AI Agent for 2048: A comparison between Brute Force Search and Q-Learning over an expanded game space

Ruben Mayer (rmayer99), Magdy Saleh (mksaleh), Jayden Navarro (jaynavar)

Introduction and Related Work

2048 is a popular game that went viral in 2015. While the mechanics of the game are simple, playing the game well can be quite challenging, as the space of potential moves is very large ($O(2^{20})$). As a result, there have been a wide range of attempts to create an AI agent that plays the game effectively.

One of the most popular methods for solving the game involves using a brute-force approach that searches the space of potential moves at a limited depth, and then uses heuristics to pick the best possible move.[1] A more sophisticated approach involves using reinforcement learning methods such as Temporal Difference Learning to model the space of potential moves.[2]

Research has shown that on a 4x4 grid, while the reinforcement learning based methods are the most effective (with the state of the art models reaching an average score of 600,000 points), the brute force approach can also yield respectable results (with one model earning an average of 300,000 points). There is little research that compares the performance of these two approaches as the state space grows. The goal of our project will be to see how the performance of reinforcement learning based models compare with a brute force approach to solving 2048 as the game grid gets larger, and the space of potential moves increases.

Specifically, for this project, we will implement a brute-force approach to playing 2048 grid by using Expectimax Optimization, as well as a Reinforcement Learning approach that uses a Q-Learning algorithm. We will subsequently compare the performance of these two algorithms on different sized grids with dimensions that range from 4x4 to 6x6.

Behaviour and Data examples

Inputs: 2048 game engine with an $n \times n$ sized grid, and a move time-limit of x miliseconds.

Output: Game score and largest tile reached

We have already built an engine to simulate the game on the command line and as discussed in the Baseline section below, we built different preliminary agents to play the games. Figure 1 shows an example image of the game being played on the command line in interactive mode.

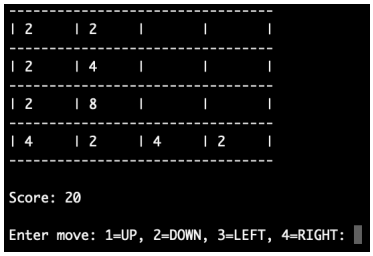

Figure 1: Example of the 2048 game engine

Baseline and Oracle

We developed three baseline players. The first performs completely random moves (*BaselineRandom*), the second tries to make moves that concentrate pieces in the top left corner (*BaselineCorner*), and the third attempts to maximize the score on every move (*BaselineGreedy*).

We ran one million simulations for each of these players on a 4x4 grid, and one thousand simulations on 4x4, 5x5, and 6x6 grids. The aforementioned trials yielded the following results:

Table 1: Score and Max Tile statistics for each agent on the 4x4 grid (1M Simulations)

	Max Score	Avg. Score	Stdev Score	Max Max Tile	Avg. Max Tile	Stdev Max Tile
Greedy	15460	3110	1534	1024	238	129
Corner	12396	2546	1224	1024	198	99
Random	6832	1095	535	512	107	55

Table 2: Average score for each agent on a 4x4, 5x5, and 6x6 grid (1K Simulations)

	4x4 Avg. Score	5x5 Avg. Score	6x6 Avg. Score
Greedy	3118	24533	265765
Corner	2503	13470	61466
Random	1100	7682	97946

As expected, *BaselineGreedy* performs the best out of the three across all metrics and board sizes, but it still does not come close to a good human player and is extremely far from the Oracle (*see next paragraph*). The reason why *BaselineCorner* performs better than *BaselineRandom* most likely stems from the fact that the corners approach is a human strategy when playing the game that intuitively works by stacking high value squares in one of the corners of the board. The random strategy will shuffle the tiles around more which will result in the game ending faster, and thus a lower score and max tile. Somewhat surprisingly, *BaselineRandom* performs better than *BaselineCorner* on the larger grid sizes, possibly due to the random shuffling causing more merges on a large board size and thus a higher score.

There is no simple Oracle that can solve 2048. While the game can be slightly simplified to a version where the player gets oracle access to the computer's moves (i.e. the future placement of the tiles), finding the optimal move in this simplified version has been proven to be $PSPACE - Complete$.[3] Therefore, instead of an oracle, we will calibrate the quality of our work by looking at previously published results. In particular, we know that the current upper bound for the average score achieved by an AI agent playing 2048 is 609,104.[4]

Challenges and Relevant Topics

The main challenge associated with designing an agent that plays 2048 involves the search of a large state space (ranging from $O(2^{20})$ to $O(2^{50})$ depending on grid size).

The most relevant topics for this project are Reinforcement Learning (Q-learning in particular) and Markov Decision Processes as well as dynamic programming in the context of search problems (which we will use for our brute force approach).

References

[1] Nneonneo. 2048 AI. *GitHub*.

[2] Kun Hao Yeh, I. Chen Wu, Chu Hsuan Hsueh, Chia Chuan Chang, Chao Chin Liang, and Han Chiang. Multistage temporal difference learning for 2048-like games. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(4):369–380, 2017.

[3] Rahul Mehta. 2048 IS (PSPACE) HARD, BUT SOMETIMES EASY. Technical report, 2014.

[4] Wojciech Jaśkowski. Mastering 2048 with Delayed Temporal Coherence Learning, Multi-Stage Weight Promotion, Redundant Encoding and Carousel Shaping. 2016.