Focus 2D Game

Gonçalo Costa, up202103336

João Correia, up202005015

Ricardo Vieira, up202005091

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

WHAT IS FOCUS? WHAT ARE THE RULES?

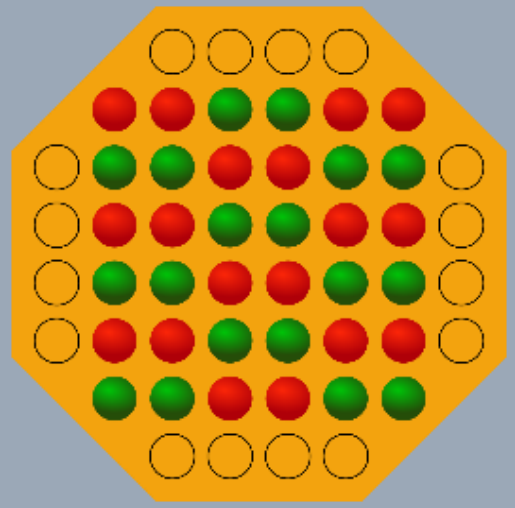
2 player game on a 6×6 board with 1×4 extensions on each side. Stacks may move as many spaces as there are pieces in the stack. Players can move a stack orthogonally, and if their piece is on top of the stack. The game ends when a player doesn’t have any piece on top of any stack.

Each stack, of 5 pieces maximum, when landing on another stack, merges with it. If the new stack contains more than five pieces, then pieces are removed from the bottom to bring it down to five. If a player's own piece is removed, they are kept outside the board to use later. If the piece is of the opponent, they are removed.

A player doesn’t need to move the complete stack. But if he doesn’t, he must only take as many pieces of the same as positions moved.

SEARCH PROBLEM FORMULATION

Initial State:



Operators:

Moving a stack (up, down, left or right)

Preconditions:

Can move as many positions as pieces on the stack and only moves if the top piece is theirs

Heuristics function:

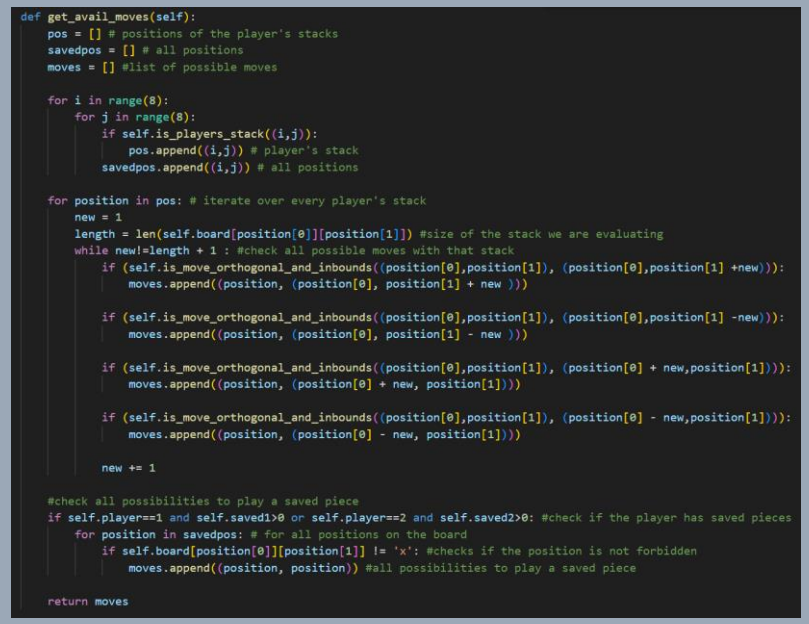
The number of stacks owned by the player and its size. If a player owns a stack his points will be the size of the stack times 2.
For each removed piece of the opponent, the player gets more points. To the points that a player already had from the first function we add 3 times the number of pieces removed from the opponent.
If a player gets the same points having one stack or two stacks it chooses the option with the most stacks.

Evaluation Function:

```python
def evaluate_board(self, board):
    player_score = 0
    opponent_score = 0
    counter = 0 # counter to help make the AI choose to have two st

    for row in board:
        for position in row:
            if position:
                if position[-1] == self.player: #when the stack is
                    counter+=1
                    player_score += 2 * len(position) #number of po
                    if self.player == 1:
                        player_score += self.removed2 * 3
                    else:
                        player_score += self.removed1 * 3
                elif position[-1] == 3-self.player: # if the stack
                    opponent_score += 2 * len(position)
                    if self.player == 1:
                        opponent_score += self.removed1 * 3
                    else:
                        opponent_score += self.removed2 * 3

    final_score = player_score - opponent_score + counter
    return final_score
```

Calculates the final score based on the stacks owned, size of that stacks and pieces removed from the opponent.

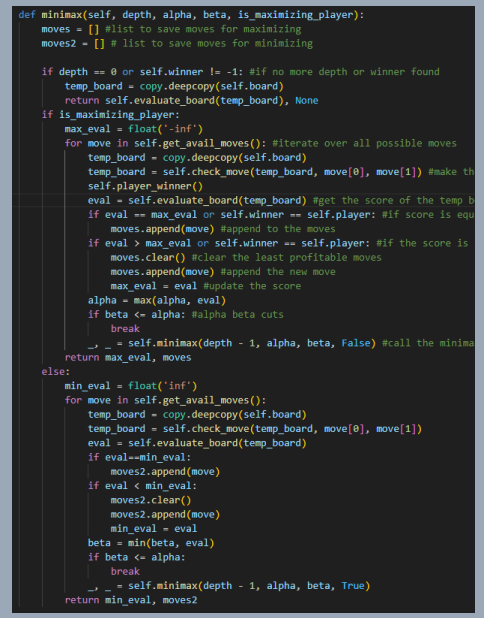
Operators:



```python
def get_avail_moves(self):
    pos = [] # positions of the player's stacks
    savedpos = [] # all positions
    moves = [] #list of possible moves

    for i in range(8):
        for j in range(8):
            if self.is_players_stack((i,j)):
                pos.append((i,j)) # player's stack
            savedpos.append((i,j)) # all positions

    for position in pos: # iterate over every player's stack
        new = 1
        length = len(self.board[position[0]][position[1]]) #size of the stack we are evaluating
        while new!=length + 1 : #check all possible moves with that stack
            if (self.is_move_orthogonal_and_inbounds((position[0],position[1]), (position[0],position[1] +new))):
                moves.append((position, (position[0], position[1] + new )))

            if (self.is_move_orthogonal_and_inbounds((position[0],position[1]), (position[0],position[1] -new))):
                moves.append((position, (position[0], position[1] - new )))

            if (self.is_move_orthogonal_and_inbounds((position[0],position[1]), (position[0] + new,position[1]))):
                moves.append((position, (position[0] + new, position[1])))

            if (self.is_move_orthogonal_and_inbounds((position[0],position[1]), (position[0] - new,position[1]))):
                moves.append((position, (position[0] - new, position[1])))

            new += 1

    #check all possibilities to play a saved piece
    if self.player==1 and self.saved1>0 or self.player==2 and self.saved2>0: #check if the player has saved pieces
        for position in savedpos: # for all positions on the board
            if self.board[position[0]][position[1]] != 'x': #checks if the position is not forbidden
                moves.append((position, position)) #all possibilities to play a saved piece

    return moves
```

Function that gets all possible moves of a player.

Implement Algorithm:

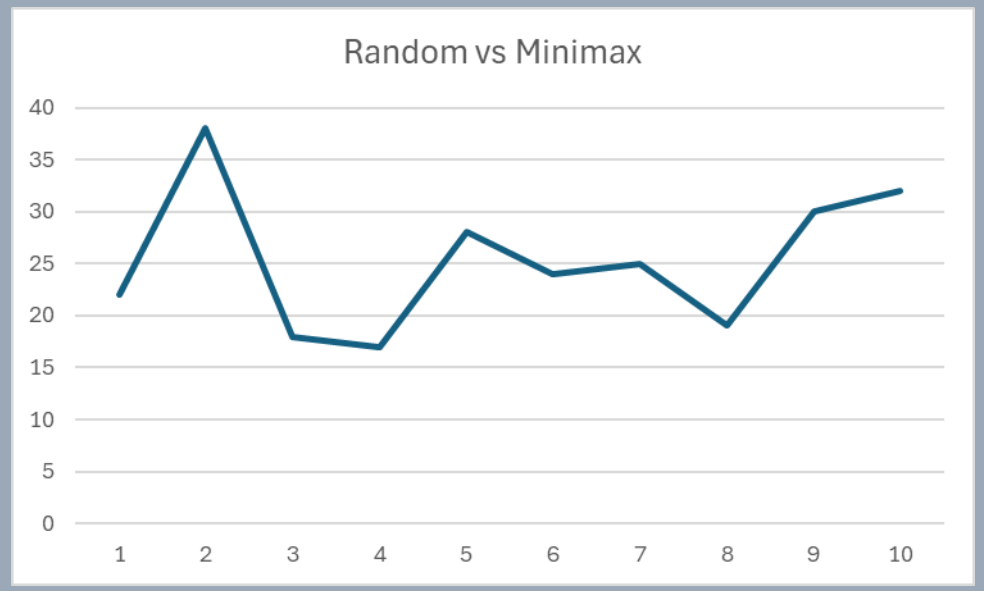


```python
def minimax(self, depth, alpha, beta, is_maximizing_player):
    moves = [] #list to save moves for maximizing
    moves2 = [] # list to save moves for minimizing

    if depth == 0 or self.winner != -1: #if no more depth or winner found
        temp_board = copy.deepcopy(self.board)
        return self.evaluate_board(temp_board), None
    if is_maximizing_player:
        max_eval = float('-inf')
        for move in self.get_avail_moves(): #iterate over all possible moves
            temp_board = copy.deepcopy(self.board)
            temp_board = self.check_move(temp_board, move[0], move[1]) #make th
            self.player_winner()
            eval = self.evaluate_board(temp_board) #get the score of the temp b
            if eval == max_eval or self.winner == self.player: #if score is equ
                moves.append(move) #append to the moves
            if eval > max_eval or self.winner == self.player: #if the score is
                moves.clear() #clear the least profitable moves
                moves.append(move) #append the new move
                max_eval = eval #update the score
            alpha = max(alpha, eval)
            if beta <= alpha: #alpha beta cuts
                break
            _, _ = self.minimax(depth - 1, alpha, beta, False) #call the minima
        return max_eval, moves
    else:
        min_eval = float('inf')
        for move in self.get_avail_moves():
            temp_board = copy.deepcopy(self.board)
            temp_board = self.check_move(temp_board, move[0], move[1])
            eval = self.evaluate_board(temp_board)
            if eval==min_eval:
                moves2.append(move)
            if eval < min_eval:
                moves2.clear()
                moves2.append(move)
                min_eval = eval
            beta = min(beta, eval)
            if beta <= alpha:
                break
            _, _ = self.minimax(depth - 1, alpha, beta, True)
        return min_eval, moves2
```

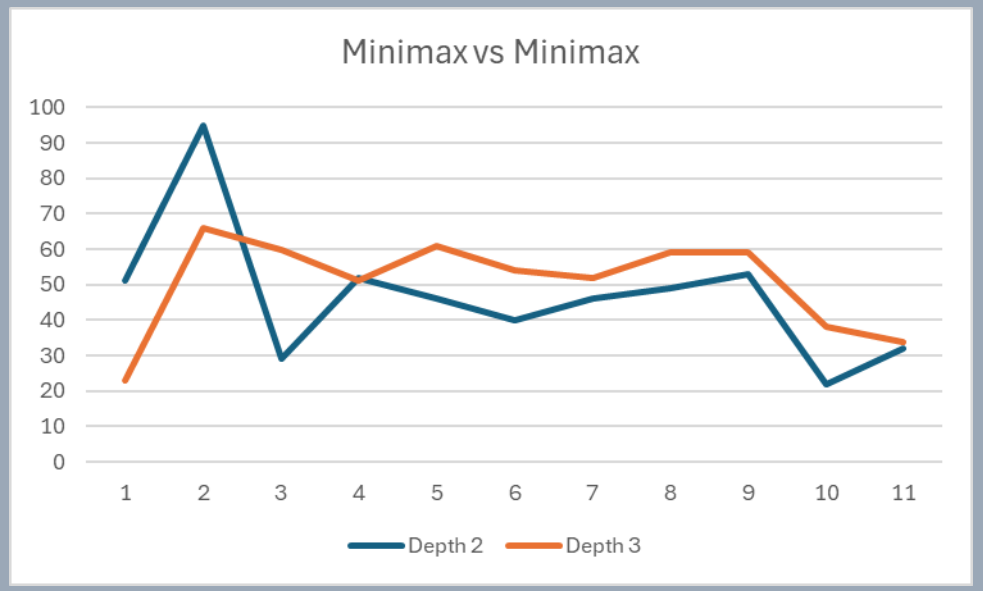
Minimax function applying Alpha-Beta cuts to calculate the best possible move of a player.

APPROACH

For the main minimax we used depth 8 in both approaches



Number of plays minimax took to finish the game. Average time to play: 0,02 seconds



Number of plays minimax took to finish the game. Against depth 2 won 7 times taking 0,0221 seconds to play, against depth 3 won 6 times, taking an average time of: 0,0155

RESULTS AND ANALYSIS

CONCLUSIONS

With this project, we were able to understand in a practical way how the minimax algorithm works. It was the first time for all the group elements creating an AI to play a game and we appreciated the challenge.

We had to think in a critical way in order to get the best evaluation of the board so that we could get to the best possible move.

The creation of the minimax was the hardest part, but we managed to write a functional algorithm that is able to win easily against humans and random and winning most times against another minimax with lower depths.

Another issue after we made the minimax more functional was the time it took to think. Using depths larger than two, it would take a long time. This led us to improve our algorithm using alpha-beta cuts, which made all moves almost instant.

Every element had an equal contribution to the development of the project.

Programming Language

Data structures

Data analysis

Development Environment

Framework

References

https://boardgamegeek.com/boardgame/789/focus

https://boardgamegeek.com/video/477999/focus/domination-focus-sid-sackson-classic-sdj-1981

https://en.wikipedia.org/wiki/Focus_(board_game)

REFERENCES AND MATERIALS USED