CSD311: Artificial Intelligence

Project Report

Group25: 65536

Prof. Snehasis Mukherjee

Shubham Maurya – 2010110615

Samar Verma - 2110110930

Anjani Nadan Shukla – 2110111205

Suchin Kumar – 2110110524

Haradeep Pidaka – 2110110229

Prashast Upadhyay – 2110110386

Anant Agrawal - 2010110080

**Introduction:-**

2048, a puzzle game crafted by Gabriele Cirulli in 2014, has garnered global acclaim and gained popularity in the academia for its minimalist design, engaging gameplay and inherent problems it possesses which can be solved in the field of mathematics and computer sciences. This report examines the game's objective, mechanics, and the strategic challenges it presents to players. The seemingly simple task of combining tiles on a 4x4 grid belies the complexity that unfolds as players strive to reach the elusive 2048 tile. Join us in unravelling the layers of this captivating puzzle game where we try to solve it beyond the elusive 2048 tile, where strategic thinking, spatial awareness, and patience take centre stage.

**Problem Analysis:-**

* The game is played on a grid consisting of 16 tiles.
* At the start of each turn, a new tile with the value of either 2 or 4 appears on the board.
* Players can move the tiles by swiping left, right, up, or down to merge them into one tile.
* When two identical-valued tiles are merged, they combine into a new tile with double the value.
* The goal of the game is to create a tile with the value of 2048 as soon as possible.
* If there are no more valid moves available, the player loses the game.
* Each time a new tile is added to the board, it increases the difficulty level of the game.

**Solver Ideation:-**

We have initially started out by constructing the board, its representations, and writing conditions for valid moves. Then we researched about the algorithms that we can use. The major candidates were MCTS, Minimax and Expectimax. Initially we used MCTS, but it was not that successful, the highest tile we can achieved was 1024. Then we dwelled within Expectimax and Minimax. Due to the probabilistic nature of Expectimax we decided to go ahead with Minimax. In the context of solving 2048, Minimax can be applied to create an AI agent, but its direct application may be computationally expensive. Variations like alpha-beta pruning are often used to enhance efficiency.

**Data Structures Used:-**

* Game Board Representation:
  + Numpy arrays efficiently represent the 2048 game board.
* Available Moves Storage:
  + A list tracks available moves on the game board.
* Move-Result Pairs:
  + List of tuples stores pairs of moves and resulting game boards.
* Utility Evaluation:
  + Tuple structure represents utility evaluation components.
* Maximization Logic:
  + Maximization achieved through set variables and comparisons.
* Conditional Evaluation:
  + If statements control evaluation based on empty cells and search depth.
* Possible Tiles Representation:
  + List of tuples holds info on potential tiles for empty cells.
* Nested Loops for Iteration:
  + Nested loops facilitate systematic exploration of the search space**.**

**Goal Ordering:-**

In the context of the 2048 game, goal ordering involves strategically arranging tiles on the game grid to achieve the highest possible value. Unlike games such as Sokoban, where boxes must be maneuvered to specific goals, 2048 focuses on merging tiles by sliding them across the grid. The ultimate objective is to reach the coveted 2048 tile by combining matching-numbered tiles. Success in the game relies on thoughtful planning and efficient tile merging to continually double their values, working towards the numerical pinnacle while navigating the limited space of the game grid.

**Algorithms:-**

* Constants:
  + UP, DOWN, LEFT, RIGHT: Enumerated constants representing the four possible moves in the game.
* Class Definition:
  + AI: Class representing the AI player. It contains methods for evaluating the game board (eval\_board), maximizing the utility (maximize), and calculating the expected utility considering future possibilities (chance).
* Evaluation Function (eval\_board):
  + This function assesses the utility of the current game board. It considers factors such as the sum of squares of all tiles (big\_t), smoothness (similarity between adjacent tiles), and the number of empty cells.
  + Adjustable weights (empty\_w and smoothness\_w) are used to fine-tune the importance of empty cells and smoothness in the overall utility calculation.
  + Returns a tuple containing the total utility and individual utilities for empty cells, smoothness, and the sum of squares.
* Maximization Function (maximize):
  + Identifies the best move by considering all possible moves and their resulting boards.
  + Utilizes the chance function to calculate the expected utility of each potential move.
  + Returns the best move and its associated utility.
* Chance Function (chance):
  + Part of the Minimax algorithm, this function explores possible future game states to calculate the expected utility.
  + Determines if the search should be terminated based on the number of empty cells and the depth of the search (depth-limited search).
  + Considers the chances of a new tile being either a 2 or a 4.
  + Returns a tuple representing the sum of utilities for all possible outcomes.
* Algorithm Execution:
* Evaluation Phase (eval\_board):
  + Calculates the utility of the current board based on factors such as the sum of squares, smoothness, and the number of empty cells.
  + Adjusts the weights of these factors to influence the AI's decision-making.
* Maximization Phase (maximize):
  + Identifies the best move by evaluating all possible moves and their expected utilities.
  + Calls the chance function to explore potential future states and calculate utilities.
* Chance Phase (chance):
  + Explores the potential outcomes of the next move, considering the appearance of a new tile (either 2 or 4) in each empty cell.
  + Calculates the expected utility for each possibility and returns the sum of utilities.

**Code Flow:-**

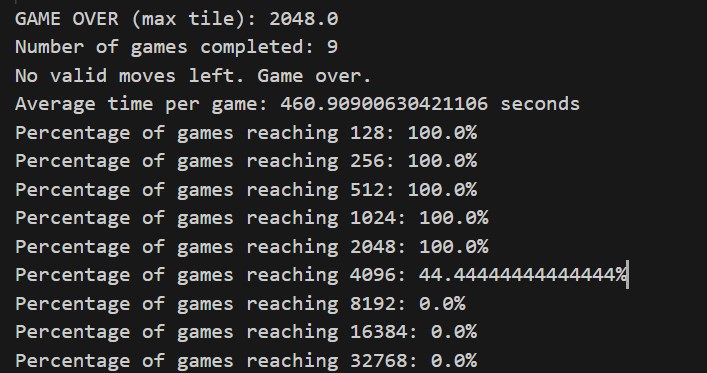
Develop a command-line interface (CLI) to present the current state of the game board. Enable the acceptance of artificial intelligence (AI) inputs for game control. Optimize decision-making by implementing algorithms such as Minimax, Monte Carlo Tree Search (MCTS), and Expectimax. Integrate AI-generated moves seamlessly into the game loop. Display the real-time status of the game board in the CLI or graphical user interface (GUI) while the game is in progress. Acknowledge and implement AI decisions, updating the board accordingly. Validate and respond to win or loss conditions. Generate a new random tile in an unoccupied area as part of the ongoing cycle. Develop a visual representation of the gaming board and establish a connection between the GUI and the underlying game logic. Present context-specific messages based on victory or defeat circumstances.

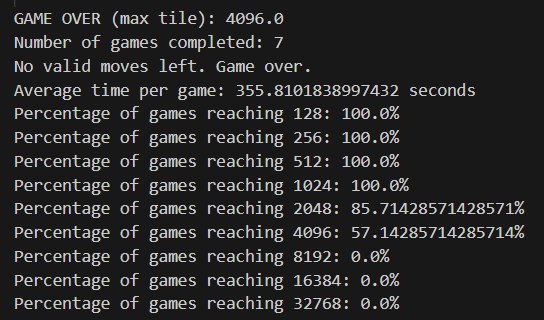
**Optimization of Memory use:-**

For the 2048 game implementation, we initially included variables such as the walls, goals, empty tiles, and dist\_to\_goal inside the AI class. However, to enhance memory efficiency, we opted to declare these variables outside the class as global variables. This adjustment significantly alleviated the memory footprint, especially when dealing with the most challenging depth, depth 5, which involves an extensive node expansion. With these optimizations, the program's memory usage was drastically reduced, enabling the successful execution of depth 5. This change proved crucial for achieving optimal memory utilization, ensuring the smooth functioning of the AI player in handling the complex computations associated with higher game levels.

**Our code on test levels:-**

We tested our code for 2 times with each testing a max time limit of one hour each.





**References: -**

<https://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048>

<https://www.cs.columbia.edu/~sedwards/classes/2021/4995-fall/reports/PMinimax.pdf>

<https://sandipanweb.wordpress.com/2017/03/06/using-minimax-with-alpha-beta-pruning-and-heuristic-evaluation-to-solve-2048-game-with-computer/>

<https://www.cs.columbia.edu/~sedwards/classes/2021/4995-fall/reports/PMinimax.pdf>