Artificial Intelligence
Checkpoint 1

Group_A1_113

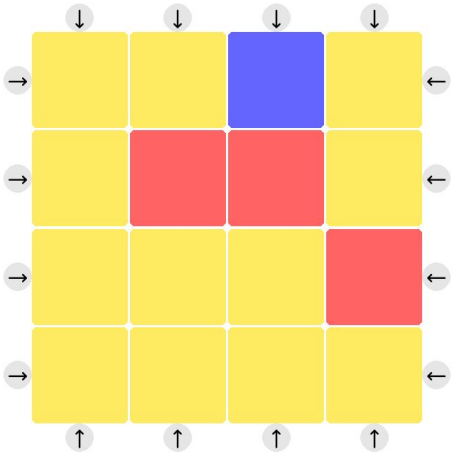
March, 2024

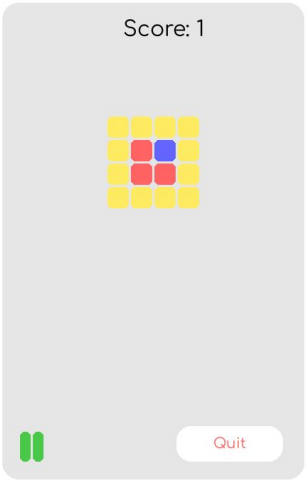
Daniel dos Santos Ferreira - up202108771
Francisco Cardoso - up202108793
Mansur Mustafin - up202102355

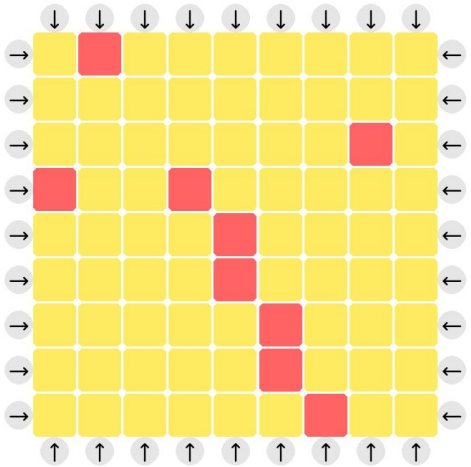
Cogito

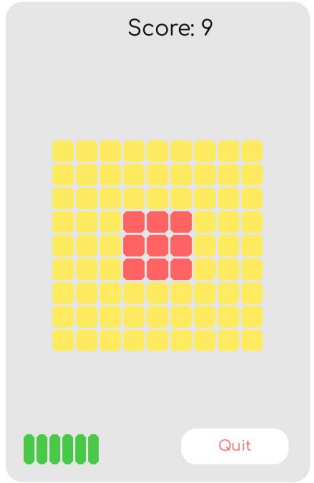
Game

Cogito is a 2D matching puzzle game played in a squared grid, typically 9x9 in size. Where the goal is to match our grid with a target grid by shifting its rows and columns.









Goal

Our goal is to implement several search algorithms and compare how well they perform.

Bibliography

- Game: Images, videos
- Gameplay: YouTube video
- PyGame documentation: Pygame.org
- PyGame tutorial: GeeksforGeeks
- Download Game: My Abandonware
- Manhattan and Linear Conflicts: Youtube video
- Artificial Intelligence A Modern Approach, 4th edition, Stuart Russell, Peter Norvig
- Academic paper with heuristics for the 24-puzzle game

Problem Formulation

State Representation

The state represents a symmetric board with a variable size, composed of the arrangement of pieces in each position on the grid. Each position can either hold a colored piece or remain empty.

Initial State

Any configuration of pieces in the board before starting the game.

Objective Test

Verify if the current grid matches the target's grid configuration.

Operators

Moves

move(idx, dir)- Shifts a row or a column one tile with wrapping.
where **idx** - index of moving row or column
dir - direction (up, down, left, right)

Each level has a twists in the moves, in some levels a move can shift more than one tile, and in others it can shift more than one row/column.

Cost : Each move (shift) has a cost of 1 move.

Preconditions: There are not any preconditions

Heuristics

Manhattan Distance

Calculates the sum of the manhattan distances (the distance between two points measured along axes at right angles) between each piece and its correct space. Boards with higher sums are considered worse than boards with lower sums.

Piece Mismatches

Calculates the number of pieces that are not in the correct space. Boards with higher sums are considered worse than boards with lower sums.

Line and Columns Difference

Compares every line and column of the current board with the target board, incrementing a counter with the difference in pieces from each line and column. Boards with higher sums are considered worse than boards with lower sums.

Correct Row Pattern

Checks the number of rows that only need to be shifted left or right to be completely correct. That is, rows that have the correct number of colored pieces and in the right configuration.

Work already carried out

Game

- Used python as the programming language;
- Implemented a graphical user interface for the game;
- Implemented a menu system;
- Implemented the game models (board, menus, levels, etc...), representing the board as a sparse matrix using a dictionary;
- Implemented the logic of the game (moves, winning and losing conditions, etc...);
- Level construction with the PyYAML module

AI

Implemented the following algorithms:

- DFS
- BFS
- Iterative Deepening Search
- Greedy, A*, Weighted A* with the “Piece mismatches”, “Line and Column Difference” and “Manhattan distance” heuristics;

We also have helper methods to calculate the time and memory spent by the AI when solving a level with a certain algorithm.