STAT1840 Project Proposal
Winning at 2048 Using Reinforcement Learning

Eliot Atlani
Mathilde Cros

November 18, 2024

Abstract

This project explores reinforcement learning as a method for playing the game 2048. Implemented in Python, the game environment provides a basis for testing various RL algorithms inspired by the Harvard STAT1840 course and recent academic research. The aim is to develop an agent that can effectively learn and adapt to the game mechanics, achieving high scores by employing advanced RL techniques. Our starting point for the 2048 game layout is based on an open-source code tutorial from GeeksforGeeks [2]. By analyzing the performance of these algorithms in the 2048 environment, we hope to evaluate their effectiveness and uncover insights into strategic gameplay in high-dimensional, combinatorial action spaces.

1 Introduction

2048 is a tile-based, single-player puzzle game where the objective is to merge tiles by powers of two to reach a tile numbered 2048. From the fact that in recent work reinforcement learning has been capable of successfully playing various uncertainty games such as backgammon [6], card games [8], and go [5], we aim with this project to investigate its potential in creating an agent capable of learning optimal moves and maximizing scores for the 2048 game.

The game presents an interesting challenge for reinforcement learning due to its stochastic nature, high-dimensional state space, and dynamic board configuration. By applying RL techniques, we aim to discover effective strategies for achieving high tiles consistently and understanding the mechanics of decision-making in uncertain environments.
Our implementation of the project can be found in the GitHub repository RL-for-2048.

2 Related Work

We found inspiration for this project in two main papers tackling reinforcement learning applied to game-playing, specifically 2048. In “Playing 2048 With Reinforcement Learning” [4], they provides us with an overview of various RL approaches to tackle the 2048 game, focusing on methods such as Q-learning, Deep Q-Networks (DQNs), and beam search. Eventually, it reports that beam-search reaches 2048 28.5% of the time compared to what the created game revealed that only about 1% games out of hundreds million ever played have been won. In “What’s in a Game: Solving 2048 with Reinforcement Learning” [3], they investigate the Monte Carlo Tree Search algorithm and compare which of three different value functions to use as the evaluation policy to yield the largest percentage of games where the tile 2048 is achieved, the largest average sum of the final board, and the largest average merge score. Lastly, a good baseline work for our project would be inspired by Robert Xiao’s work in his github repository "AI for the 2048 game" [7], where he

implements expectimax optimization to search upwards of 10 million moves per second.

Since our work in class will have included the Monte Carlo Tree Search rather than these other methods, we will focus on improving the performance on the models around Goenawan et al.'s work [3].

3 2048 Game Definition and Layout

To implement the game logic, we based our code off of the tutorial from GeeksforGeeks [2], where some adjustments were needed to improve the visuals and certain rules to the game were omitted.
The basic rules and mechanics of the game are:

- Tiles with the same number merge when they collide, producing a tile with double the value.
- After each move, a new tile (with the number 2 or 4) appears at a random empty spot on the board. According to this implementation of the game [1], the probability to obtain a tile with the number 2 is 90 % and 4 is 10%.
- Players can swipe up, down, left, or right to shift all tiles in that direction, merging all cells where applicable.

4 Models and Analysis

4.1 Baseline Model

To establish a benchmark, our baseline model will simply be randomly choosing at each move whether to go up, down, left or right, which act as a control for evaluating RL-based models. Indeed, this provides us with a baseline score, where no "thought through decision" is being made, enabling us to compare the performance of more sophisticated RL algorithms.

We will then implement a second baseline model based on simple model making decisions based on commonly recommended by 2048 players rather than RL. These simple heuristics are:

- Attempting to keep the highest tile in one corner.
- Computing the sum of cells along rows/columns to prioritize moves increasing the overall board value.
- Applying penalties for moves that reduce the board's order to maintain the board's structure.

4.2 Defining the Reinforcement Learning Problem

Our model will treat the current configuration of the 2048 board as the *state* and each directional move as a potential *action*, and frame it as a Markov Decision Process (MDP) with discrete states and actions.

The **state space** is represented by a three-dimensional numpy array of size (16, 4, 4), that is 256 entries for each state representation. The second and third dimensions, representing a 4x4 grid, correspond to the tiles on the game board. The first dimension, with a size of 16, encodes the possible values each tile can hold, where each entry represents a number in the game that can either be 0 (indicating an empty tile) or a power of two. Specifically, a tile's value is given by 2^i where $0 \leq i < 16$. For example, if the current board configuration has a tile with the value 2 located at position (3, 2), the array entry at (1, 3, 2) is set to 1, while all other entries are set to 0.

The **action space** in 2048 consists of four directional moves that can be made from any given state: moving all tiles to the left, up, right, or down. However, not all moves are available in every state due to board

constraints. For example, if tiles are already aligned along one edge of the board, certain moves may be restricted. To handle this, we represent the available actions at each state with a dynamic list that reflects the current game constraints. For instance, in a state where only moves to the up and right are possible, the action list would include only those moves.

The **reward function** for each action is determined by the sum of the values on all tiles merged as a result of the move. For example, if two tiles with the number 8 merge alongside two tiles with the number 2, the reward for that action would be 20. This reward structure incentivizes the agent to create higher-value tiles through merges, aligning well with the overall objective of the game to reach a tile with a value of 2048.

4.3 Algorithms that we'll explore

From the exploration of the papers from the Related Works section, we will explore then evaluate the ability to achieve high scores and reach high tiles of the following RL algorithms to train our 2048-playing agent:

- **Policy Gradient Methods:** This method optimizes the policy without learning a value function and can learn a stochastic policy, which is a good thing given the random spawning of tile. Our attempt at improving the current state of the art methods is to **combine this with a reward shaping using new criteria or with Actor-Critic value function to reduce the variance**.
- **Monte Carlo Tree Search (MCTS):** This algorithm is a promising approach for solving finite-state games like 2048, where simulating potential future moves allows us to identify the optimal action. The strength of MCTS lies in its ability to explore and evaluate future game states effectively through simulations. In order to push further than the current approaches of this method, we will focus on **developing strategies to extend the depth of simulation as far as possible while minimizing computational overhead using new heuristics and/or pruning methods**.

4.4 Performance Evaluation

We want to assess the effectiveness of each RL model in achieving strategic gameplay, comparing them against our heuristic baseline. This analysis will provide insights into the potential of reinforcement learning for complex decision-making tasks in high-dimensional, uncertain environments like 2048.

The primary measure is the average score achieved by the agent over multiple games, which indicates its overall performance. We then track the highest tile reached in order to represent the agent's ability to make effective merges.

Finally, we will be tracking the percentage of games in which the agent achieves the 2048 tile, which reflects its success in meeting the game's objective. We will additionally keep track of the stability of each algorithm over time by analyzing how quickly the agent converges to an optimal policy and whether it maintains performance consistency across games.

References

[1] Gabriele Cirulli. 2048 game source code, 2014. URL `https://github.com/gabrielecirulli/2048/blob/master/js/game_manager.js#L71`. Accessed: 2024-11-18.

[2] GeeksforGeeks Contributors. 2048 game tutorial, 2023. URL `https://www.geeksforgeeks.org/2048-game-in-python/`. Accessed: 2024-11-18.

[3] Abner et al. Goenawan. What's in a game: Solving 2048 with reinforcement learning. *Stanford CS 229 Final Reports*, 2020. URL `https://web.stanford.edu/class/aa228/reports/2020/final41.pdf`.

[4] Yunpeng Li and Yifan Peng. Playing 2048 with reinforcement learning. *arXiv preprint arXiv:2110.10374*, 2021. URL https://arxiv.org/pdf/2110.10374.

[5] David et al. Silver. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. doi: 10.1038/nature24270.

[6] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995. URL https://www.bkgm.com/articles/tesauro/tdl.html.

[7] Robert Xiao. 2048 ai by nneonneo, 2015. URL https://github.com/nneonneo/2048-ai.

[8] Daochen et al. Zha. Rank the episodes: A simple approach to exploration for offline reinforcement learning. *arXiv preprint arXiv:1910.04376*, 2019. URL https://arxiv.org/abs/1910.04376.