Reinforcement Learning Agent for 2048: Initial Analysis

Edward Julian Garcia Gaitan

Universidad Distrital Francisco José de Caldas

ejgarciag@udistrital.edu.co

Miguel Alejandro Chavez Porras

Universidad Distrital Francisco José de Caldas

miachavezp@udistrital.edu.co

Abstract

This project aims to create an agent capable of developing a strategy to not only win the game but also to achieve the highest possible score within the given rules.

Introduction

In an AI and RL perspective, 2048 presents a challenge due to its large state space, tile generation, and the need for long-term strategic planning. Unlike games with more static environments, 2048 requires agents to make decisions under uncertainty, as each move not only affects the current board state but also influences future possibilities.

Developing an AI agent for 2048 involves designing strategies that balance immediate rewards with long-term planning. In this project, the goal is to create an agent that can not only win the game but also have the highest number possible, demonstrating efficient learning and adaptation in a dynamic environment.

System Requirements Document

Functional Specifications

Sensors:

- **Game Grid State:** A 4x4 matrix representing the current configuration of the 2048 board. Each cell holds a tile value (e.g. 2, 4, 8...) or 0 if empty. This acts as the agent's observation of the environment.
- **Score Tracker:** The current cumulative score, which increases when the tiles are merged.
- **Game Status:** Indicates whether the game is ongoing or running as a terminal state flag.

Actuators:

- **Actions:** The agent can perform one of four discrete actions: `up`, `down`, `left`, or `right`, corresponding to swipe movements that alter the state of the board.

Reward Function:

- Positive reward proportional to the value of tiles merged in a move.
- Small penalty for invalid or ineffective moves (i.e., moves that do not change the board state).
- Additional bonus reward for reaching tile 2048.
- Terminal reward based on the final score or highest tile reached.

Use Cases

Use Case 1: Valid Movement and Merge

- **Context:** The agent observes two adjacent `2` tiles on the left side of the grid.
- **Action:** Executes the `left` move.
- **Effect:** The tiles merge into a `4`.
- **Reward:** +4 for the merge.
- **Adaptation:** The agent strengthens the association between similar board patterns and beneficial moves.

Use Case 2: Invalid Move

- **Context:** The agent attempts a `down` move when no tiles can shift in that direction.
- **Action:** Executes the `down` move.
- **Effect:** The board state remains unchanged.
- **Reward:** -1 penalty for wasting a move.
- **Adaptation:** The agent learns to avoid unproductive movements.

Use Case 3: Game Over

- **Context:** The board is full and merges are not possible.
- **Action:** Executes the `down` move.
- **Effect:** The environment signals a terminal state.
- **Reward:** Zero or negative reward depending on whether the target tile (e.g., 2048) was reached.
- **Adaptation:** The agent adjusts its long-term strategy to delay or prevent losing configurations.

High Level Architecture

Components Diagram

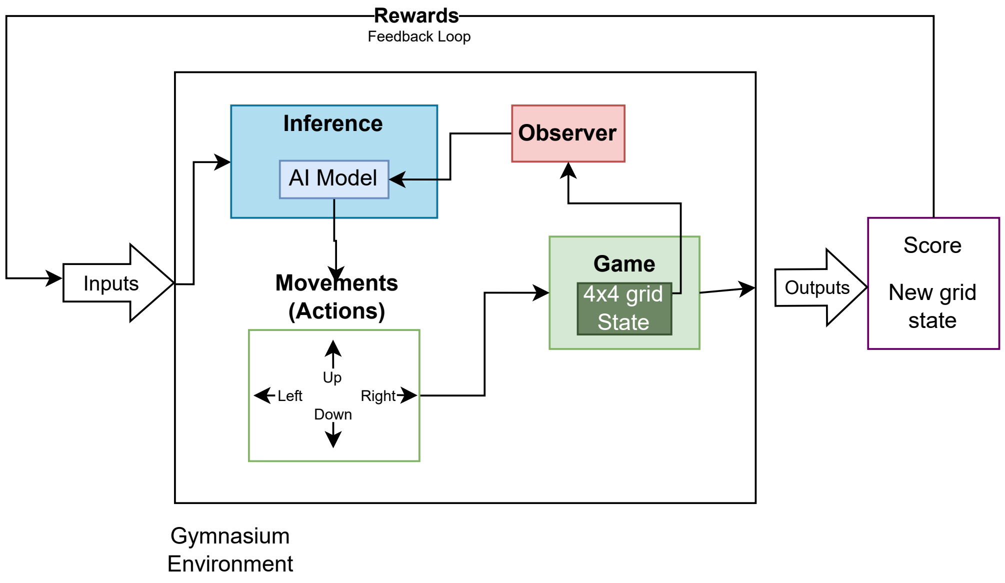

Figure 1: System Components Diagram

The previous diagram provides a representation of the system's components. It illustrates the elements, subsystems, and their interconnections. Specifically, it highlights how the AI agent interacts with the game environment.

Considering the above, feedback mechanisms can be discussed, but first, a review of the elements presented in the image will be provided.

- Inference is a subsystem responsible for all aspects related to the AI model and its processing.
- Movements (Actions) constitute an actuator, or action element, through which the Inference subsystem manipulates the 4x4 grid.
- The Observer is an element that monitors the game matrix within the Game subsystem and transmits this information to the Inference subsystem.

- The Game subsystem encompasses all components related to the game's logic, including aspects such as scoring and the generation of blocks with each movement.
- The score and new grid state represent the system's outputs, through which feedback is provided to the system; this has been addressed in detail previously.

Feedback loops

Feedback loops are essential for the AI model to learn about the game and its own behavior within it. For instance, due to the game's inherent nature, there are instances where a move is not possible because the board is full. The model must learn that in this state, or at this moment of the 4x4 grid, the given move cannot be executed and must instead be performed in another direction. Another scenario concerns the game's strategy: considering the game's essence, following certain movement patterns can be either beneficial or detrimental to the model's objective. Here,loops become critical, enabling the model to learn from past behavior and assess the most efficient option.

In summary, these cycles allow the AI model to learn about movements and, ideally, to evaluate its strategy for the subsequent move or board state. It is important to mention the environment of the system.

The environment of the game is a 4x4 grid that generates a block with number 2, enabling movement in all directions and introducing an element of randomness closely related to chaos theory. The objective of the AI agent is to achieve the highest possible score. When two tiles with the same number are joined, their values are summed, following a power of 2 progression.

These feedback loops are based on rewards obtained by the AI model, making it easier to refine its movements and optimize them with each simulation. This iterative process ensures that the agent progressively improves its strategy to achieve optimal performance.

The generation of blocks in every movement, and the random appearance of a block in each movement, is influenced by Chaos Theory. Consequently, feedback loops are constant and abundant within this highly dynamic environment.

Preliminary Implementation Outline

Selected Tools and Frameworks

The following tools will be used in the development of the agent:

- **Gymnasium:** To simulate the game environment and standardize agent-environment interaction.
- **Stable-Baselines3:** A reinforcement learning library with pre-implemented models such as Q-learning and DQN.
- **PyTorch:** To potentially build custom neural networks if a more tailored DQN is needed.
- **NumPy:** For matrix operations and board state calculations.

With these libraries and frameworks, we aim to develop a robust AI agent aligned with the course objectives.

Development Timeline

Week	Activity Description
1	Finalize system design, define requirements and functional architecture. Prepare environment specs.
2	Build the custom 2048 environment using Gymnasium. Define observation and action space.
3	Implement basic agent logic (random moves or heuristics) to validate environment response.
4	Integrate Q-learning algorithm. Start training with basic reward functions.
5	Analyze training results. Tweak hyperparameters (learning rate, epsilon, etc.).
6	Replace Q-learning with DQN. Set up neural network architecture for state evaluation.
7	Begin DQN training. Monitor agent behavior and learning progress.
8	Tune DQN architecture and rewards. Introduce replay buffer and target networks.
9	Add performance logging and early-stopping criteria. Begin evaluating agent consistency.
10	Integrate visualization tools (e.g., plot learning curve, game progress).
11	Refactor code, add modularity, and improve readability. Final training phase.
12	Final evaluation: analyze win rate, max tile, and generalization. Prepare report and documentation.

Table 1: 12-Week Development Timeline for the RL 2048 Agent

Conclusions

The design of a reinforcement learning agent for the game 2048 presents a rich environment for a adaptative enviroment. By identifying key components such as virtual sensors, actuators, and a reward function aligned with performance goals, we established the foundation for an intelligent agent capable of interacting dynamically with a non-deterministic environment.

The use for cybernetic principles, principaly feedback-loops play a very high important role in the agent to evaluate and adjust its strategy over time.

Ultimately, this project serves as a foundational step toward building an autonomous, adaptive system capable of learning optimal behaviors through iterative simulation and self-improvement.

References

2048game.com. (2025, Mar.). *2048 and math.* ([Online]. Available: `https://2048game.com/blog/2048/2048-and-math/`)

Amazon.com. (2025, Mar.). *What is reinforcement learning?* ([Online]. Available: `https://aws.amazon.com/what-is/reinforcement-learning/`)

Foundation, F. (n.d.). *Gymnasium.* Retrieved 2025-04-09, from `https://gymnasium.farama.org/` (Accessed: April 9, 2025)

Kondo, N., & Matsuzaki, K. (2019). Playing game 2048 with deep convolutional neural networks trained by supervised learning. *Journal of Information Processing*, *27*, 340–347.

LLC, G. (n.d.). *Tensorflow.* Retrieved 2025-04-09, from `https://www.tensorflow.org/` (Accessed: April 9, 2025)

PyTorch. (n.d.). *Pytorch.* Retrieved 2025-04-09, from `https://pytorch.org/` (Accessed: April 9, 2025)

Ravichandiran, S. (2020). *Hands-on reinforcement learning with python: Apply reinforcement learning techniques to real-world environments using tensorflow 2.0 and pytorch.* Packt Publishing.

Schnur, D. (n.d.). *The 4-step framework to solve almost any problem like top strategy consultants.* Retrieved 2025-04-09, from https://medium.com/better-humans/the-4-step-framework-to-solve-almost-any-problem-like-top-strategy-consultants-66bf9f11afbd (Accessed: April 9, 2025)

Supahub. (n.d.). *Ai feedback loop: What it is, how it works, and its importance.* Retrieved 2025-04-09, from https://supahub.com/glossary/ai-feedback-loop (Accessed: April 9, 2025)