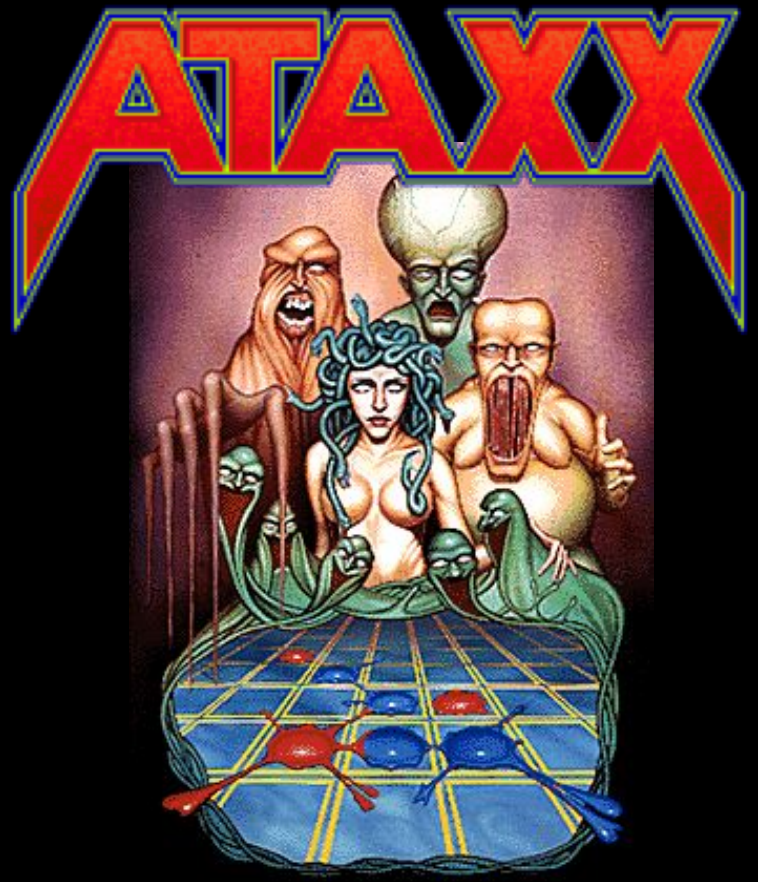
Adversarial Search Methods: Minimax for Playing an Attaxx Game

Pratical assignment developed for EAIDC Class



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

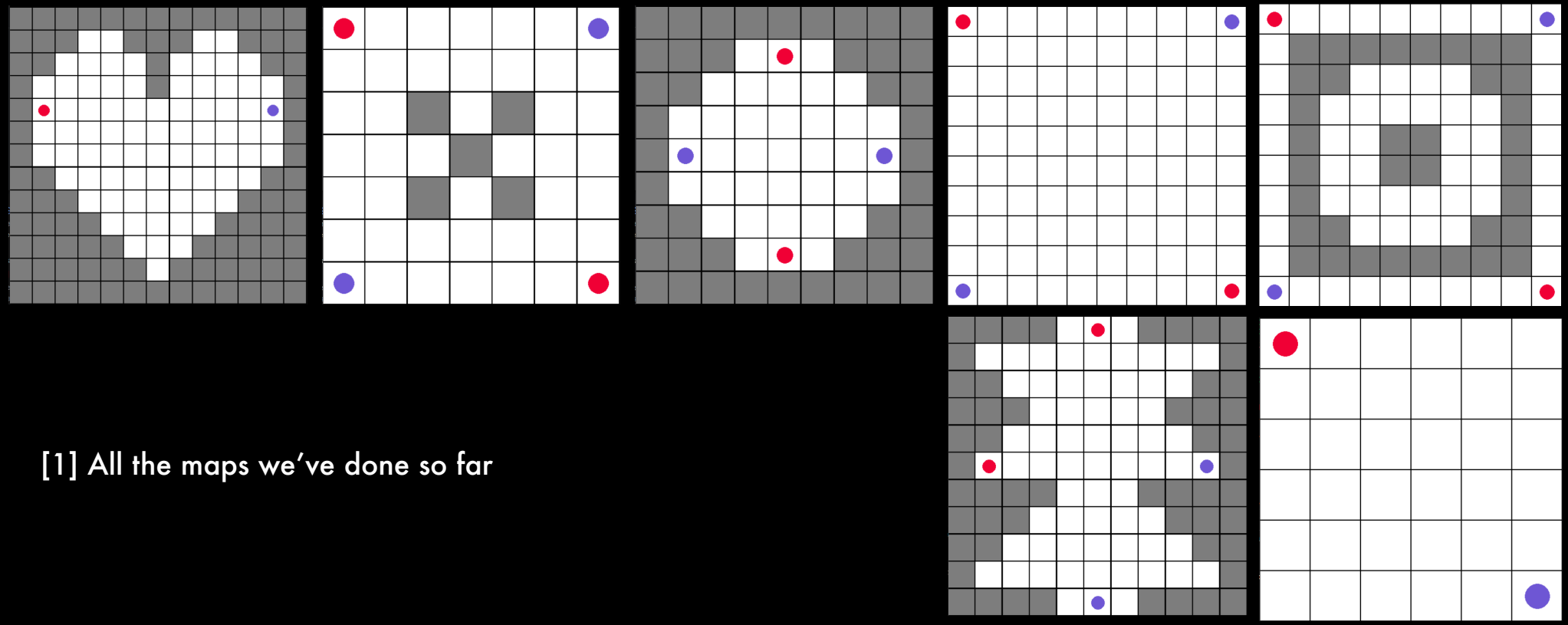
U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Inês Cardoso, Pedro Sousa, Class PL1

How the work will be developed:

- Firstly, we start by formulating our problem,then we begin to develop the base code of the game and the different types of algorithms (the main algorithm we'll be using in this Project will be the Minmax), lastly we compare data regarding the effiency of the various algorithms (Human vs Computer and Computer vs Computer) and analyze the results.



[1] All the maps we've done so far

Formulation of the problem:

- Initial state: The selected board;
- Objective State: When the Board is full or one of the players has zero pieces left;
- Operators:

Movement	Pre Conditions	effects	Cost
(X - 1 , Y - 1)	The square must be empty and must not be a wall		Depends on the heuristic
(X , Y - 1)			
(X + 1 , Y - 1)		Add a piece to the chosen square.	
(X - 1, Y)			
(X + 1, Y)		In case there's a piece of the opponent player adjacent to the chosen square, said piece becomes yours.	
(X - 1, Y + 1)			
(X , Y + 1)			
(X + 1 , Y + 1)			
(X - 2 , Y - 2)			
(X - 2 , Y)			
(X - 2 , Y + 2)			
(X , Y - 2)		Moves the piece to the chosen square.	
(X , Y + 2)			
(X + 2 , Y - 2)		In case there's a piece of the opponent player adjacent to the chosen square, said piece becomes yours.	
(X + 2 , Y)			
(X + 2 , Y + 2)			

Algorithms implemented:

- Random: Plays a random move;

```
def algo_random():
    return (random.randint(1, 10))
```

- Greedy: Plays the move which makes the piece difference bigger;

```
def algo_greedy():
    salt = random.random()
    return (conta_pecas(movimento.jog) - conta_pecas(troca_jog(movimento.jog))+salt)
```

- Center Control (CC): Focuses on dominating the center;

```
def algo_centercontrol():
    salt = random.random()
    yc = abs(gamestate.N / 2 - movimento.yf)
    xc = abs(gamestate.N / 2 - movimento. xf)
    score = 100 - (yc + xc) + 2*conta_pecas(movimento.jog) + salt
    return score
```

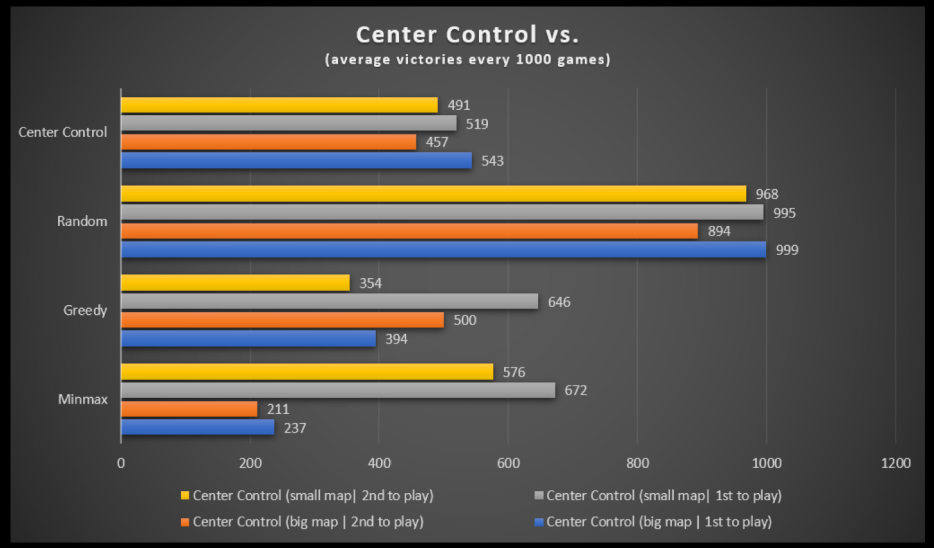
- MinMax: paired with the Greedy heuristic, it does a depth-first search, until a chosen depth or if an objective state is reached, in order to find the best move.

```
def algo_minmax(depth, minimizer, alfa, beta):
    if depth == 3 or fim_jogo == -1:
        return (algo_greedy() * (-1))

    if minimizer:
        movimento.jog = minmaxmov.min
        value = +1000
        for yi in range(gamestate.N):
            for xi in range(gamestate.N):
                if gamestate.tabuleiro[yi][xi] == movimento.jog:
                    for k in range(0, gamestate.N):
                        for l in range(0, gamestate.N):
                            movimento.yi = yi
                            movimento.xi = xi
                            movimento.yf = l
                            movimento.xf = k
                            if movimento_valido(movimento):
                                temp = copy.deepcopy(gamestate.tabuleiro)
                                executa_movimento()
                                evaluation = algo_minmax(depth +1, False, alfa, beta)
                                gamestate.tabuleiro = temp
                                value = min(value, evaluation)
                                beta = min(beta, evaluation)
                                if beta <= alfa:
                                    break
```

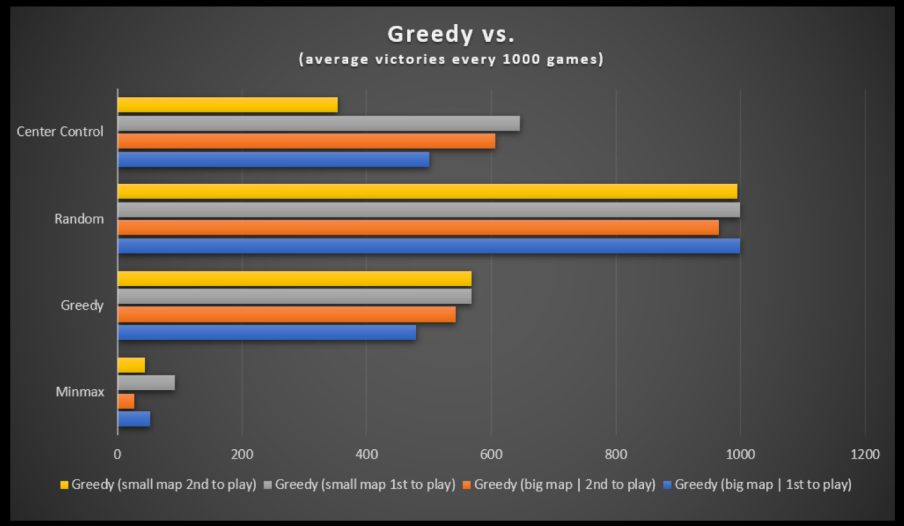
```
        movimento.jog = minmaxmov.max
        return value
    else:
        movimento.jog = minmaxmov.max
        value = -1000
        for yi in range(gamestate.N):
            for xi in range(gamestate.N):
                if gamestate.tabuleiro[yi][xi] == movimento.jog:
                    for k in range(0, gamestate.N):
                        for l in range(0, gamestate.N):
                            movimento.yi = yi
                            movimento.xi = xi
                            movimento.yf = l
                            movimento.xf = k
                            if movimento_valido(movimento):
                                temp = copy.deepcopy(gamestate.tabuleiro)
                                executa_movimento()
                                evaluation = algo_minmax(depth +1, True, alfa, beta)
                                gamestate.tabuleiro = temp
                                value = max(value, evaluation)
                                alfa = max(alfa, evaluation)
                                if beta <= alfa:
                                    break
        movimento.jog = minmaxmov.min
        return value
```

Experimental results:

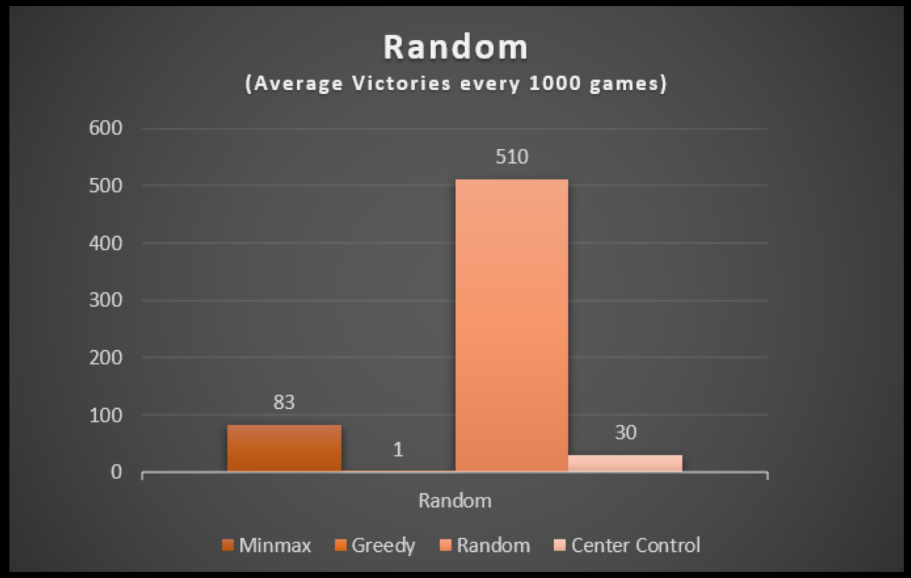


- As we can see, Minmax excelled at winning against the greedy and the random heuristic (hitting a 90% winrate), and when played against itself going first, it had a 80% winrate.
- Against the Center Control algorithm, it did not stand out as much, with only a 20% winrate, our conclusion was that, as the center control algorithm aims to control the center of the map, it focuses on surrounding its own pieces, making them much harder to be captured back.
- So, going further into the center control algorithm, we can see that, against random moves, it excels with almost 100% winrate when going first, arround 97% when playing in small maps, and 90% when played in a big map.
- Against other algorithms, the trend is generally the same, except against the greedy algorithm, where it drawed in 500 games playing in the big map, when going second.

Experimental results:



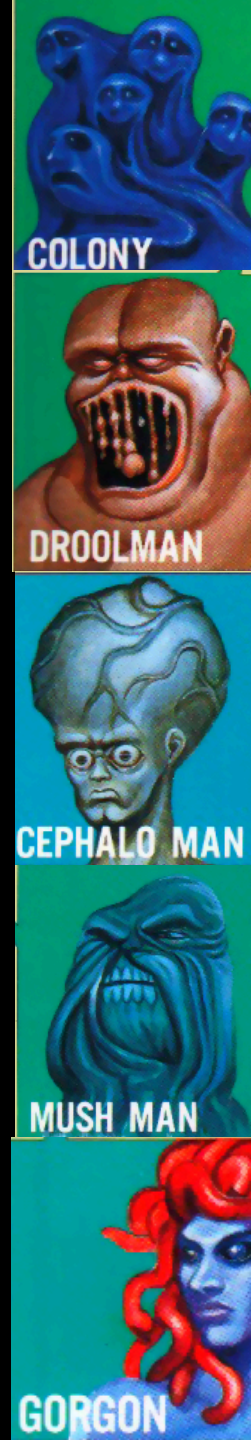
- In the greedy algorithm, we can pretty much see what we expected from an algorithm focused on capturing the most pieces as possible. Excelling in the smaller maps (grey and yellow) it oughts to finish the game as quick as it can by the lack of space and time for the opponent to multiply, yet having slightly less victories when faced against the same opponent in the bigger maps. When faced against the minmax (where it searches for every possibility in a certain depth), it is completely crushed as the minmax opposes the agressive behaviour.



- The random algorithm, given its nature, loses against every other algorithm by a firm margin, winning in average 1 in every 1000 games against the Greedy, 30 in 1000 against the Center Control, and, against itself, it has a 50% chance of winning, 51% when it plays first and 49% when playing second.
- An interesting curiosity is the relatively high win chance against the Minmax. As the Minmax expects the optimal plays from the opponent, the random nature of the algorithm has led it to ocasionally win against the Minmax.

Conclusions:

- After the development of this project we conclude that we achieved all of our objectives: we developed a simple Artificial Intelligence application for playing the two player game of Attax using different boards with different algorithms (Greedy, Centre Control, Random,MinMax) mainly focusing on the MinMax algortithm.
- After studying the results, we came to the conclusion that the MinMax algorithm is, in a general way, remarkably better compared to the other one.



Project Files:

This QR code will open our Github where you can access all of the documents regarding this project.

References:

- Python Tutorials on W3 Schools, https://www.w3schools.com/python/;
- Minmax Guide, https://www.freecodecamp.org/news/minimax-algorithm-guide-how-to-create-an-unbeatable-ai/;
- PyGame Library Documentation, https://www.pygame.org/docs/;
- Random Library Documentation, https://docs.python.org/3/library/random.html
- Copy Library Documentation, https://www.geeksforgeeks.org/copy-python-deep-copy-shallow-copy/;
- Sleep Library Documentation, https://realpython.com/python-sleep/ ;
- Files available on classroom Moodle.