Asssignment #1

IA - TURMA 4

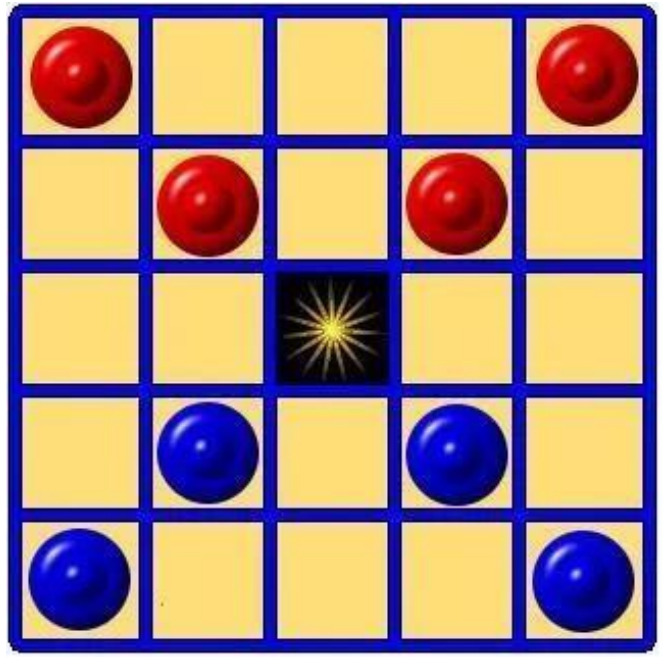
JOÃO FRANCISCO FERREIRA MALDONADO – UP202004244

JOSÉ LEANDRO RODRIGUES DA SILVA – UP202008061

TOMÁS PEREIRA MATOS GOMES - UP202004393

Work Specification

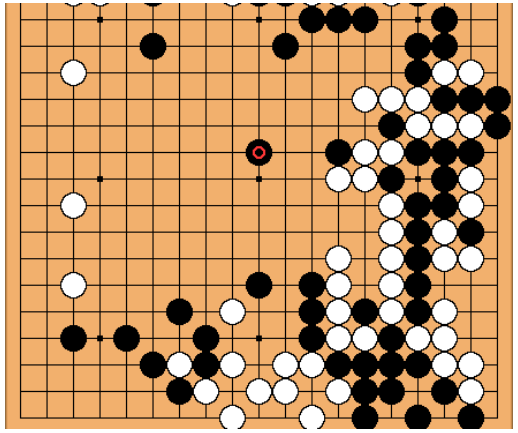
The aim of this project is to implement the game "Black-Hole Escape" and solve it using different versions of adversarial search methods. The game is a two-player game played in a 5x5 board with each player having four spaceships of their own that are placed on the board initially. The pieces can move horizontally or vertically and must move until they are stopped by other piece or the borders of the board. The players' objective is to land two of their spaceships on the center square (black hole).



Our goal is to develop adversarial search methods that can explore the possible game states and find the optimal sequence of moves while also predicting the opponent's moves. Then, we will compare the performance of different levels of difficulty using various evaluation functions and different depth levels of Minimax, different successor generation ordering, and also using Monte Carlo Tree Search algorithm with different configurations. The analysis of the results will be based on various quality parameters such as wins, draws, losses, and the number of plays required to obtain a win/loss, as well as the average time taken to obtain the plays.

Related work

Some research has been conducted on similar games such as the Go game, and the Connect Four game. These games have been used as a benchmark to evaluate the performance of various game-playing algorithms such as the Minimax algorithm, the Monte Carlo Tree Search algorithm, and their variants. In addition, some studies have compared the performance of these algorithms using different evaluation functions, depth levels, and search strategies.



Formulation of the problem as a search problem

Initial State: Each player pieces are in the original positions, it is red players' turn

State: The board will be represented as a two dimensional array, where the '1' correspond to the red pieces, the '2' to the blue pieces and the '0' to the empty cells/black-hole. The state will also include whose turn it is by an integer, being '1' if it is the red player turn or '2' if it is the blue player turn, the board dimensions and the black hole position

Objective test: Check if one of the player has only two pieces left.

OPERATORS	**Name**	**Preconditions**	**Effects**	**Cost**
	Move piece	-The piece to be moved must be owned by the player whose turn it is. -The destination square must be empty. -The move must be vertical or horizontal.	-The piece is moved from its current position to the new position. -If the destination square is the black hole, the moved piece is removed from the board.	Fixed cost of 1, as all moves are considered equally good or bad in this game.

Evaluation functions

The evaluation function is used by the search algorithm to evaluate the quality of a given state. Here are some heuristics that were used to design the evaluation function:

- **Piece mobility:** The player pieces that have more potential moves are more valuable than those that are more constrained.
- **Manhattan Distance to black hole:** The player pieces that are closer to the black hole are more valuable than those that are farther away.
- **Number of pieces in the black hole:** The player pieces that are on the black hole are more valuable than those that are not.
- **Central pieces:** This heuristic prioritizes pieces that are located in the central positions of the game board. It assigns a higher value to pieces that are in the central positions and have friendly pieces on the opposite side, and a negative value to those with unfriendly pieces.

Note that we also gathered some of the heuristics above in only one heuristic function, assigning different weights to each particular heuristic.

Implementation work

Basic structure:

- Programming Language: Python (pygame - https://www.pygame.org/docs/)
- Data Structures:
  - State Class:
    - Defined by a list of the pieces, the selected piece and the **player turn**
    - Draw method, used to draw every piece
    - **available_moves** method to get a list of available moves for all pieces of the current player turn
    - **is_valid_move** method to check if the selected move is on the available moves of the selected piece
    - **is_piece_at** method to check if a cell is empty
    - **move_piece** method to move a piece and mark it as removed if it reached the black hole
    - **check_win** method that returns true if one of the players has removed two pieces
    - **get_piece_at** method that returns a piece at the given coordinates or None if it is empty
    - **get_outcome** method that returns 1, 0 or –1 (win, tie or loss) for terminal nodes (game has ended), depending on the player turn

Implementation work

- Data Structures:
  - Piece Class:
    - Defined by its color, current position, if it is selected, if it has been removed, and the player who controls it
    - Draw method, used to draw the piece on the screen
    - **landed_on_black_hole** method, used to determine if the piece can be removed
    - **available_moves** method that returns a list with all available methods of the piece
  - Game Class:
    - Defined by winner, two players (human or AI) and state instantiation
    - Run method that contains the game loop
    - Draw, draw_grid and draw_board methods, use to draw the game
    - Events method used to listen to the user input and react accordingly

Approach

In order to create more diverse and complex evaluation functions we combined, with weights, the evaluation functions mentioned before. These functions take into account:

1. the Manhattan distance to the center, the number of available plays and the number of plays made
2. the Manhattan distance to the center, the number of pieces aligned with the black hole, the number of available plays and the number of plays made
3. the number of pieces aligned with the black hole (weight 4), number of pieces removed (weight 100) and number of plays made
4. number of pieces removed (weight 100), number of plays made (weight 10) and the number of plays available (weight 4)

The only operator used was the move operator that moved a given piece to a given location.

Minimax with alpha-beta pruning

- Generate the complete tree to the end states or to a certain depth (limited depth first search)
- Apply the utility function to these states
- Apply alpha-beta pruning, cutting subtrees that won't affect the final result
- Backpropagate the utility values
- Choose the movement with the highest value

Time Complexity: O(b^m) or O($b^{\frac{m}{2}}$) if the nodes are perfectly arranged

Spacial Complexity: O($b \times m$)

Monte Carlo Search

Class Node:

- Parent node
- N , representing times this node was visited
- Q, representing reward (wins-losses) from this node
- Children nodes
- Outcome, representing if it was a win, a loss or a tie

def value method that returns the node value according to the expression:

$$\frac{Q}{N} + c \times \sqrt{2 \times \frac{\log(parent.N)}{N}}$$

Where *c* is normally $\sqrt{2}$

Class UctMctsAgent:

- Root_state
- Current node

•Rollouts, run_time, node_count for statistics

best_move method that returns the best move after the algorithm has finished

select_node method that represents the selection phase

expand method that represents the expansion phase

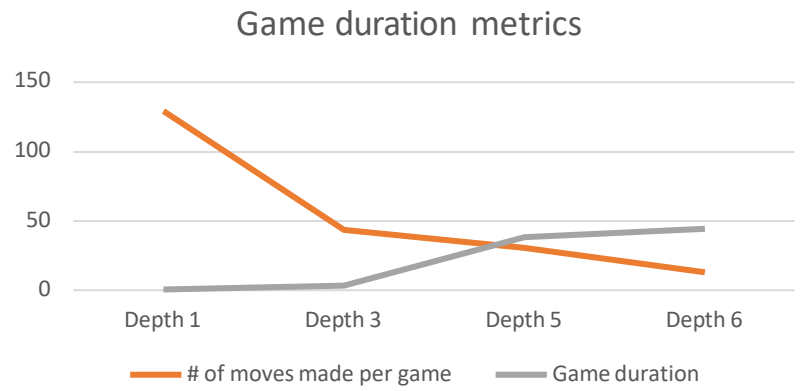
roll_out method that represents the simulation phase

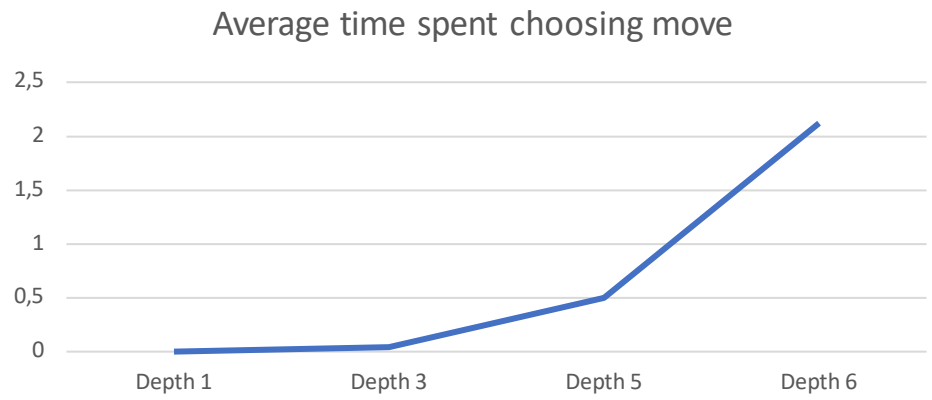
backup method that represents the backpropagation phase

search method that agregates all phases

Experimental Results

After well defining both AI approaches we ran several games to compare their performances. In order to better understand what to expect from each approach we started by simulating it against itself:





Next we compared the diferent heuristics performance:

AI	H1 D5	H2 D5	H3 D5	H4 D5
H1 D5	-	6-14	11-9	6-14
H2 D5		-	14-6	4-16
H3 D5	20-0	0-20	-	3-17
H4 D5			2-18	-

AI	H1 D3	H2 D3	H3 D3	H4 D3
H1 D3	-	1-19	12-8	6-14
H2 D3		-	17-3	8-12
H3 D3			-	6-14
H4 D3				-

AI	H1 D1	H2 D1	H3 D1	H4 D1
H1 D1	-	4-16	1-19	8-12
H2 D1		-	4-16	5-15
H3 D1			-	19-1
H4 D1			2-18	-

Experimental Results

Then we chose some of the better performing heuristics and tested them versus higher depths:

AI vs AI	Result
H2 D3 vs H1 D5	0-20
H2 D3 vs H3 D1	20-0
H2 D3 vs H3 D2	14-6
H3 D5 vs H2 D3	5-15
H2 D3 vs H3 D5	3-17

The monte carlo algorithm was also tested (results vs Minimax H4 D3):

Research Time (s)	Avg n # of rollouts	Result
0.5	3	0-20
1	5	1-19
3	9	1-19
5	12	2-18

Conclusions

With the development of this game we were able to contact with and implement algorithms that allow a computer to make a decision about a move, and planning next ones, in a game, according to the context.

It is more difficult to analyze the performance of higher depth minimax since it can see plays that humans don't. For instance, the AI sometimes chooses not to place a piece in the black hole, if it only depends on it to place it, and move other piece if it creates a better board state for the player and removes the piece later.

During the analysis of the data presented before we concluded the following:

- **The evaluation function used in minimax has a huge impact in its decision making ability**
- **The depth reached in a minimax algorithm influences the quality of the decision but also increases exponentially the time it takes to make a decision. We consider that the user should not wait more than one second for the computer to move so we chose the depth 5 as the highest difficulty.**
- **In this scenario the Monte Carlo Tree Search is not very effective since there is a huge state size and this algorithm needs a considerable amount of rollouts, which can only be achievable if we search the best play for an amount of time that makes it unplayable.**

References

https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/

https://towardsdatascience.com/monte-carlo-tree-search-an-introduction-503d8c04e168

https://stackabuse.com/minimax-and-alpha-beta-pruning-in-python/

https://www.scirp.org/journal/paperinformation.aspx?paperid=90972

https://rstudio-pubs-static.s3.amazonaws.com/594608_66ae5424cf0b4b8aacd307fc05d1c9b3.html