# COMS4995 Parallel Functional Programming Project

# 2048 Game Solver


# Linh Bui - ltb2128


2048 is a single-player puzzle game designed by Gabriele Cirulli. The game involves sliding numbered tiles on a grid to combine them into a tile with the number 2048. The game ends when the grid is full, and no more moves are possible. Players aim to achieve the highest score by strategically merging tiles.

### Game Mechanics

- **Grid Structure**: The game uses a 4x4 grid, initialized with two tiles, each showing a value of 2 or 4.
- **Movement**: Tiles can slide in four directions: up, down, left, or right. All tiles move simultaneously in the selected direction.
- **Tile Merging**: When two tiles of the same value collide during a move, they merge into a tile with double the value.
- **Tile Addition**: After each move, a new tile (value 2 or 4) is added to a random empty cell.
- **End Condition**: The game ends when no valid moves are possible.

## Algorithm Overview

The algorithm implemented simulates the gameplay of 2048 using artificial intelligence (AI). The AI leverages the **Expectimax algorithm** with several heuristic functions to evaluate board states and make optimal moves. Below, we break down the components:

#### a. Board Representation

- The board is represented as a 4x4 matrix of integers, where each cell contains the tile value or 0 for an empty cell.
- Functions are used to manage tile addition, movement, and merging.

#### b. Expectimax Algorithm

The Expectimax algorithm is a decision-making model used to evaluate possible future states of the game. It alternates between two types of nodes:

**1. Maximizing Nodes (AI's Turn)**:

- **Action**: The AI generates all possible moves (Up, Down, Left, Right) and evaluates the resulting board states.
- **Purpose**: The AI uses heuristic functions like monotonicity, smoothness, and weighted sum to compute scores for each resulting board state.
- **Outcome**: The AI selects the move that leads to the board state with the highest expected score, optimizing for long-term success.
- **Tree Generation**: These nodes expand fully, exploring all potential outcomes of the AI's moves.

**2. Chance Nodes (Board's Turn)**:

- **Action**: After the AI selects a move, the game board considers all empty cells and calculates the probabilities for placing a new tile (value 2 with 90% probability or value 4 with 10% probability).
- **Purpose**: This step models the game's randomness and uncertainty after each move.
- **Note**: These actions by the board are not actually performed in the game but are simulated solely for the AI's calculations.
- **Outcome**: For each empty cell, two potential states are considered: one with a tile of value 2 and another with a tile of value 4.
- **Tree Generation**: The algorithm calculates the expected score for each Chance Node by multiplying the heuristic scores of the resulting board states by their probabilities. No further branching occurs at these nodes.

**Randomness and Tree Generation**:

- The "random" tile placement happens exclusively at Chance Nodes.
- The game tree branches explicitly at Maximizing Nodes, where the AI explores all possible moves.
- Chance Nodes handle randomness probabilistically without further expansion.

- The depth of the tree determines how many future moves are evaluated. For example, a depth of 2 includes one AI move followed by one random tile placement.

**Sequential vs Parallel Execution**

1. **Sequential Execution**:

- The game tree is explored level by level, generating all possible moves for Maximizing Nodes and then computing expected scores for Chance Nodes sequentially.
- **Step 1**: For the Maximizing Nodes (AI's turn), the algorithm generates all potential moves (e.g., Up, Down, Left, Right) from the current board state.
    - For each move, the resulting board state is calculated, forming the next level of the tree.
- **Step 2**: For the Chance Nodes (Board's turn), the algorithm adds a new tile (value 2 with 90% probability or value 4 with 10% probability) to every empty cell on the resulting boards.
    - The probabilities of the tile placements are used to calculate the expected scores for these nodes. These scores are heuristic evaluations of the resulting board states after the tile is added. For each tile placement (value 2 or 4), the heuristic score measures how favorable the new board state is for the AI's strategy.
    - These expected scores are then associated with the AI's moves.
- **Step 3**: Once the scores for all Chance Nodes are computed, they are propagated back up the tree to determine the best move at the root level (AI's initial decision).

This approach ensures that all possible outcomes are explored and scored, but it can be computationally expensive as the tree depth increases.

2. **Parallel Execution**:

- The `parMap` function from Haskell's `Control.Parallel.Strategies` is used to evaluate multiple branches of the game tree simultaneously, dividing the computational workload across processor cores.
- Task Division for Maximizing Nodes (AI's Turn):
    - All potential moves (e.g., Up, Down, Left, Right) are treated as independent tasks.

    - For each move, the resulting board state is calculated and its heuristic score is evaluated in parallel.
    - Each branch of the tree corresponding to an AI move is handled as a separate computation, enabling efficient exploration of all possible outcomes.
- Task Division for Chance Nodes (Board's Turn Simulation in tree):
    - For each resulting board from the Maximizing Node phase, the addition of a new tile (value 2 or 4) to every empty cell is simulated.
    - These simulations for all empty cells are treated as independent tasks and evaluated in parallel.
    - The expected score for each Chance Node is computed by combining these parallel evaluations, reducing the time required for this probabilistic step.
- Overall Process:
    - Parallel execution ensures that both Maximizing Nodes and Chance Nodes are evaluated simultaneously across all branches of the game tree.
    - This parallelism reduces the computational overhead of evaluating deeper trees, as multiple processor cores can handle different parts of the tree concurrently.

### Example of Expectimax Decision Tree

Assume the current board state:

**4 4 0 0**

**8 8 16 16**

**32 64 128 256**

**512 1024 2 0**

**Initial Board**

**├── Move: Up (AI)**

**│ ├── Add Tile: 2 at Cell 1 (Board - Simulated)**

**│ │ ├── Move: Up (AI)**

**│ │ ├── Move: Down (AI)**

**│ │ ├── Move: Left (AI)**

**│ │ └── Move: Right (AI)**

**│ └── Add Tile: 4 at Cell 1 (Board - Simulated)**

**│ ├── Move: Up (AI)**

**│ ├── Move: Down (AI)**

**│ ├── Move: Left (AI)**

**│ └── Move: Right (AI)**

**├── Move: Down (AI)**

**│ ├── Add Tile: 2 at Cell 1 (Board - Simulated)**

**│ │ ├── Move: Up (AI)**

**│ │ ├── Move: Down (AI)**

**│ │ ├── Move: Left (AI)**

**│ │ └── Move: Right (AI)**

**│ └── Add Tile: 4 at Cell 1 (Board - Simulated)**

**│ ├── Move: Up (AI)**

**│ ├── Move: Down (AI)**

**│ ├── Move: Left (AI)**

**│ └── Move: Right (AI)**

**…**

At depth 3, the tree includes two levels of AI decisions and one level of board simulations (Chance Nodes). For each AI decision, the algorithm evaluates all possible board states resulting from potential tile additions and propagates heuristic scores back to the root. From then AI chooses the path with the highest score and continues the game. Only then Board will actually generate a random 2 or 4 according to rules.

## 5. Results and Observations

- Success Rate: The AI reaches 2048 in approximately 51.6% of 50 games.
- Workload is balanced, even for 10 cores.
- Depth is 8, so AI looks forward to 8 tree depth in advance before making decision, however consistency to reach 2048 is still average.

Example of final result of single game:

Final board after 957 moves:

[2048,4,8,2]

[32,16,2,0]

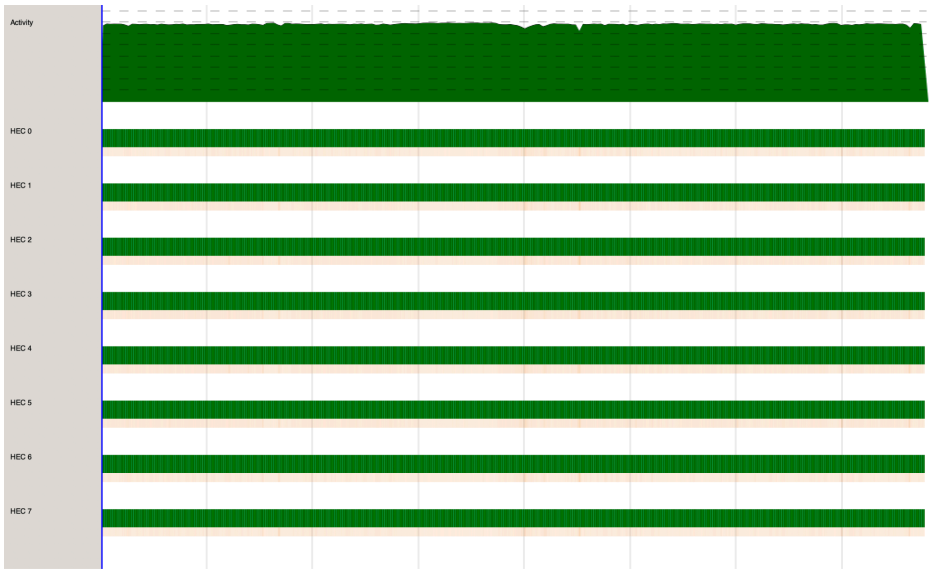
[4,8,0,0]

[0,4,0,0]

Result when running 10 games with different number of cores:

| Core | Moves | Total time | Time per move |
|---|---|---|---|
| 10 | 7917 | 239.7 | 0.03 |
| 8 | 8948 | 293.8 | 0.032 |
| 6 | 8674 | 310.7 | 0.035 |
| 4 | 9000 | 328.3 | 0.036 |
| 2 | 7922 | 330.6 | 0.041 |
| 1 | 8266 | 453.7 | 0.054 |

Threadscope when running with 10 threads:




## 6. Conclusion

This project demonstrates the potential of heuristic-based AI, combined with the Expectimax algorithm, in solving strategic problems like the 2048 game. By evaluating board states through advanced heuristics such as monotonicity, smoothness, and weighted sums, the AI is able to consistently make decisions that outperform typical human strategies. The parallel execution of game tree evaluations across multiple cores significantly improves performance, enabling deeper tree exploration and faster decision-making.

Despite its strengths, the current implementation shows limitations in late-game scenarios, where the success rate of reaching 2048 is moderate at 51.6%. Future improvements could focus on enhancing the consistency and scalability of the AI by implementing adaptive tree depths based on game progression, optimizing the heuristic evaluation for high-tile positions, and experimenting with reinforcement learning techniques. This combination of enhanced decision-making strategies and computational efficiency could make the AI even more robust and effective in handling the increasing complexity of late-game states.