Artificial Intelligence
1st Assignment

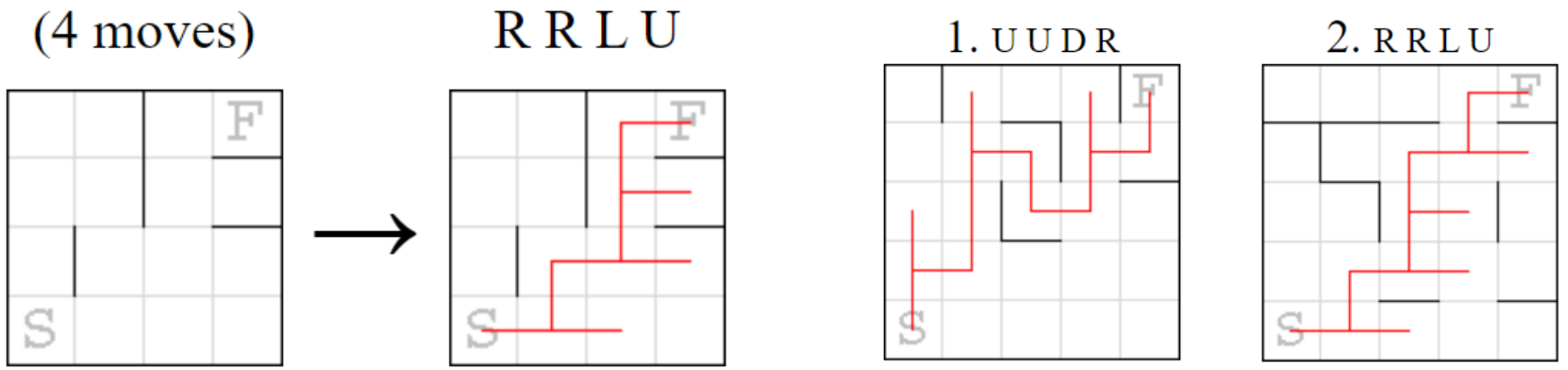
Juan Bellon, up201908142

Luísa Araújo, up201904996

Nuno Melo, up202003324

Robot Mazes

- Single player game.
- **Main goal**: Find the shortest sequence of necessary moves to reach the exit.
- From the departure point until the exit the robot is going to follow every move of the sequence in a cycle.
- If the robot bumps into a wall it does not move and passes to the next instruction (operator) of the sequence.
- Example:



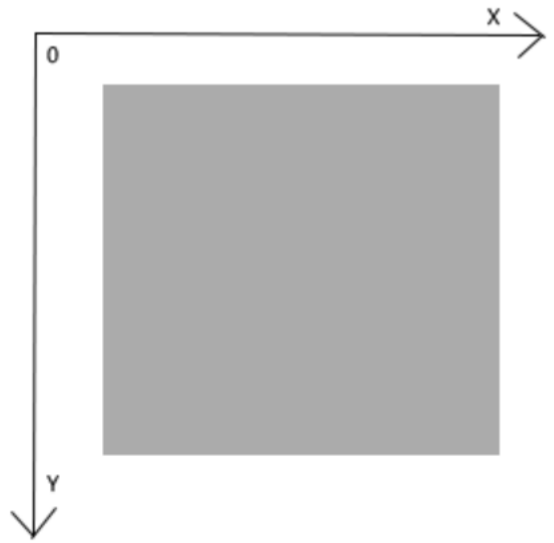
Formulation as a search problem (Part 1)

1. **State Representation**:

- StartPosition = (xStart,yStart)
- GoalPosition = (xGoal,yGoal)
- Maze represented as a 2D array (Lines x Columns). Each element of the array represents another array. Elem = [u, d, l, r] where each index represents a boolean that indicates if there is a wall or not in that cell (example: maze(1,3) = [1,0,0,1])
- RobotPosition = (xRobot, yRobot)
- sequenceSize -> Represents the length of the, so far, generated sequence.

2. **Initial State:** RobotPosition = StartPosition
3. **Objective Test:** RobotPosition = GoalPosition
4. **Heuristic Functions:** Manhattan, Euclidean and Chebyshev distances.
5. **Cost**: Length of the sequence of moves.

Formulation as a search problem (Part 2)



1. Operators:

- Let’s assume xSize and ySize are the labyrinth’s dimensions.
- Considering the xy axis has its origin in the superior left corner.

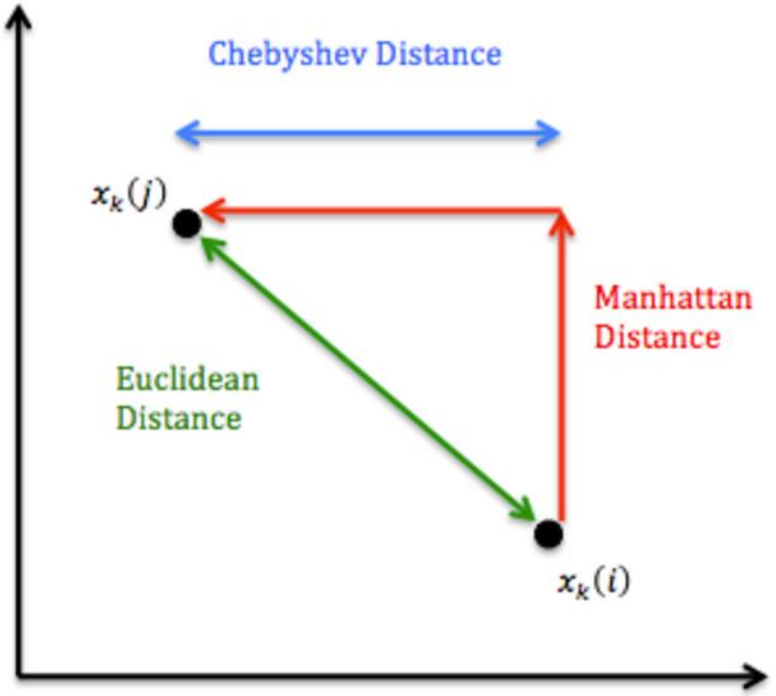
Name	Preconditions	Effect	Cost
UP	y>0 & place(x,y)!=1	y = y-1	Number of times that appears in the sequence
DOWN	y<ySize & place(x,y)!=1	y = y+1	Number of times that appears in the sequence
LEFT	x>0 & place(x,y)!=1	x = x-1	Number of times that appears in the sequence
RIGHT	x<xSize & place(x,y)!=1	x = x+1	Number of times that appears in the sequence

Implementation

- **Programming language**: Python.
- **Development Environment**: Visual Studio Code and Pycharm.
- **Main Libraries:** Pygame for the graphical interface, Time and MatPlotLib for the experimental results, IterTools for generating sequences.
- **Data Structures**:

1. Maze - Three dimensions array, for example: [[[1,0,0,1],[1,0,1,1]],[[1,0,0,1],[1,1,0,1]]]
2. Robot, start and goal positions as arrays, for example: (x,y) = [1,0]
3. For storing the generated sequences, a list is used as a queue (FIFO – first in first out)

Approach



- **Heuristics:**
- Manhattan distance

manDist = dist(final, robot) = | xFinal – xRobot | + | yFinal – yRobot |

- Euclidian distance

eucDist = dist(final, robot) = $\sqrt{((xFinal - xRobot)^2+(yFinal – yRobot)^2}$

- Shebyshev distance

shebyDist = dist(final, robot) = max(| xFinal – xRobot |, | yFinal – yRobot |)

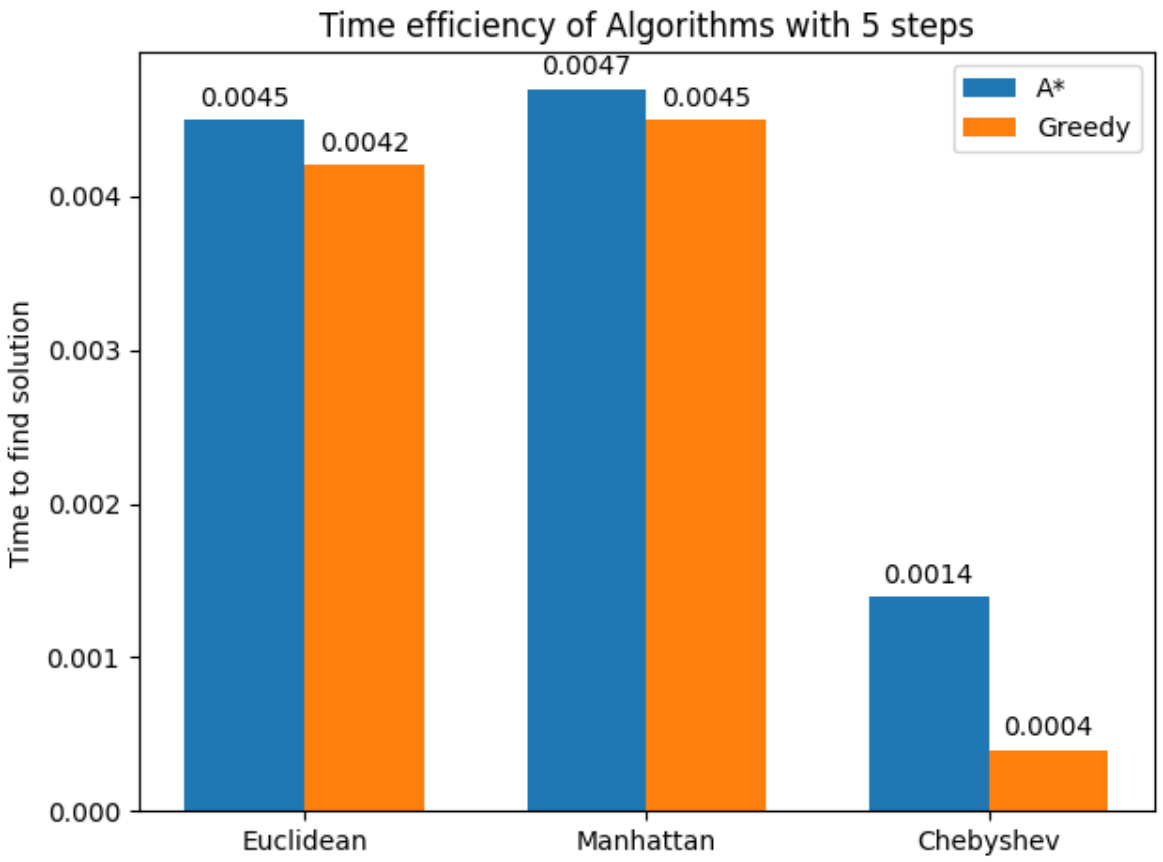
Implemented Algorithms

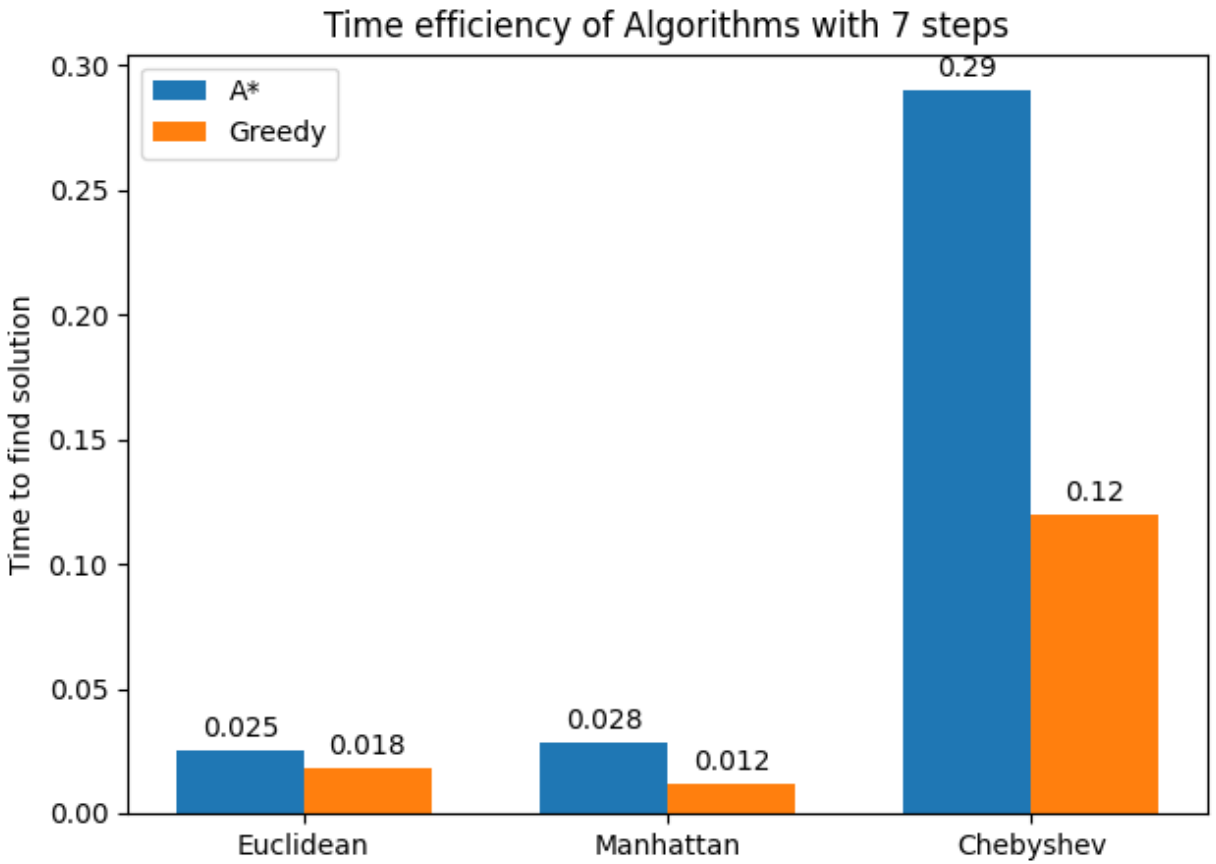
- **Breadth-First Search –** Nodes at lowest depth are expanded first.
- **Depth-First Search –** Always expand one of the deepest nodes in the tree.
- **Greedy Algorithm –** Expands the node that appears (according to a heuristic) the closest to the solution.
- **A* (A-Star) –** Combines the greedy with UCS (uniform cost search) minimizing the sum of the path already carried out with the minimum expected until the solution (cost + heuristic).

- **Note:** At first, Uniform Cost Search was implemented but later deleted, since when the cost is equal to the depth of the node, Depth-First Search is the same as Uniform Cost Search, which is the case of this problem.

Experimental Results

- **Comparing heuristics**

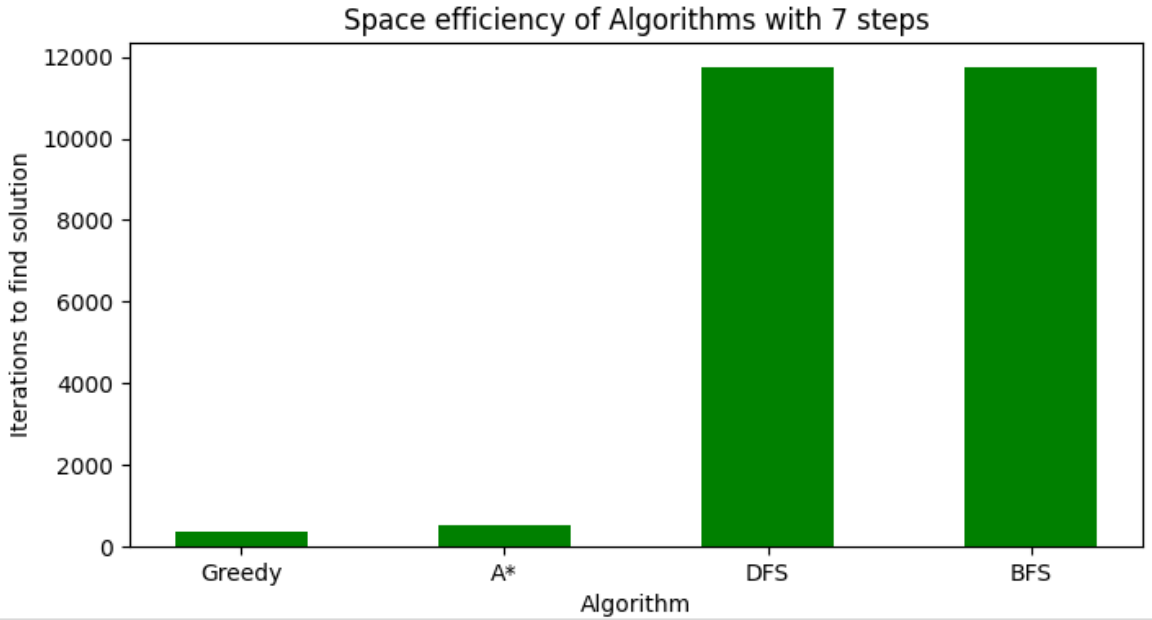


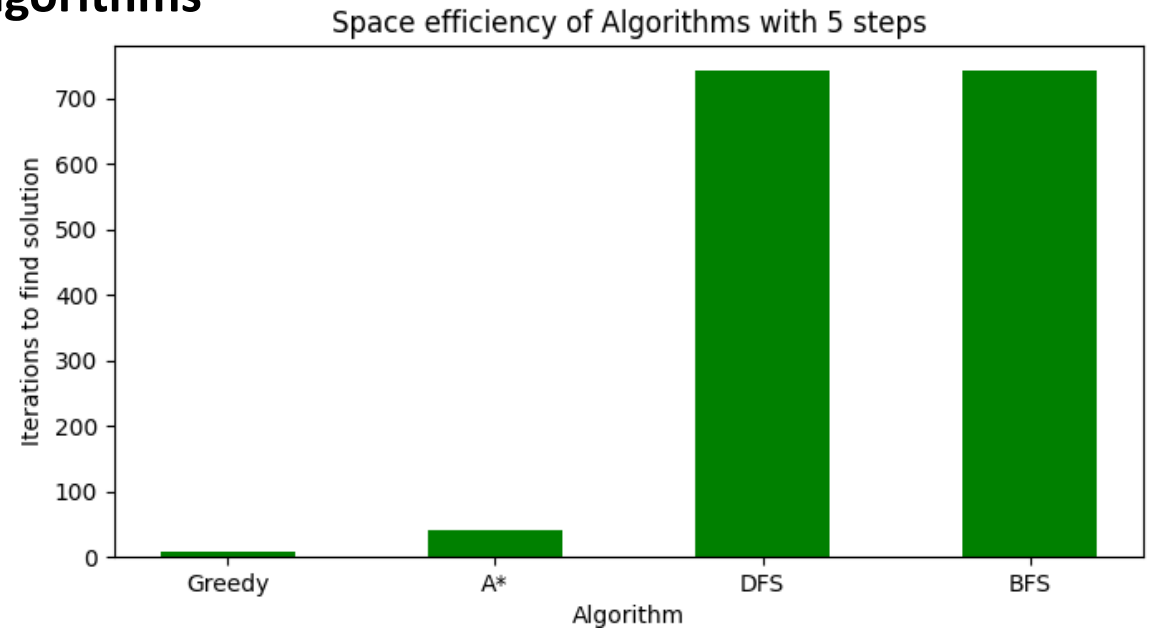


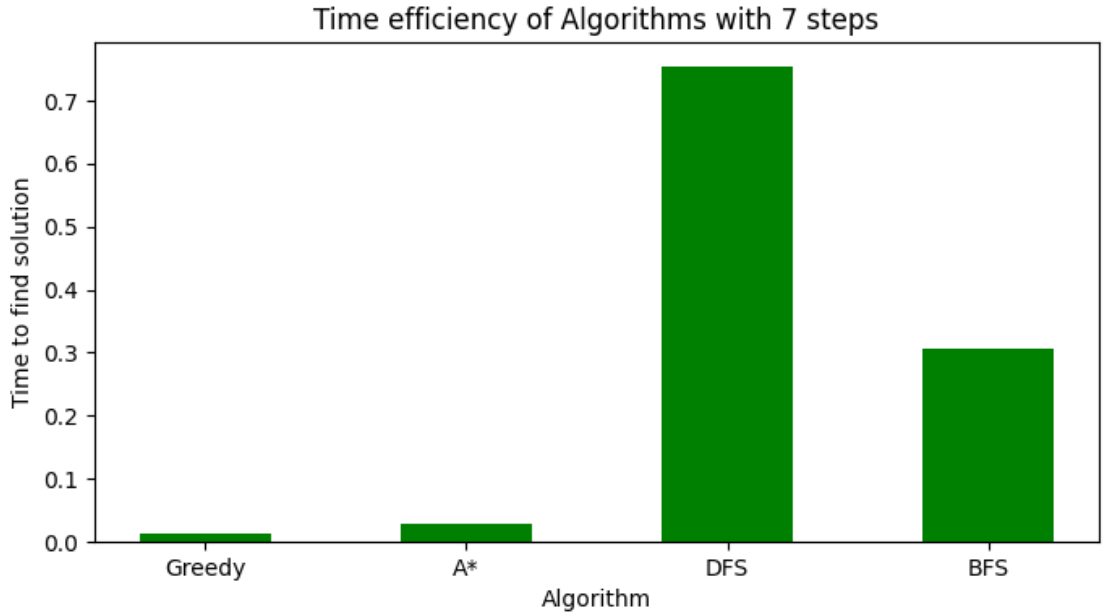
Note: Steps are moves in the sequence

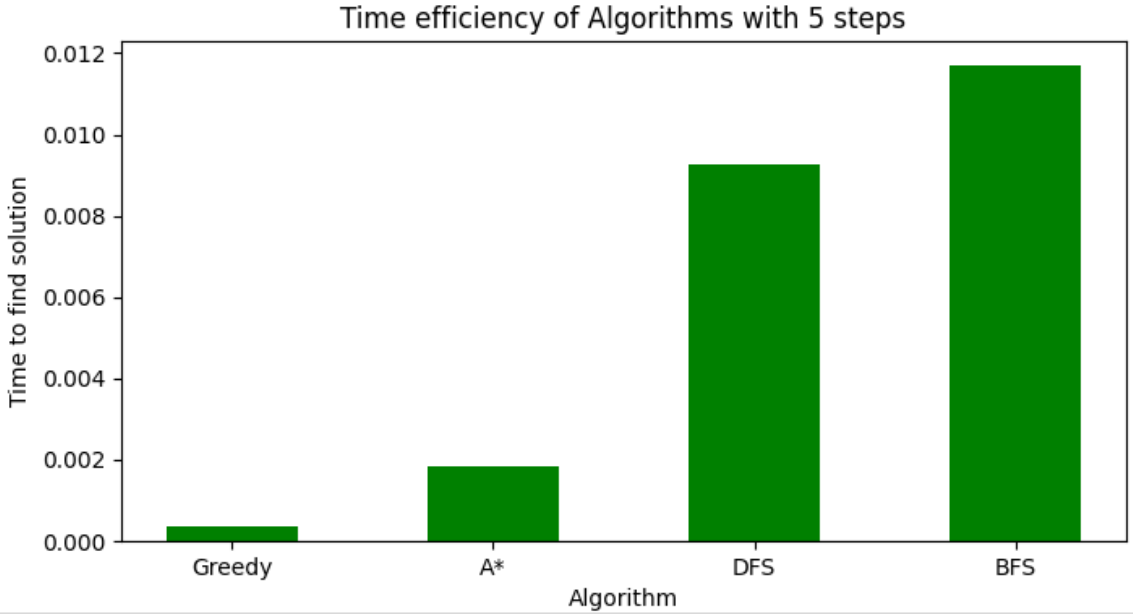
Experimental Results

- **Comparing algorithms**









Conclusions

For the following conclusions we use execution time and space as evaluators. The faster the program (or algorithms), the better.

- **Heuristic Functions conclusions:** When testing and comparing heuristics, two different mazes were used. These had different solution's (sequences) sizes: 5 and 7. For the shorter solution, Chebyshev distance worked better than the others and even though the difference between Euclidean and Manhattan distance in this scenario weren't as different, the Euclidean one still proved to be better. For the longer solution, Manhattan and Euclidean distance worked much better than the Chebyshev distance, being Manhattan distance the best heuristic, for both algorithms (A* and greedy).
- **Search Methods conclusions**: As expected, BFS and DFS were the worst algorithms when it comes to the mentioned evaluators. Still, time-wise BFS proved to be significantly better than DFS for longer sequences (7) and worse than DFS for shorter sequences (5). Overall, greedy obtained the best results in both sequences and according to both criteria (space and time). Even though A* is better than BFS and DFS, it's a little bit worse than greedy in all scenarios.

References

- **Problem Mazes:** https://erich-friedman.github.io/puzzle/robot/
- **Information about some Heuristics:** https://www.omnicalculator.com/math/manhattan-distance
- **Pygame documentation:** https://www.pygame.org/docs/
- **Search Methods:**

https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

https://towardsdatascience.com/understanding-a-path-algorithms-and-implementation-with-python-4d8458d6ccc7

Overall search:

- https://mat.uab.cat/~alseda/MasterOpt/AStar-Algorithm.pdf
- http://bryukh.com/labyrinth-algorithms/