
Reinforcement learning agent for Kirby's Dreamland 🎮 🤖

Advances in AI – WS 2024/25 HSD

Maurice Füsser
Martikelnumber: 925312

SCAN ME



Contents

1. Intodruction

1.1 Project Description

The goal of this project is to develop and implement a Reinforcement Learning (RL) agent capable of successfully completing the first level of Kirby's Dreamland on the GameBoy. This is achieved using the PyBoy emulator, which allows direct interaction with the game and provides an environment for training the RL model.

The agent employs a Double Deep Q-Network (DDQN) to optimize its playing strategy through experience. The training runs for 25,000 epochs, with each episode consisting of up to 2,500 steps. Throughout the training, the agent encounters various challenges, including obstacles, enemies, and platform mechanics. By continuously interacting with the game environment, the agent learns to overcome obstacles, defeat enemies, and complete the level.

The learning process is structured around a reward system, which assigns positive rewards for successful actions and penalties for undesirable behaviors. For example, defeating enemies is rewarded, while losing a life incurs a heavy penalty. This mechanism promotes an optimized playing strategy, enabling the agent to make efficient decisions.

The DDQN network facilitates stable and efficient learning. Unlike traditional Q-learning methods, it utilizes two separate neural networks – an online network and a target network – to mitigate overestimation of Q-values. This results in more robust and efficient learning, which is crucial for navigating complex levels.

1.2 Motivation

Since childhood, I have been a huge fan of video games, particularly classic Nintendo titles. The ability to analyze game mechanics and develop strategies has always fascinated me. When I discovered that Reinforcement Learning (RL) could be used to program an AI agent capable of autonomously completing levels in Super Mario, I was immediately intrigued.

However, instead of Mario, I wanted to tackle a new challenge – Kirby’s Dreamland on the GameBoy. Compared to Super Mario, Kirby's environment is more complex due to a wider range of movement options, including jumping, flying, and various attacks. These expanded mechanics make the game an ideal platform for exploring and demonstrating RL agent capabilities.

Additionally, Kirby’s Dreamland is often overlooked as an RL test environment. While many RL projects focus on well-known environments like Atari or Super Mario, I wanted to show that even more complex platformers with non-linear progression can be successfully navigated by RL agents. This project provided me with the opportunity to delve deeper into RL concepts and explore the limits of AI-controlled GameBoy gameplay.

2. State oft he Art

2.1 Reinforcement Learning (RL)

Reinforcement Learning (RL) is a machine learning technique where an agent learns to make optimal decisions through interaction with an environment. The agent receives rewards or penalties based on how its actions impact the achievement of a specific goal. This process occurs in a loop, enabling the agent to develop a strategy (policy) over time that maximizes its long-term reward. [1]

2.2 Deep Q-Network (DQN)

Deep Q-Networks (DQN) combine traditional Q-learning with deep neural networks, allowing RL agents to handle high-dimensional state spaces that are challenging for classical RL methods.

A notable application of DQNs was their successful implementation in Atari 2600 games. In the study by Mnih et al. (2013), a DQN agent was trained directly from raw pixel data and was able to play multiple Atari games at a human-level performance or even surpass human players.

The key innovation of DQNs lies in their ability to approximate the Q-function – which represents the expected reward of an action in a given state – through a neural network. This allows the agent to learn directly from raw inputs (e.g., pixel values) without manual feature extraction. Additionally, DQNs employ techniques such as the replay buffer, which stores past experiences and samples them randomly for training. This reduces correlation between consecutive learning steps, leading to a more stable learning process.[2]

2.3 Double Deep Q-Network (DDQN)

Although DQN marked a significant advancement in RL, it suffers from a major drawback: the overestimation of Q-values. This occurs because the same neural network is used for both action selection and Q-value evaluation, leading to overly optimistic estimates that destabilize learning and result in suboptimal strategies.

To address this issue, Double Deep Q-Networks (DDQN) were introduced. DDQNs extend the original DQN architecture by employing two separate networks: one for action selection and another for value estimation. This separation prevents the model from learning systematically overestimated Q-values, thereby enhancing training stability and efficiency.

DDQNs represent a robust improvement over DQNs and were specifically chosen for this project to handle the dynamic environment of Kirby's Dreamland more effectively.[3]

[1] Vgl. Vincent François-Lavet et al[2018] S. 15-17
[2] Vgl. Mnih et al[2013], o.S.
[3] Vgl. Van Hasselt et al., [2015], o.S.

3. Technology and Methodology

3.1 PyBoy Emulator and Environment

„This project utilizes the PyBoy Python library to train a Reinforcement Learning (RL) model for *Kirby's Dreamland*. PyBoy emulates the GameBoy ROM and provides a Python API, enabling direct access to crucial game information such as Kirby's lives, position, score, and level progress. This direct interaction is essential for RL training, as it allows precise analysis of the game state and flexible adaptation of the reward structure.

A key advantage of PyBoy is its high emulation accuracy, ensuring a realistic gameplay experience. Additionally, PyBoy supports Reinforcement Learning through fast step-by-step rendering, allowing the agent to learn incrementally from its actions. The control mechanism is handled via the send_input() function, which sends specific button commands to control the game.

The agent can directly read the game state from the emulator's memory and adjust its behavior in real-time. This optimization enables the agent to refine its strategy more quickly and learn which actions lead to better outcomes.

Furthermore, PyBoy offers additional features such as screenshots and video recording, allowing training progress to be monitored. Overall, PyBoy provides a stable and adaptable environment for RL training, enabling efficient and data-driven optimization of the agent."

*ChatGPT

3.2 Processing of Game Data (Preprocessing)

Instead of analyzing raw image data from the game, the agent utilizes the internal game state directly from the PyBoy emulator's memory. This approach has significant advantages over traditional screen image processing, as it allows for a more efficient extraction of relevant information.

A conventional RL agent that processes image data would need to handle each frame as raw input, possibly downscale it to a smaller resolution, and perform grayscale conversion. This image processing requires substantial computational resources, as the neural network would have to learn to extract visual features such as character positions, obstacles, and enemies from pixel information. This not only increases memory consumption but also significantly extends training time.

Instead, I directly accessed the emulator's memory to retrieve numerical state values, such as Kirby's position, enemy locations, and obstacles, which are then passed to a Convolutional Neural Network (CNN). The CNN, which is discussed in detail in the Network Architecture section, processes this information in a compact array format (Figure 1). By avoiding complex image processing, this approach makes computation more efficient and resource-friendly.

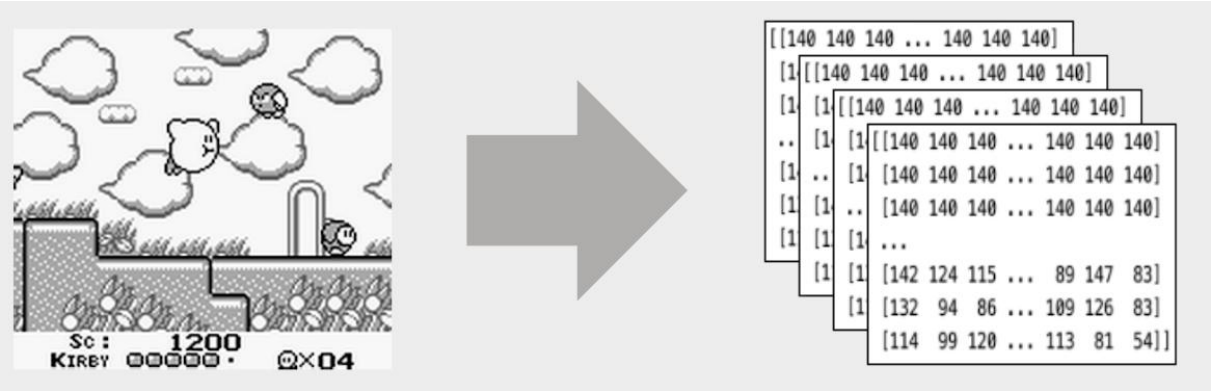


Figure 1: image for Array (Gamestate)
Source: Poster „Schafft es Kirby?

Another advantage of this method is that the agent can make decisions more quickly. As no additional calculations are required for image analysis, the model can make more precise decisions in less time. This enables better adaptation to the environment and leads to more efficient training overall.

3.3 Action space – controlling the agent

The Action Space determines which actions the RL agent can perform in the game. Compared to other games such as Super Mario, Kirby has an extended control system with running, jumping, flying and attacking, which is why the model not only has to learn individual actions, but also combinations.

As can be seen in the image, actions are saved as numerical indices and sent via PyBoy (Figure 2). In addition to basic movements such as running, jumping and attacking, there is a special action (index 6) that releases all previous inputs in order to simulate realistic control scenarios.

```python
# Define possible actions
action_mapping = {
    0: [WindowEvent.PRESS_ARROW_RIGHT],
    1: [WindowEvent.PRESS_ARROW_LEFT],
    2: [WindowEvent.PRESS_BUTTON_A],
    3: [WindowEvent.PRESS_BUTTON_B],
    4: [WindowEvent.PRESS_ARROW_UP],
    5: [WindowEvent.PRESS_ARROW_DOWN],
    6: [WindowEvent.PRESS_ARROW_RIGHT, WindowEvent.PRESS_BUTTON_A],  # Right + Jump
    7: [WindowEvent.PRESS_ARROW_RIGHT, WindowEvent.PRESS_BUTTON_B],  # Right + Attack
    8: [WindowEvent.PRESS_BUTTON_A, WindowEvent.PRESS_ARROW_UP],  # Jump + Up
    6: [
        WindowEvent.RELEASE_ARROW_RIGHT,
        WindowEvent.RELEASE_ARROW_LEFT,
        WindowEvent.RELEASE_BUTTON_A,
        WindowEvent.RELEASE_BUTTON_B,
        WindowEvent.RELEASE_ARROW_UP,
        WindowEvent.RELEASE_ARROW_DOWN,
    ],
}
```

Figure 2: shows how the actios are defined
Source: main.py

3.3.1 Importance for reinforcement learning

A well-defined action space is crucial for the learning success of the RL agent. If the action space is kept too simple, the agent will not be able to perform all the necessary movements and may

learn inefficient game strategies. On the other hand, if it is too complex, the number of possible actions increases drastically, which can slow down learning.

The targeted definition of simple and combined actions creates a good balance that allows the agent to learn efficiently without being overwhelmed by an excessive variety of actions. This is particularly important for Kirby's Dreamland, as the game relies heavily on dynamic movement and timing.

4. Implementation oft he RL-Agenten

4.1 Network architecture

„The RL agent uses a Convolutional Neural Network (CNN) to analyze the states of the game and derive optimal actions. A CNN is a specialized form of neural network that is particularly suitable for processing spatial data.

CNNs consist of several layers that successively extract important features from the input data by recognizing local patterns such as edges, shapes or directions of movement. In contrast to the classic processing of image data, the model uses the numerical game states from the PyBoy emulator. This enables significantly more efficient processing, as no complex image processing is required and training is therefore faster and more resource-efficient

faster and more resource-efficient (Figure 3).“ *ChatGPT

```python
def _build_model(self):
    """
    Constructs the neural network used for predicting Q-values.
    The architecture includes convolutional layers for feature extraction
    followed by fully connected layers for decision making.
    """
    model = nn.Sequential(
        nn.Conv2d(
            4, 32, kernel_size=3, stride=1, padding=1
        ),  # Input: 4 frames, Output: 32 feature maps
        nn.ReLU(),
        nn.Conv2d(
            32, 64, kernel_size=3, stride=2, padding=1
        ),  # Downsamples by a factor of 2
        nn.ReLU(),
        nn.Flatten(),  # Flattens the output for the fully connected layers
        nn.Linear(64 * 10 * 8, 128),  # Adjusted to match flattened size
        nn.ReLU(),
        nn.Linear(
            128, self.action_size
        ),  # Output layer with Q-values for each action
    )
    return model
```

Figure 3: shows how the CNN
Source: agent.py

- **Input layer:** The model processes four consecutive frames, allowing the agent to capture not only the current state, but also motion sequences.

- **Feature extraction through convolutional layers (Conv2d):** The first layers consist of convolutional layers that extract important features from the gamestate.

- **ReLU activation:** Ensures that only important information is processed further and that the model can recognize complex relationships.

- Flattening: The extracted features are converted into a one-dimensional vector.

- **Output layer:** The model outputs a Q value for each possible action, which indicates the expected long-term reward for a particular action.

4.2 Training and hyperparameter-tuning

An episode in training is defined by various conditions: It ends either when Kirby successfully completes the level, loses all his lives (e.g. due to too many hits or falling into the abyss) or when he has performed 2,500 actions within an episode. Every action Kirby performs, from moving to jumping to attacking, counts as a single step. Even if Kirby stands still, this is counted as an action. This mechanic is explained in more detail later in the reward system.

After each episode, the learned parameters of the model are updated. The neural network adapts through backpropagation and optimizes itself iteratively over many thousands of epochs. In the training, the model was trained with 25,000 epochs.

An epsilon-greedy strategy was used so that Kirby can train optimally and also try out enough. This ensures a good balance between exploration (trying out new actions) and exploitation (using already known successful strategies).

This strategy is crucial, as too early exploitation could lead to Kirby retaining inefficient strategies. Too much exploration, on the other hand, would slow down learning.

4.2.1 Definiton of hyperparameter

The training was carried out with the following hyperparameters:

- Batch Size: 256 - Determines how many experiences are processed per learning step.
- Gamma: 0.97 - Influences how heavily future rewards are weighted.
- Learning Rate: 1e-4 - Specifies how quickly the network adapts.
- Epsilon Start: 1.0 - At the beginning Kirby explores a lot and tries out many random actions
- Epsilon Decay: 0.999999975 - The probability of random actions decreases with every action Kirby performs
-Epsilon End: 0.1 - Kirby concentrates on the strategies he has learned. At the end, 10% remains for exploration in order to develop new strategies in later training phases that no longer have too serious an impact.
-Target Update Frequency: 5000 - The target network is updated regularly to enable more stable learning processes.
-Replay Memory Size: 500,000 - Saves past experiences that are used for training.

4.3 Reward system and learning

The reward system is a central component of the RL model, as it tells the agent which actions are advantageous or disadvantageous. It helps as a feedback mechanism that significantly influences Kirby's decision making during training. The smallest adjustments to the rewards or punishments can have drastic effects on the agent's behavior. Therefore, developing an effective reward system requires a lot of experimentation and optimization to develop an intuitive idea of how certain adjustments affect the training outcome.

4.3.1 Importance of the reward system

A well-balanced system ensures stable learning progress and prevents the agent from developing suboptimal strategies. Unfavorable reward structures can cause the agent to learn undesirable patterns, such as intentionally losing lives to maximize other rewards. Targeted reinforcement of certain behaviors ensures that Kirby can complete the level more efficiently.

4.3.2 Design of the reward system

Action	**Reward / Punishment**	**Goal**
Move right	+10	Promoting forward movement
Move left	-5	Prevent Kirby from going backwards
Lose life	-10	Avoidance of risky actions
No movement	-1	Faster level completion
Momement on Y-axis only	-40	Avoidance of constant flying
Score increase	+5	Incentives for attack instead of collision
Reach Warpstar (level target)	+20000	Strong reward for completing levels

This reward system allows Kirby to focus on running, attacking and completing the level efficiently. Experimentation with various parameters has ensured that the agent inadvertently learns sub-optimal strategies, such as deliberately losing lives to defeat opponents.

4.3.3 Learning strategy and iterative optimization

The reward system was tested in several stages and gradually adapted to encourage optimal behavior.

- **First phase:** Kirby often ran around aimlessly or got stuck. The rewards were fine-tuned.

- **Second phase:** Running to the left was penalized more to prioritize movement to the right.

- **Third phase:** Enemy attacks were rewarded more so that Kirby would fight more actively instead of just running.

- **Final optimization:** Reaching the Warpstar has been given the highest reward to reinforce the overall goal.

These adjustments significantly improved training. Kirby learned to move around, fight enemies, avoid obstacles and traverse the level efficiently.

4.4 Memory buffer

The replay memory buffer stores the agent's experiences in the form of state, action, reward, next state and the end-of-episode indicator. These experiences are then used to improve the stability of the learning process and reduce the correlations between successive learning steps.

5. Challenge and insights

During the development and training of the RL agent for Kirby's Dreamland, several challenges arose that not only deepened the understanding of reinforcement learning, but also highlighted the need for iterative optimization methods. In this section, some of the central problems and possible solutions are explained in more detail.

5.1 Challenge of the reward system

“One of the biggest challenges was the design of a functioning reward system. Since the RL agent learns through rewards and punishments, these had to be carefully tuned. A small change could have drastic effects on the game behavior.

The goal was to find an optimal balance so that Kirby not only completes the level as quickly as possible, but also develops meaningful strategies: (Figure 4)

- **Problem:** Kirby learned to complete the level by simply flying over all obstacles without fighting enemies or using platforms.

- **Solution**: Rewards for defeating enemies and overcoming obstacles were increased.

- **Problem:** Kirby realized that it was more beneficial to intentionally take damage to defeat enemies instead of actively attacking.

- **Solution:** Punishments for losing lives were increased to encourage more defensive behavior.

These challenges illustrate that the reward system cannot simply be developed according to a fixed scheme, but must be optimized through iterative experimentation.” *ChatGPT

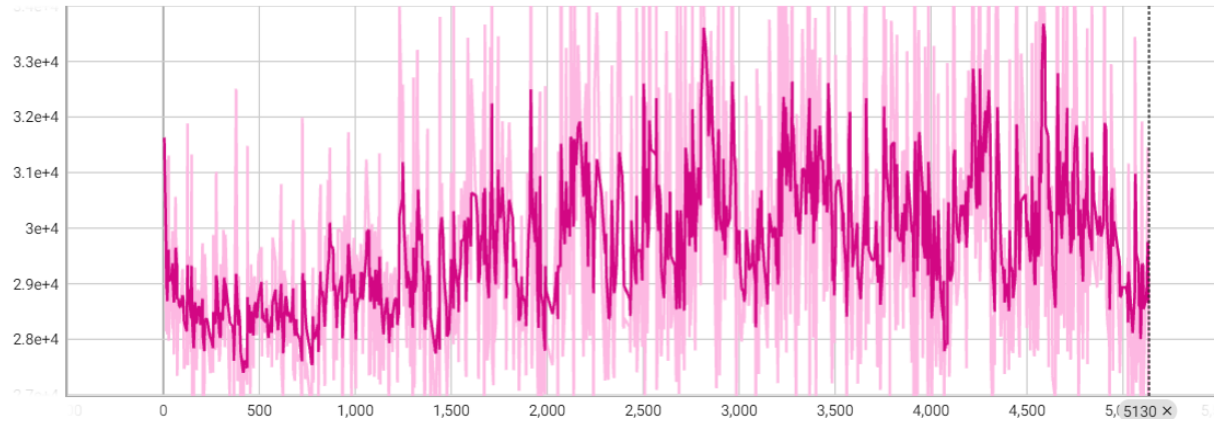


Figure 4: Shows how the rewards (total) stand at level completion
Source: Tensorboard

5.2 Epsilon-Greedy Strategie

“Another hurdle was the optimal choice of the epsilon-decay value. Since the epsilon-greedy strategy initially prioritizes random actions to explore different strategies, it needs to shift the focus to learned patterns over time.

- Problem: Dropping the Epsilon value too quickly caused Kirby to settle on inefficient strategies after just a few thousand episodes instead of experimenting further.

- Solution: The decay value was chosen so that Kirby only relies predominantly on learned patterns after many thousands of episodes (final epsilon value of 0.1 after around 9,000 episodes) (Figure 5).

- Optimization possibility: An adaptive epsilon adjustment that adapts to the learning success could further improve the problem. For example, the epsilon decay could not be linear, but adapt dynamically to the progress of the agent."

- *ChatGPT

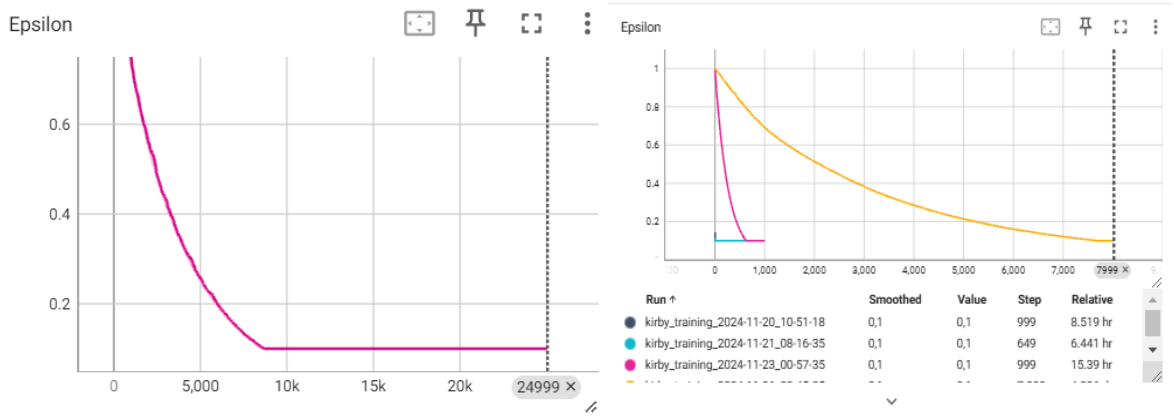


Figure 5: Shows how Epsilon decreases after each epoch and converges to 0.1
Soruce: Tensorboard

5.3 Monitoring and analysis with Tensorboard

To optimize the learning process, the TensorBoard was used to monitor various metrics of the training. This allowed errors to be detected more quickly and adjustments to be made more efficiently. For example, it was possible to see how the level completions developed over the training period and whether the agent learned certain suboptimal behaviors. (Figure 6)

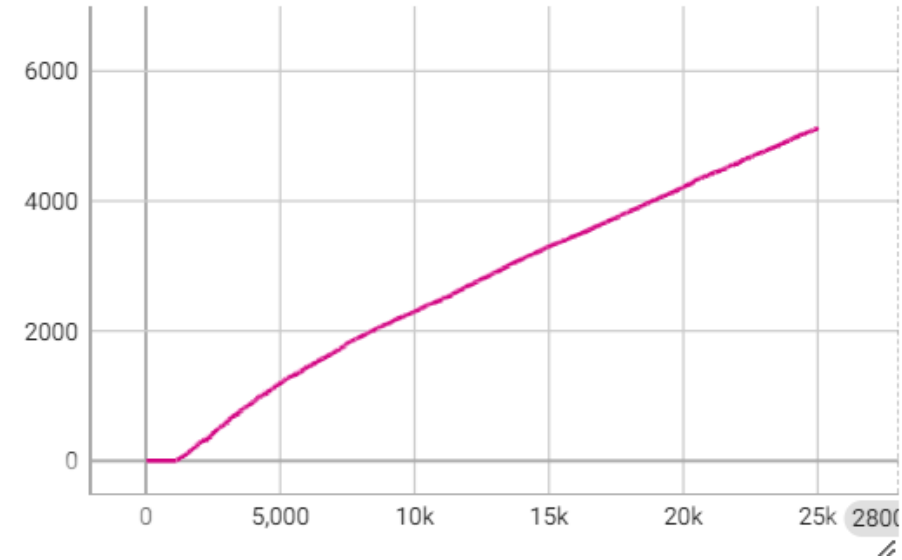


Figure 6: Total level completions by 25000 epochs
Source: Tensorboard

5.4 Comparison of different training models

The progress of Kirby (agents) was evaluated on several models with different training times.

Epochs	**Training time with hardware**	**Observation**
1000 epochs	Approx. 20h with an Nvidia RTX 3070 (GPU)	Kirby did not yet show any real learned behavior, tried out many different actions and had no recognizable strategy
8000 epochs	5 days on an Nvidia RTX 3070 (GPU)	Kirby was able to complete the level but with inefficient methods
25000 epochs	5 days on an Nvidia RTX 4090 (GPU)	Kirby had developed a strategy and was able to complete the level well, over 5000 times in 25000 epochs

Another problem was that Kirby realized that taking damage was a quicker solution to defeating enemies than actively attacking. This shows that an RL agent does not necessarily act "intelligently", but is merely optimized to maximize its rewards.

To solve this problem, a gradual adjustment of the reward system would have made sense, so that new reward mechanisms are introduced after a certain number of epochs.

5.5 Training progression - episode length

The diagram shows the length of the episodes over the entire training course. An episode ends when Kirby either completes the level, loses all his lives or has performed 2,500 actions. At the beginning, it is clear to see that most episodes only include a small number of actions. This indicates that Kirby failed early on in the level, usually by falling into the abyss or by repeatedly losing lives with enemies that resulted in the loss of all lives.

As training progresses, the dispersion of episode lengths decreases and stabilizes in a range between around 1,800 and 2,000 actions. This shows that Kirby acts more and more efficiently, survives longer stages in the level and is more likely to reach the goal. The frequent episodes with a high number of actions indicate that Kirby has learned an effective strategy to successfully complete the level.

This development confirms that the training is progressing and the RL agent is increasingly able to apply successful strategies to master the level. (Figure 7)

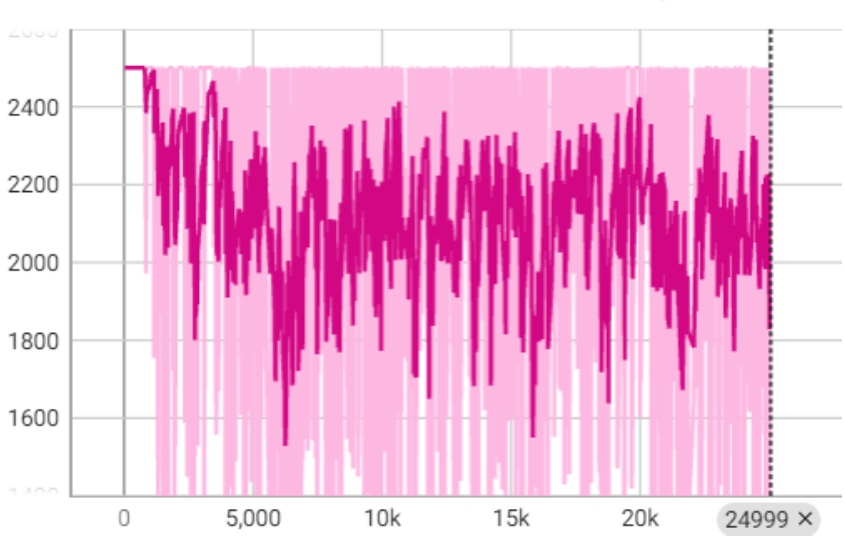


Figure 7: Episode length over the entire training course
Source: Tensorboard

6. Summary and outlook

6.1 Summary of the project

The project „Reinforcement Learning: Playing Kirby's Dreamland“ demonstrates the successful application of a Double Deep Q-Network (DDQN) in combination with the PyBoy emulator to autonomously control Kirby through the first level of the game. Throughout the training process, it became evident that Reinforcement Learning is an extremely powerful technique, but one that requires precise tuning of the network architecture, hyperparameters, and, most importantly, the reward system.

The biggest challenge was defining an effective reward system and selecting the optimal hyperparameters. Through continuous adjustments and extensive testing, a model was developed that enabled Kirby to successfully complete the level.

One of the most exciting insights gained was how small changes in parameters could drastically impact the agent’s behavior. This made the project both fascinating and challenging, as it required a deep understanding of the underlying mechanisms. Despite occasional frustrations due to unexpected results or long training times, it was motivating to see how the RL agent gradually developed effective strategies.

Another key learning experience was the use of tools like TensorBoard to monitor and analyze the training process. These tools made it possible to visualize the model’s behavior and make targeted optimizations.

Ultimately, the project was a highly rewarding experience, showcasing the immense potential of Reinforcement Learning for solving complex problems

6.2 Potential Improvements

Although Kirby successfully completes the level, there are still areas for improvement. Fine-tuning the reward and penalty system could help further reduce undesired behaviors, such as intentionally taking damage as a faster method to defeat enemies.

Another critical optimization would be the adjustment of the epsilon decay to ensure that the agent explores sufficiently in the early training phases but does not shift into an exploitative phase too early or too late. A dynamic epsilon decay strategy, which adapts based on training progress, could be a more effective approach.

Extending the training duration with a gradual adjustment of the reward structure could further enhance learning efficiency. A staged reward system—where the agent first learns fundamental mechanics before optimizing for level completion—might yield better results.

A particularly interesting next step would be to expand the model's capabilities beyond the first level, allowing it to tackle later levels and boss fights. This would require a completely redesigned reward system, incorporating boss-specific mechanics like strategic attacks and adaptive movement.

Another optimization could involve training multiple models in parallel, each with slightly different hyperparameter variations. This would enable a more comprehensive comparison, making it easier to identify the most effective model for the given problem.

6.3 Outlook

Reinforcement Learning is a rapidly evolving research field with tremendous potential, and this project has demonstrated that classic video games can be successfully mastered using RL models.

A promising future enhancement could be exploring alternative RL algorithms, such as Proximal Policy Optimization (PPO), to improve learning stability and efficiency. Additionally, implementing automated hyperparameter optimization could make the training process more adaptive and efficient, reducing the need for manual experimentation.

Beyond game development, Reinforcement Learning has numerous real-world applications in fields such as robotics, autonomous navigation, and complex decision-making systems. The principles applied in this project can be easily transferred to other problem domains, showcasing the versatility of RL techniques.

In summary, this project has demonstrated that given sufficient computational power, time, and iterative experimentation, remarkable results can be achieved in Reinforcement Learning. It will certainly not be the last RL project, and the insights gained here serve as a valuable foundation for future experiments and developments in this exciting research area

List of references

1. V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare und J. Pineau, „An Introduction to Deep Reinforcement Learning“, Foundations And Trends® in Machine Learning, Bd. 11, Nr. 3–4, S. 219–354, Jan. 2018, doi: 10.1561/2200000071. https://arxiv.org/pdf/1811.12560
2. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., & DeepMind Technologies. (n.d.). Playing Atari with Deep Reinforcement Learning. *DeepMind Technologies*. https://arxiv.org/pdf/1312.5602
3. H. Van Hasselt, A. Guez und D. Silver, „Deep Reinforcement Learning with Double Q-Learning“, *Proceedings Of The AAAI Conference On Artificial Intelligence*, Bd. 30, Nr. 1, März 2016, doi: 10.1609/aaai.v30i1.10295. https://arxiv.org/abs/1509.06461

Sources

* Marked sections have been summarized, generated or quoted by ChatGPT.

1. Link to Poster
2. Link to the project GitHub
3. Link to PyBoy

Source code directory

- **main.py:** Main script for initializing and controlling the training process of the RL agent.
    - Includes the initialization of the PyBoy emulator, the loading of the environment and the configuration of the DDQN agent
    - Contains the training loop that executes actions, updates the environment and optimizes the model.

- **agent.py:** Implementation of the Double Deep Q-Network (DDQN) architecture.
    - Defines the CNN-based neural network for learning from the GameState.
    - Includes the replay memory buffer to store and randomly retrieve past experiences
    - Includes the Epsilon Greedy strategy to balance between exploration and exploitation.
    - Saves and loads model weights

- **environment.py:** Defines the Gym-compatible environment for Kirby's Dreamland
    - Creates an interface between the PyBoy emulator and the RL agent
    - Defines the action space and processes GameState information.
    - Implementation of the reward system
    - Frame stacking