Shivaansh Kapoor
Email: shivaansh@ucsb.edu
Perm: 8542821

Architecture:
The architecture of my program consists of multiple functions and no classes. The first set of functions are to generate the initial grid (make_initial_grid) and randomly add a 2 or 4 to the grid (add_2_or_4). Then, I implemented the functions to move the grid (moveLeft, moveRight, moveUp, and moveDown) using a few helper functions. Lastly, the program contains a heuristic function, minimax_search function, and nextMove function to determine the next move based on the current grid.

Search:
The basis of the search that is done in my program is the heuristic function. I experimented with a few heuristic functions by using a combination of weights, the max value in the grid, and the actual values in the grid, but the optimal heuristic seemed to be a simple linear combination of the weights and the values in the cell. Here are the weights that I assigned to each cell:

12288	24576	49152	98304
6144	3072	1536	768
48	96	192	384
24	12	6	3

This approach tries to keep the highest element in the top right and the other elements next to it for optimal merging. Here is the heuristic function:

$$heuristic(grid,\ weights) = \sum_{i=0}^{3} \sum_{j=0}^{3} (grid[i][j] \times weights[i][j])$$

The actual search is done by a minimax tree with a depth of 2. First, it uses the initial grid and splits it off into the 4 grids of the potential moves it can do. Then, instead of making 2 min children for each empty cell (one for 2 and one for 4), we find the worst place that the game can put a 2 based on the weights (we don't care about 4 in this case because 2 will always lead to a worse heuristic) and use that as our singular min child. Lastly, we split each of the min children into the 4 grids of the potential moves it can do and calculate the heuristic of each of the leaves. Then we select the move with the max heuristic in the leaves as the next move.

This approach to search has its benefits, as we only need to store and search through 24 grids which decreases the runtime by a lot.

Challenges:
I ran into quite a few challenges while working on this project. The first challenge I ran into was picking a heuristic. I experimented with a few different heuristics, but the one that worked best was assigning weights which encouraged keeping the biggest blocks close to each other in a corner so that it would be easy to merge them. The second challenge I ran into was when I was trying to make 2 min children for each empty cell (the function is called minimax_search_allmins), but it was not working well in the game. I then pivoted to only considering the worst case for min and treating it like an actual game instead of relying on randomness and considering all the moves that min can make. This approach worked very well with my heuristic and played the game with the same strategy that I play it with.

Weaknesses:
The main weakness in my approach is that I only search to a depth of 2. This means that my program only searches 2 moves ahead. I want to experiment with searching deeper in the tree, especially since it won't take too much more time or space with my approach. This may result in an even more optimal result, and I may be able to get 4096 more often. To do this easily, I should probably convert my function to a recursive function which takes the depth as a parameter. This would allow me to experiment with all sorts of depths and pick which one works best for my heuristic.