U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Match-the-Tiles
Artificial Intelligence - IART

Class 1 Group 12

Beatriz Costa Silva Mendes	up201806551@fe.up.pt
Nuno Guilherme Amaral Santos	up201405774@fe.up.pt
Telmo Alexandre Espírito Santo Baptista	up201806554@fe.up.pt

Explanation of the project to be performed

The theme of this project is the game Match-the-Tiles, it consists of a board of N by M, where N, M is greater than 1, 1 or more colored blocks and equal number of goals, one for each block, matching in color. The objective of the game is to put each colored block on top of a goal with the same color. Along the board there are tiles (represented with color grey) that act as obstacles and do not move. Collision is enabled and can happen between any type of tile but no movement after said collision. Plus, every time the player moves a block in a direction (up, right, down, left) all colored blocks move the maximum distance in that direction as demonstrated in figure 1. The level ends when all colored blocks are in the same position of their designated goals.

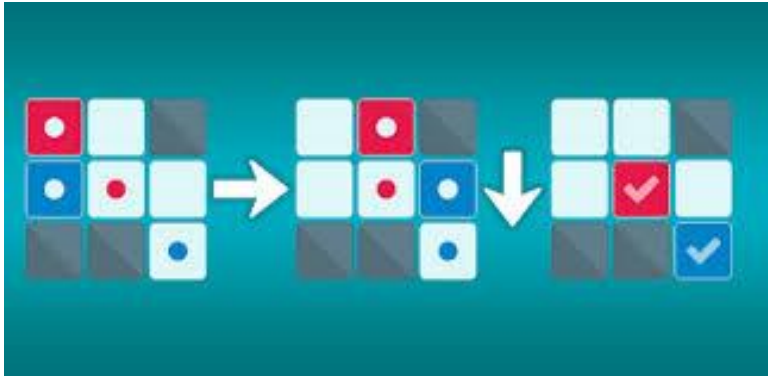

Figure 1 - Movement right and down.

The objective for this project is to find a way to evaluate a move and a game state so we can make consecutive moves and win the game with the least amount of movements possible. The evaluation of a move becomes more complex with the increase of colored blocks and size of the board, as well as the position of the goals, since it might require multiple colored blocks for one of them to stop in the goal.

Rules:

- All movable tiles (colored blocks) move as a group;
- You can only swipe up,down, left or right;
- Block and goal color must all match to beat the level;
- Tiles must all finish their movement in their designated goal to finish.

Formulation of the problem

State: Game state is represented by a NxM matrix, where each cell represents a Tile. A tile can be empty, a wall (obstacle), a goal, a colored block or a group consisting of a goal and a block overlapped.
Initial state: Generated game.
Successor Operators: Generates valid states that result from execution.There are four actions (move the colored blocks right, left, up and down).
Objective Test: Checks whether the state corresponds to the objective configuration (all the colored blocks are in their designated goals).
Solution Cost: Each move (step) costs 1, the cost of the solution being the number of moves to solve the problem.

Operators:

1- Move Colored Blocks to the left

- **Preconditions**: the colored block that will be moved can't have a wall directly to its left;
- **Effects**: the colored blocks move to the left until they are stopped by a wall, the board border or another colored block, in the later case the colored block only gets stopped after the colored block to its left stops its movement;
- **Costs**: 1

2- Move Colored Tiles to the right

- **Preconditions**: the colored block that will be moved can't have a wall directly to its right;
- **Effects**: the colored blocks move to the right until they are stopped by a wall, the board border or another colored block, in the later case the colored block only gets stopped after the colored block to its right stops its movement;
- **Costs**: 1

3- Move Colored Tiles upwards

- **Preconditions**: the colored block that will be moved can't have a wall directly on top of it;
- **Effects**: the colored blocks move upwards until they are stopped by a wall, the board border or another colored block, in the later case the colored block only gets stopped after the colored block directly on top of it stops its movement;
- **Costs**: 1

4- Move Colored Tiles downwards

- **Preconditions**: the colored block that will be moved can't have a wall directly below it;
- **Effects**: the colored blocks move downwards until they are stopped by a wall, the board border or another colored block, in the later case the colored block only gets stopped after the colored block directly below it stops its movement;
- **Costs**: 1

Formulation of the problem - Heuristic

For this game, the heuristic function turned out to be very hard to find because a certain move might put the colored block near the goal, but at the same time might be turning the solution harder to find due to, for example, the existence of obstacles. To finish a level of the game, the player has to pick a strategy to move the block to the designated goal, or the opposite, chose a path from the goal to that block. With the adding of blocks with same/different colors, it becomes difficult to evaluate the chosen strategy move by move, so the heuristic of calculating the distance between the block and corresponding goal is not the ideal.

To lessen the problems of efficiency related to the heuristic for this game we chose to come up with several functions to be compared between themselves and added together to form the ideal heuristic function.

- **Heuristic Function 1 - Distance**: calculate the distance between a colored block and its designated goals to decide the better way to achieve the solution. Since we have N blocks and N goals, each block may have multiple possible goals, and there are multiple blocks, so in this heuristic we give two extra functions. One will perform over the list of distances of a block to their goals, and the second will perform over the list of the results of the previous function of each block.
- **Heuristic Function 2 - Punitive**: if a colored block moves to a tile where the only possible move is to move in the opposite direction, the current move won't be considered ideal (unless the colored block is already in its designated goal). Keeping this in mind, the punitive heuristic function won't use this move to find the ideal path.

Implementation

For this project, we chose Python as the programming language, using Pygame for the graphic interface.

As data structures, we will be using a *GameState* class to manage the moves and effects to the board. *GameState* stores the matrix (2D array) representing the board, where each cell of the matrix is an object *Tile*. Additionally, the *GameState* also stores three Lists of coordinates, one for each of the important game tiles (walls, blocks, goals) in order to easily access their values on the matrix. *GameState* also provides functions to change the game state (*swipe_down*, *swipe_up*, *swipe_left*, *swipe_right*) that calculate the positions of each block after the movement and then calls the function *move* in order to change the values on the matrix. It also provides the function *is_game_over* that evaluates if the current game state represents the objective state.

The *Tile* class is used know if a certain cell of the *GameState* is a wall, a goal, a block or none of them. Note that a tile can be a goal and a block at the same time.

We created a class named *AStar*, where the A* algorithm is implemented. This algorithm will be used in our project to analyse the best solutions for the puzzles presented.

Our project is divided in 5 files:

- game.py - contains the main loop of the game
- game_state.py - contains the *GameState* class
- tile.py - contains the *Tile* class
- algorithms.py - contains the implementation of searching algorithms (such as A*, and later others)
- heuristics.py - contains the implementation of multiple heuristics

References

- **Google Play - Match the Tiles**
  - https://play.google.com/store/apps/details?id=net.bohush.match.tiles.color.puzzle&hl=pt_PT&gl=US
- **Match Tiles - Sliding Puzzle Game**
  - https://www.youtube.com/watch?v=_6Yzx5eVVUs
- **Heuristic Search, Chris Amato Northeastern University**
  - http://www.ccs.neu.edu/home/camato/5100/heuristic_search.pdf
- **Heuristics, from Amit's Thoughts on Pathfinding**
  - http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html