



I.A.

GRUPO 01: ANTHONY KEVIN, ANETE PEREIRA, EDUARDO DUARTE

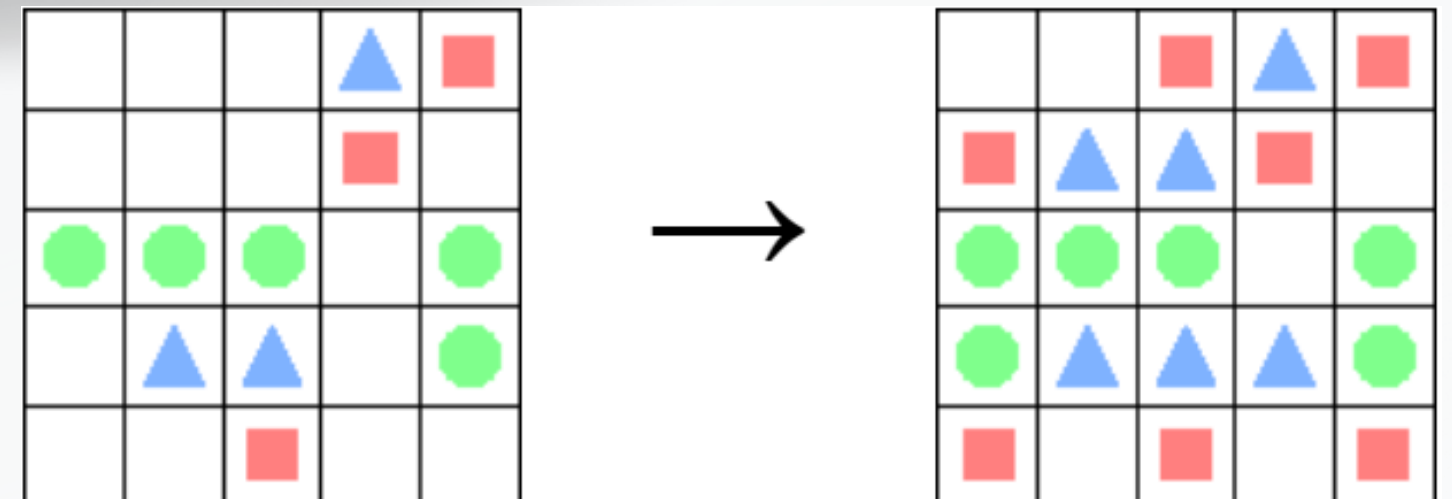
SPECIFICATION OF THE WORK TO BE PERFORMED



Symmetry Puzzles

The goal of the game is to attain symmetry in all rows and columns of the grid by filling in the empty spaces with shapes. To achieve this, the player needs to ensure that the shapes in each row and column form a palindrome, while ignoring the empty spaces. It's important to note that the player is allowed to add new shapes to the board, but they cannot remove or swap any existing ones.

OBS: Each puzzle has a unique solution.






FORMULATION OF THE PROBLEM AS A SEARCH PROBLEM

State representation: Represented as a 2D grid of dimensions $N \times N$, where each cell can be either empty or occupied by a puzzle piece.

Initial state: The initial state consists of a partially filled $N \times N$ grid of letters. The letters are randomly placed on the grid such that the symmetry condition is satisfied, i.e., the grid has at least one line of symmetry.

Objective test: The goal is to fill the grid with shapes in such a way that every row and column forms a palindrome, ignoring any empty cells. That is, the shapes in each row and column, read from left to right or top to bottom, form the same sequence of shapes when read in reverse order. If the grid satisfies this condition, the goal is reached



FORMULATION OF THE PROBLEM AS A SEARCH PROBLEM

Operators

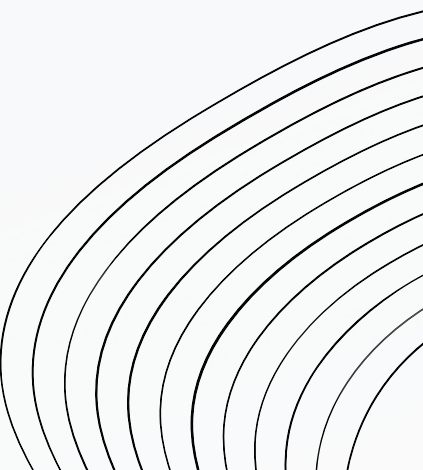
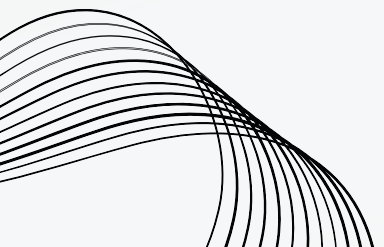
- **Add shape:** Adds a shape from the set of available shapes to an empty cell in the grid. The cost of this operator is 1.

Preconditions:

- **Add shape:** The cell in which the letter is being added must be empty.

Cost: Each action has a cost of 1

Heuristics/evaluation function: count the number of rows and columns that are currently palindromes and subtract this value from the total number of rows and columns in the grid. This provides an estimate of how far the current state is from the goal state.



IMPLEMENTATION WORK ALREADY CARRIED OUT

Programming language: Python

Development Environment: Anaconda 3, Visual Studio Code

Data structures:

- **board:** list of lists that represents the current state of the game board

Libraries:

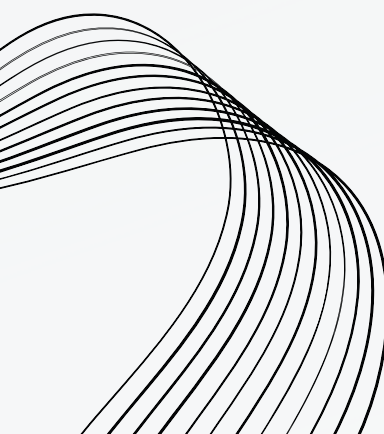
- **Tkinter:** used to create the game interface, including the game board and the buttons. Uses the Canvas widget to draw the game board, and the Button widget to create the buttons for starting a new game and exiting the application.

THE APPROACH FOR THE PROBLEM



Heuristic: After trying multiple options, our group decided to use an heuristic that splits the board into two halves and compare the different colors in each side and scores based on the differences. The score is calculated by adding both horizontal halves and vertical halves sub-scores.

Support functions:

- **Generate successors:** generates every board that can be created using available moves.
 - **Compare Halfs:** function that generates the sub-score explained above.
- 

IMPLEMENTED ALGORITHMS

Since the problem involves a single player game with unweighted moves, there were only a few options that made sense, so our group decided to implement the following algorithms:

- Breadth First Search (BFS)
- Depth First Search (DFS)
- Iterative Depth Search (IDS)
- Greedy Search
- A* Search

EXPERIMENTAL RESULTS

The following table consists of the average of several games from different board sizes that were solved by all implemented algorithms:

	3x3	4x4	5x5	6x6
BFS	422	584	2423	-
DFS	102	8374	-	-
GREEDY	389	227	514	245367
A*	14	29	1767	11232

CONCLUSIONS

- the A^* search algorithm is more efficient than DFS or BFS, especially when the puzzle involves weighted moves and/or multiple players, and mostly than Greedy Search. Therefore, it may be worth considering using the A^* search algorithm for solving more complex and challenging symmetric puzzles.
- it should be noted that DFS and BFS have limitations and may not be efficient in all cases. DFS can get stuck in an infinite branch or a loop without a cycle detection strategy, while BFS can get stuck in a very deep branch before exploring other shorter branches.
- It can be noted that in most cases, DFS is much slower than other search algorithms.

REFERENCES

<https://erich-friedman.github.io/puzzle/symmetry/>