CS 221 Project Report
Applying Varied Artificial Intelligence Techniques to Play 2048

Zhiyang He, Charles Lu, Stephen Ou
{hzyjerry,clu8,sdou}@stanford.edu

December 11th, 2015

1 Introduction

Released in 2014, the single-player puzzle game 2048 quickly became a worldwide sensation: in addition to hijacking classrooms and quickly occupying peoples phone screens, it also spawned countless spin-offs, strategy guides, and self-professed gurus. Meanwhile, the game offers a good platform to experiment and compare both traditional and cutting edge artificial intelligence techniques.

2048 is played on a 4 by 4 grid. Every turn, a random tile with a value 2 or 4 will appear on an empty cell. Then the player can choose to slide left, right, up, or down. All the tiles will slide all the way towards that direction. If any two tiles collide and have the same value, they will merge into one tile, with a new value being the sum of the two old tiles. Additionally, when two tiles collide, the score will be incremented by the value of the newly merged title. There are two ways to end the game. If the value of one tile reaches 2048, the game is considered a win (though the player can continue playing). Otherwise, if all the cells on the grid are occupied with mismatched tiles and no move is possible, the game is lost.

2 Task Definition

Our goal for this project is to build and investigate a number of models which play to maximize the game score at a reasonable runtime. The input-output behavior is as follows: on each turn, given the state of the 2048 game board (as well as move history when necessary), our models will return one of four moves (left, right, up, down) to attempt to maximize the score.

Next, we would like to discuss how we model the game of 2048 into specific game states. For each game state, there are two things to keep track of: the board and the score. The board is a four by four matrix, and each cell contains an integer (that is a power of 2) that indicates the current value of that cell. 0 is used to indicate unoccupied cells. The score variable is used to keep track of all the points received so far. The rule of the game of 2048 states that score is incremented by the sum of the two tiles when two tiles with the same value get merged.

There are two main methods that are available through the 2048 game state: `getLegalActions()` and `generateSuccessor()`. We will describe

each of them in details below.

`getLegalActions()`: If the current agent is the human, there are four possible moves. The human can swipe left, right, up, or down. There is one special case. A move is considered invalid if the board in the successor state is the same as the current state. For example, if the left three columns are all filled with tiles and they are have different values, swiping left is not a valid action because the successor state will not change. Next, if the current agent is the computer, there are maximum of sixteen possible moves. The computer can add a new tile with a value of 2 into an unoccupied cell.

`generateSuccessor()`: If the current agent is the human, all the tiles will slide in the direction specified. If the two neighboring tiles in that direction have the same value, they will collide and form a new tile with a new value that is the sum of the two old values. For example, if the board currently consists of only two tiles, both with a value of 2, at the bottom row. Swiping left will result in a merged tile with a value of 4, sitting in the bottom left corner. Next, if the current agent is the computer, a new tile with the value of 2 will be added at the specified cell given by the row and column number.

3 Infrastructure

We built a frontend interface for 2048 setup. We forked the original 2048 repository by Gabriele Cirulli and added our custom Javascript functions that talk to a lightweight Python server that computes the optimal move. The setup used an Flask, an open source Python web framework. We wrote a JavaScript function that serializes the current board as a string and passes it to the Flask server via an AJAX request. The server computes the optimal move using a specified approach, and returns a response back to the frontend. Then, a callback JavaScript function updates the board using the optimal move.

One problem we ran into while doing simulation is that the HTTP request and overhead of displaying the front-end visualization was a bottleneck in terms of speed. While it is interesting to see the game being solved in the real user interface, the speed becomes problematic when we wanted to do a lot of simulation to get abundant results.

Therefore, we built a more robust backend-only simulator so we can run the game quickly. It uses the logic in gameState.py to generate a successor based on a move picked by the agent specified. It starts from a board with only 1 tile and outputs the final score and number of moves once all tiles have been filled. Without involving the frontend which requires a lot of back and forth HTTP request, the backend simulator was be able to finish one full iteration of the game in few seconds.

To further increase the speed of the simulation, we decided to take advantages of the myth machines on Stanford campus. We parallelized the game simulation across 30 machines and ran them concurrently to get results. We were able to run 100 full iterations of the 2048 game across all 30 servers and gather 100 data points in less than 1 minute.

4 Approaches

Hello

Mininum Score	140
Maximum Score	1456
Mean Score	571.255
Median Score	546
Standard Deviation	280.527

Table 1: Random Agent

Mininum Score	64
Maximum Score	156
Mean Score	94.775
Median Score	88
Standard Deviation	18.791

Table 2: Up Down Agent

Mininum Score	656
Maximum Score	1708
Mean Score	946.457
Median Score	796
Standard Deviation	261.353

Table 3: Up Left Agent

Mininum Score	5580
Maximum Score	33900
Mean Score	20293.433
Median Score	20712
Standard Deviation	2956.264

Table 4: Expectimax Agent with depth 2 and snake evaluation function

Mininum Score	1496
Maximum Score	16364
Mean Score	7397.181
Median Score	7168
Standard Deviation	3250.412

Table 5: Expectimax Agent with depth 2 and game score evaluation function

Mininum Score	2600
Maximum Score	21036
Mean Score	13943.141
Median Score	14804
Standard Deviation	5279.458

Table 6: Minimax Agent with depth 2 and snake evaluation function

Mininum Score	2368
Maximum Score	24676
Mean Score	15064.333
Median Score	15660
Standard Deviation	4782.109

Table 7: Minimax with Alpha-Beta Pruning Agent with depth 2 and snake evaluation function

Mininum Score	81900
Maximum Score	172426
Mean Score	137770
Median Score	158984
Standard Deviation	48849.40651

Table 8: Expectimax Agent with depth 3 and snake evaluation function

5 Literature Review

6 Error Analysis

7 Future Works

References

[1] https://github.com/ov3y/2048-AI