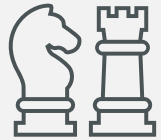
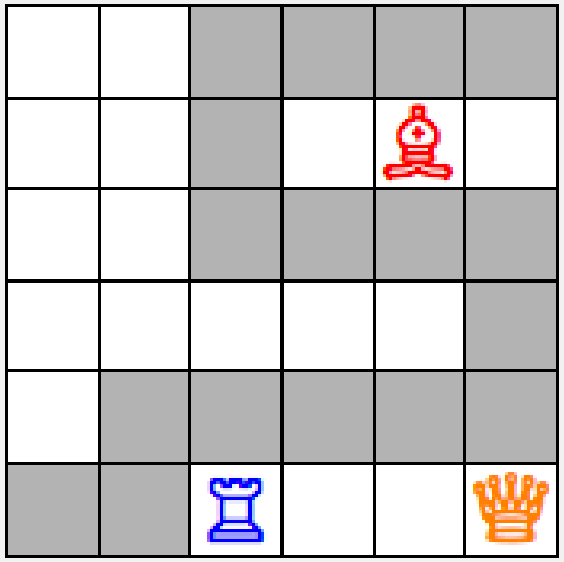


CHESS SNAKE PUZZLES

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.

REFERENCES

- https://pedros.works/chess-snake?W=4&L=s0n5b8f11&A=Erich_Friedman&N=2&T=Set_I&U=https://erich-friedman.github.io/puzzle/snake/
- https://erich-friedman.github.io/puzzle/snake/
- https://realpython.com/pygame-a-primer/

FORMULATION OF THE SEARCH PROBLEM

- Levels of difficulty
  1. 5X5 - 2 Chess pieces
  2. 6x6 - 3 Chess pieces
- Representation of the game state

  Matrix of objects in which we locate the Chess pieces – those have a vector of positions that they can eat.
- 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 number of times each piece eats a segment of the snake

OPERATIONS, PRECONDITIONS, EFFECTS, COSTS

Operations	Preconditions	Effects	Cost
Up	Y>0 & M[X, Y-2] ≠ snake & M[X-1, Y-1] ≠ snake & M[X+1, Y-1] ≠ snake & M[X-1, Y-2] ≠ snake & M[X+1, Y-2] ≠ snake & M[X, Y-1] = empty	M[X,Y-1] = snake Y = Y - 1 Update each piece with the number of times it attacks and if it does, remove that position from the vector Vector with the positions of the pieces/ pointer to the object piece	1
Down	Y<5 & M[X, Y+2] ≠ snake & M[X-1, Y+1] ≠ snake & M[X+1, Y+1] ≠ snake & M[X-1, Y+2] ≠ snake & M[X+1, Y+2] ≠ snake & M[X, Y+1] = empty	M[X,Y+1] = snake Y = Y + 1 Update each piece with the number of times it attacks and if it does, remove that position from the vector Vector with the positions of the pieces/ pointer to the object piece	1
Left	X>0 & M[X-2, Y] ≠ snake & M[X-1, Y-1] ≠ snake & M[X-1, Y+1] ≠ snake & M[X-2, Y-1] ≠ snake & M[X-2, Y+1] ≠ snake & M[X-1, Y] = empty	M[X-1,Y] = snake X = X - 1 Update each piece with the number of times it attacks and if it does, remove that position from the vector Vector with the positions of the pieces/ pointer to the object piece	1
Right	X<5 & M[X+2, Y] ≠ snake & M[X+2, Y-1] ≠ snake & M[X+2, Y+1] ≠ snake & M[X+1, Y-1] ≠ snake & M[X+1, Y+1] ≠ snake & M[X+1, Y] = empty	M[X+1,Y] = snake X = X + 1 Update each piece with the number of times it attacks and if it does, remove that position from the vector Vector with the positions of the pieces/ pointer to the object piece	1

WORK IMPLEMENTED

```
class Tower(Piece):
    def __init__(self, col, line, totcol, totline):
        super().__init__(col, line, totcol, totline)
        super().perpendicularAttack()


class Bishop(Piece):
    def __init__(self, col, line, totcol, totline):
        super().__init__(col, line, totcol, totline)
        super().diagonalAttack()


class Queen(Piece):
    def __init__(self, col, line, totcol, totline):
        super().__init__(col, line, totcol, totline)
        super().diagonalAttack()
        super().perpendicularAttack()


class Horse(Piece):
    def __init__(self, col, line, totcol, totline):
        super().__init__(col, line, totcol, totline)

        l = self._line
        c = self._col
        if (l > 0):
            l -= 1
```

- The project is being developed in Python 3 and the development environment chosen was Visual Studio Code
- The data structures used so far are a list of lists (a matrix that represents the board), lists of tuples (to save the positions that each piece can attack)
- We already implemented the board and the classes for each type of chess piece and their attack strategies
- We plan on using PyGame in the development of the GUI