Andy Wu
Perm: 7952278
CS165A
Professor Xifeng Yan

MP2 Report

Architecture:

For this project, I decided not to use a class but rather a collection of helper functions to build the NextMove function. Given a 4x4 grid and the step number of the game, the NextMove function utilizes a minimax algorithm to search through the best possible moves given the worst possible tile placements. I implemented partial game simulation through the moveUp, moveDown, moveLeft, and moveRight functions. Furthermore, a heuristics function was developed based on similar adjacent tile values and weighted matrix multiplication. The minimax function utilizes these helper functions in two separate functions: maxPlayer and minPlayer to simulate the adversarial aspect of minimax search. These functions combine to output the optimal move code based on minimax.

Search:

I utilized the minimax algorithm to search for the most optimal move based on the least optimal tile generation in 2048. I chose minimax because it is more optimal for worst case scenarios despite expectimax being more realistic to this game. The minimax algorithm used the player as the max player and the computer, who randomly generated 2 or 4 tiles, as the min player. With these parameters, the max player will play to attempt to resolve the worst case tile placements.

The minimax function utilizes a heuristics function to calculate a score for each possible move and tile placement. This program a weighted matrix:

[49, 36, 25, 16]
[36, 35, 24, 15]
[25, 24, 23, 14]
[16, 15, 14, 13]

This heuristic sums the product of the value of a tile and its corresponding weighted matrix value, giving more precedence to the top left corner in order to keep the largest numbers in the top left corner. I decided to utilize squared values for the heuristics function in order to further enforce the importance of keeping the largest values in the top-left. However, after extensive testing, I decided that rather than having diagonals be the same value, the similar values will be in an L-shape in order to encourage the option of merging. Furthermore, each step from the top and left edges will decrease the value of the weight by 1 in order to continue pushing the largest numbers to the top left. In addition to the weighted function, the heuristics function also awards points for having adjacent tiles the same (based on the value of the tiles) and deducts points for different tiles next to each other (based on the difference of the two tiles) in order to further encourage merging.

Because the algorithm uses minimax search, alpha-beta pruning is naturally implemented to cut unnecessary branches to free up memory and speed up processing speed. Furthermore, the “do nothing” case for the max player is omitted as an option because, after extensive testing, this option had an insignificant and sometimes negative effect on the success of the code. Therefore, discarding this case will significantly increase the speed of the code as well as avoid some cases where “do nothing” will result in the highest score despite leading to a terminal state.

Challenges:

One significant challenge was finding the right heuristics for the score function. Having too much of a difference in the weighted matrix may lead to the program to not merge tiles while too little of a difference in the weighted matrix may lead to the program ejecting the highest tile from the top left corner. Furthermore, adding the rewards and deductions for the adjacent tiles sometimes led to the program to “do nothing” because the rewards were too high for adjacent tiles. These issues were mainly found and solved through trial and error. Repeated testing of the weighted matrix greatly improved the heuristics. Additionally, removing the “do nothing” option removed the issue of awarding too many points for adjacent tiles.

Weaknesses:

One thing that may be weak is the algorithm used. Because minimax takes the worst case scenario, the algorithm may not accurately predict the tile placements based on chance. Implementing expectimax will greatly improve the accuracy of the simulations although allowing worst case scenarios to occur without consideration. Although both algorithms are viable in this case, minimax has its drawback of not being able to simulate chance like expectimax does.

Another weakness is the heuristics functions. Although thoroughly tested, there can still be improvements to this function. Some things that may help are considering more factors in the function such as points for more white spaces and a more strict weighted matrix tailored for combining tiles.

Furthermore, some functions could be optimized more for better runtime and therefore higher depth. This optimization could occur in min player’s children because the difference between inserting a 2-tile and a 4-tile does not seem to have a significant effect. Additionally, optimizing the many for-loops in the program could also greatly improve run time. Another idea is to use C++ to run the code even faster.