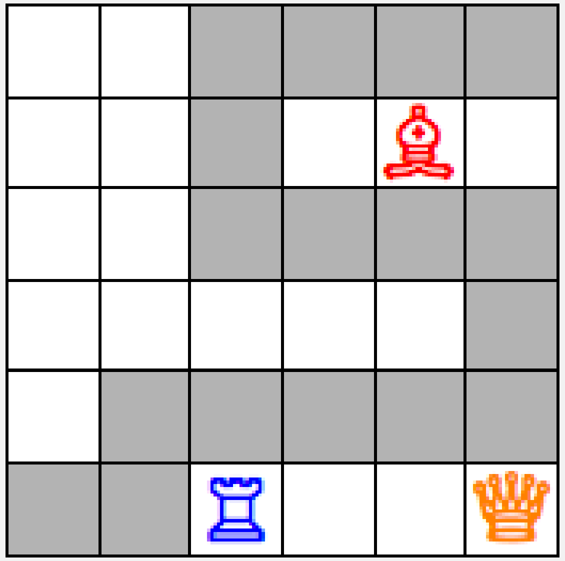
CHESS SNAKE PUZZLES

Group:
Carolina Cintra – up201906845
Marta Mariz – up201907020
Sara Sá – up201804838

DEFINITION OF THE GAME



- Chess Snake Puzzles is a puzzle created by Erich Friedman, inspired by Snake and including rules from Chess.
- The board can be either 5x5 or 6x6, includes 2 or 3 chess pieces that can't be moved, and space so the snake can move to the end point.
- The rules are the following:
  1. The snake can only move orthogonally.
  2. The snake cannot cross a chess piece.
  3. The snake cannot touch itself (even diagonally).
  4. In the result, each piece must attack an equal number of segments of the snake.

FORMULATION OF THE SEARCH PROBLEM

- Levels of difficulty
  1. 5X5 - 2 Chess pieces
  2. 6x6 - 3 Chess pieces
- Representation of the game state
  - The board for each level of difficulty is stored in a dictionary in python language
  - Each piece as a bitmap with the positions where it can attack the snake
  - The Snake itself has a bitmap as well
- Operators
  1. Up
  2. Down
  3. Left
  4. Right
- Heuristics

1. Distance from the snake’s head to the finish point.
2. Absolute difference between the maximum and minimum number of times a piece attacks a segment of the snake

IMPLEMENTED SEARCH ALGORITMS

- BFS (Breadth first search).
- Uniform Cost.
- Greedy Search.
- A* search with the heuristic - Distance from the snake’s head to the finish point.
- A* search with the heuristic - Absolute difference between the maximum and minimum number of times a piece attacks a segment of the snake.

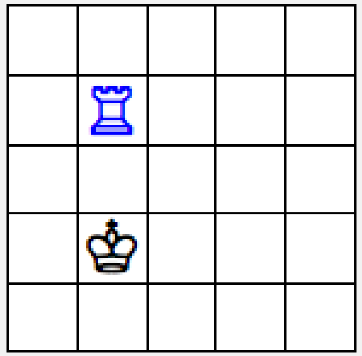
HEURISTICS

- **Manhattan distance from the current position of the snake to the end**
  - This heuristic is admissible because it never overestimates the cost to get to the final state, it often underestimates because it does not take into account the number of attacks rule.
- **Absolute of the difference between the number of attacks of the piece with the most attacks and the piece with least**
  - This heuristic is also admissible because to get to the final state it is always necessary for all to pieces to have the same number of attacks. We thought about doing the sum of the difference between all the pieces, but this one would no longer be admissible because different pieces can attack the same slot therefor there would be cases where the function would overestimate the cost to the end.

EXPERIMENTED RESULTS – EASY PUZZLE

Board1: [['T', 1, 1], ['K', 1, 3]]

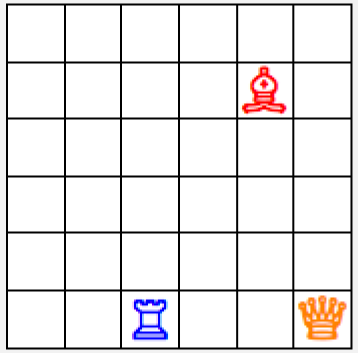
Search Problem	Number of Nodes	Time of Execution	Depth	Disk Usage
BFS	259	0.0074176788330078125	12	usage(total=146899341312, used=46535917568, free=92829929472)
Uniform Cost	263	0.009547233581542969	12	usage(total=146899341312, used=46535921664, free=92829925376)
Greedy Search	280	0.009185314178466797	12	usage(total=146899341312, used=46535925760, free=92829921280)
A*Search-Manhattan	189	0.008191823959350586	12	usage(total=146899341312, used=46535925760, free=92829921280)
A*Search-Difference between attacks	176	0.1064174175262451 2	12	usage(total=146899341312, used=46535925760, free=92829921280)



EXPERIMENTED RESULTS – HARD PUZZLE

Board11: [['B', 4, 1], ['T', 1, 5], ['Q', 5, 5]]

Search Problem	Number of Nodes	Time of Execution	Depth	Disk Usage
BFS	965	0.031042337417602540	12	usage(total=146899341312, used=46535999488, free=92829847552)
Uniform Cost	1039	0.06488490104675293	12	usage(total=146899341312, used=46535999488, free=92829847552)
Greedy Search	520	0.020253658294677734	12	usage(total=146899341312, used=46535999488, free=92829847552)
A*Search-Manhattan	338	0.02003335952758789	12	usage(total=146899341312, used=46536003584, free=92829843456)
A*Search-Difference between attacks	547	3.4627976417541504	12	usage(total=146899341312, used=46536003584, free=92829843456)



CONCLUSIONS ABOUT THE RESULTS

- We found that the most efficient method was A* search, specifically with the manhattan distance between the current and the final point. It was one of the fastest in both situations and it also had to explore a small number of nodes to get to the solution.
- The second heuristic was good in terms of number of nodes explored, but costly in time. We think that this could be caused by the sequential comparison between bitmaps that had information of the attacks.
- The uniform cost search was overall the most costful in terms of number of nodes explored, specially in the hard puzzle, which had na impact in the time of execution.
- In the easy puzzle, the greedy algorithm wasn't that efficient either, which could mean the heuristic function was not so fitting to a smaller board.

CONCLUSION

- With this project we learned more about the implementation of the different types of search algorithms, either the uniformed as well as the informed search algorithms.
- Also we learned about heuristics and how can we can make them admissible – we concluded that a good heuristic function isn't always obvious.
- As this was a complicated search problem that was affected by different aspects of the game – the path of the snake, the final point and the attacks of the chess pieces – conciliating these in na heuristic function and a formalized search problem was challenging.

REFERENCES

- https://pedros.works/chess-snake?W=4&L=s0n5b8f11&A=Erich_Friedman&N=2&T=Set_1&U=https://erich-friedman.github.io/puzzle/snake/
- https://erich-friedman.github.io/puzzle/snake/
- https://realpython.com/pygame-a-primer/
- https://www.101computing.net/getting-started-with-pygame/
- https://coderslegacy.com/python/pygame-font/