Parallelized Expectiminimax with 2048

Matthew Broughton (mb4207), Kent Hall (kjh2166)

November 22, 2021

1 Background

2048 is a game where players move tiles valued at powers of 2 around a 4x4 grid while trying create a tile of value 2048. Every turn, a 2 or 4 valued tile appears on the board. The player can then shift the tiles in one of 4 directions. The tiles move until they hit the edge of the board or another tile. If two tiles collide and both are the same value, they merge and the resulting tile has double the value. In this fashion the player can manipulate the tiles to create as many high value tiles as possible. While the game is considered won when a tile of value 2048 is created, the player can continue playing the game until no possible moves are left.

2 Adversarial Algorithm

The optimal algorithm for an AI to run off of to try and win a game of 2048 is expectiminimax. Expectiminimax is a variation of minimax, which is a adversarial algorithm for choosing the next move in a game. This incorrectly assumes that the computer is intentionally placing new tiles in antagonistic ways against the player, but the algorithm still has the same result of picking the optimal move (most of the time). In minimax, the algorithm creates a decision tree where each level represents the player's or the computer's turn (alternating), and each node represents an action. Each node contains a value corresponding to how good the board is for the player, computed via some heuristic. The algorithm then finds the optimal move for the current turn by searching the tree (via DFS) for the move that not only maximizes the board's value now, but also maximizes it over the course of the next X turns while also minimizing the possible damage by the computer over the same time frame. Expectiminimax adds in an additional level to the tree that simulates random element, for our purposes it'll simulate whether a 2 or 4 tile is placed. Note that this search can be sped up some what with the help of Alpha/Beta pruning, which removes branches of the tree that cannot possibly be optimal from consideration as the algorithm runs.

3 Parallelization

The majority of the move decision process is comprised of a DFS for deciding the optimality of the 4 possible moves for the current turn. Because each recursive call of DFS is independent of all other calls on the same level, parallelization of expectiminimax should be straightforward$^{(TM)}$ as each recursive call can receive its own thread. We will likely make use of parBuffer to ensure the overhead of the threads does not end up dominating the runtime of the algorithm.