Project Proposal: 2048Solver

Alex Bala (ajb2320) & Michal Hajlasz (mah2350)

November 27, 2023

1 Introduction

The game 2048 is a solo puzzle game involving a 4x4 grid in which players merge numbered tiles. Tiles are numbered by powers of 2, and only tiles of the same value can merge to form a single tile with value equal to the sum of the merged tiles. A move consists of sliding all tiles in one direction (up, down, left, or right). After each move, the computer randomly places an additional 2 or 4-tile onto the board. The objective of the game is to create a 2048 tile, although the game goes on until the player is out of valid moves.

2 Algorithm

We will use an expectminimax algorithm that chooses the next player's move by (incorrectly) assuming that the computer is intentionally placing new tiles on the board in a way to stop the game as soon as possible. We will use multiple heuristics to form a utility function that the player is trying to maximize and the computer will attempt to minimize. These heuristics will be decided based on what we find works best (i.e. what results in the highest game score) and we will also employ (at least some) alpha-beta pruning to reduce computation time.

3 Parallelism

We will parallelize our 2048 solver as follows:

- **Concurrent State Evaluation**: After a move is made by the player or AI, the game states will be evaluated concurrently. When deciding which of the four possible moves to make, each of these choices are independent of each other and can be run in parallel.
- **Heuristic Evaluation**: Since we are using multiple heuristics to guide our minimax search, we will run each of these heuristic calculations in parallel.

- **Separate State Management**: We will use separate state management in our program. Each thread or process will operate on its independent copy of the game state. This approach eliminates the need for synchronization, as there are no shared data structures, thereby reducing the risk of data races. However, it increases memory usage as each parallel unit maintains its own state copy.
- **Dynamic Task Allocation**: If we have time, we may implement dynamic task allocation to adjust the parallel workload based on runtime performance metrics. This helps in optimizing the parallel processing based on the current computational load.
- **Parallelism with Alpha-Beta Pruning**: We will try to find a way to combine the sequential optimization technique (AB pruning) with parallel optimization techniques in order to come up with a strategy that combines both in an efficient manner.

4 Performance Evaluation

We will compare the solver's performance in both sequential and parallel configurations. Key metrics for evaluation will include the average time taken to complete games, the maximum score achieved, and the number of games successfully reaching the 2048 tile. Additionally, we will analyze the impact of parallelism on computation time and resource utilization, particularly focusing on how well the dynamic task allocation balances the workload across processors. The outcomes of this performance evaluation will help us decide the best heuristics, optimize our algorithm, and ensure that our solver is both effective in gameplay strategy and efficient in resource usage.