AI Project
(AI for Mancala Game)

Dalia Ayman Ameen	1600521
Doaa Yehia Sayed Ahmed	1600525
Dina Adel Ismail	1600540
Zeinab Ismail Hefny	1600613
Sara Raafat Mohamed	1600623

1- A brief description of the game and the implementation including any bonus features:

A brief description of the game:

The game consists of a board, this board is split 2 halves for each player, each half consists of 6 holes and one bucket. Initially each hole has 4 stones.

The player should choose from the non-empty holes to play, if the number of stone in the chosen hole is “x”, then the “x” stones are distributed to the next “x” holes.

If the last stone reaches the bucket, then the player play again.

If any of the half’s board is empty, then the game is over and the other half board stones be collected to its player.

The game has 2 modes: stealing & non stealing, in stealing mode if the last stone reaches an empty hole (in the player half) so the player takes all the stones (which exists in the hole in front of the empty hole) from the other player.

Implementation:

First, we implement the structure of the game and this part is the helper functions, then the AI takes part (this is found in the minimax and score functions).

The detailed functions in the next section.

Bonus:

1- **Save & load function**: we support a “**quit**” option where we store the last state of the board, then when loading the game again the player is asked whether he want to start the new game or load his last state.

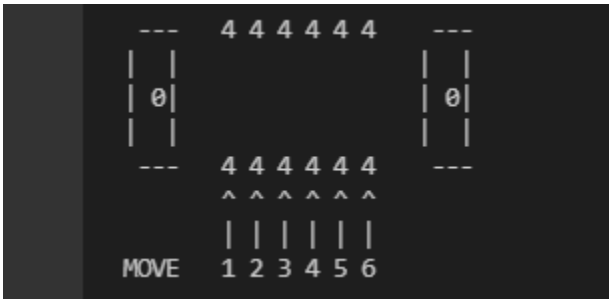
2- **Support various difficulty levels corresponding to different game tree depths.**

2-A detailed description of the utility function(s) used by your algorithm:

Helper Functions:

def initialize_board (): function to initialize the board, we implement our board as a 2D numpy array, each row is the half board for each player. The array is 2*8, the 8 is (6 holes for each player, bucket, and non-used element just to make the board symmetric)

def print_board(board): function to print the board after each play.



def valid_move(board, move, player): function to decide whether the move (play) is valid or not, it takes the board (2D numpy array) & the move (which should be a number from 1 to 6) and the player (which is: 1 for (AI) or 2 for ther player) as arguments, then It makes sure that the player doesn't choose to play (in the non-used elements or in the buckets or an empty hole).

def is_game_over(board): function to check whether the game is over or not, it takes the board (2D numpy array) as an argument and checks if the sum of the stones in all the holes of any half of the board is empty, it the sum is zero, then the game is over.

def collect_reminder_stones(board): function which first check if the game is over or not, and then collect the reminder stones in the holes and put it them in the corresponding player bucket.

def decide_winner(board): function decides the winner by comparing the buckets for each player (compare between board[0][0] & board[1][7]).

def player1_move(board, no_of_iterations, index): This function is a helper function to the play move function.

It takes the board , no of moves to play in the same move and the index of the next hole as a parameter.it then lists the possible_moves of the player and iterates through these moves and distributes stones in these holes.

It then returns the last location it stands on.

def player2_move(board, no_of_iterations, index): Same as player 1 but for player 2

def play_move(board, hole, player): It calls the player's move and then return the last location and next player to play.

def get_valid_moves(board, maximizingPlayer): It loops over the side of the board of the player and checks for the correct moves that he could play and returns a list of these valid moves

def stealing_mode(board, hole, player,last_location): This function implements the mode of stealing: It checks if the last position's size is 1 and the opponent's size is not zero then it steals all the stones and puts it in the pocket.

def eval_board(board, mode, player): This Function is applied to evaluate the score of the board to allow minimax algorithm to choose the best suitable score, It is the scoring Function Of the algorithm

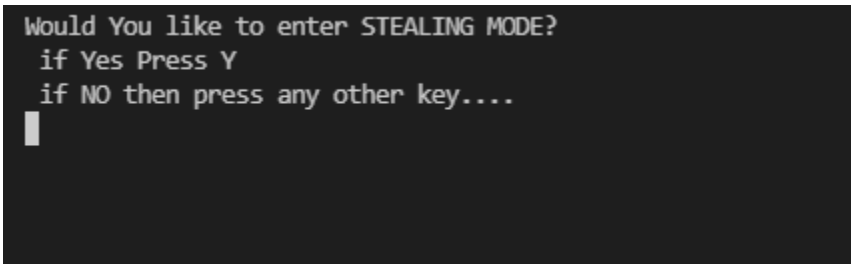
def minimax(board, depth = 10000, alpha = -999, beta = +999, maximizingPlayer = 0, mode = False):

This function implements minimax alpha-beta algorithm where we adopt the pruning method to decrease the number of traversed nodes in a tree during depth first search method to choose the best suitable moves to try to win the game!

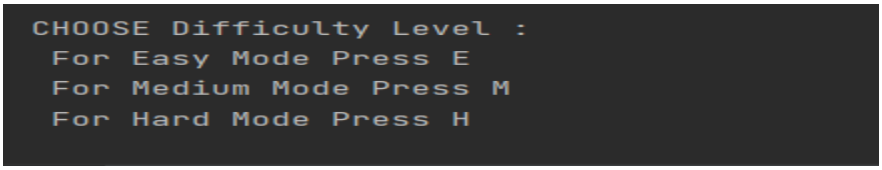
3-A user guide with snapshots:

1- A welcome message by our team then, the player is asked whether he want to start the game or not. If he pressed (Y or y) he will start, if any key else is pressed then the computer will start.

2- Then the player is asked to chooses the mode of the game (stealing or no stealing). If he pressed (Y or y) the stealing mode is activated, if any key else is pressed then the no stealing mode is activated.



3- The player chooses the difficult level:

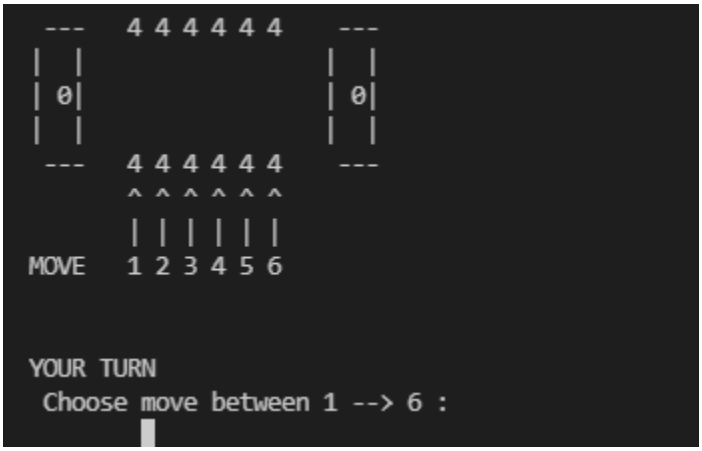


4- The game starts. 

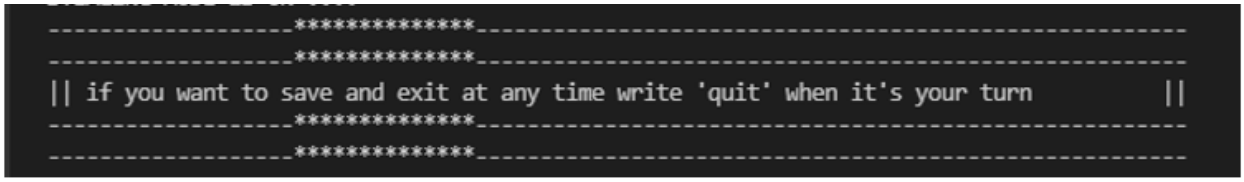
In the player turn he has to choose a number between 1 – 6.

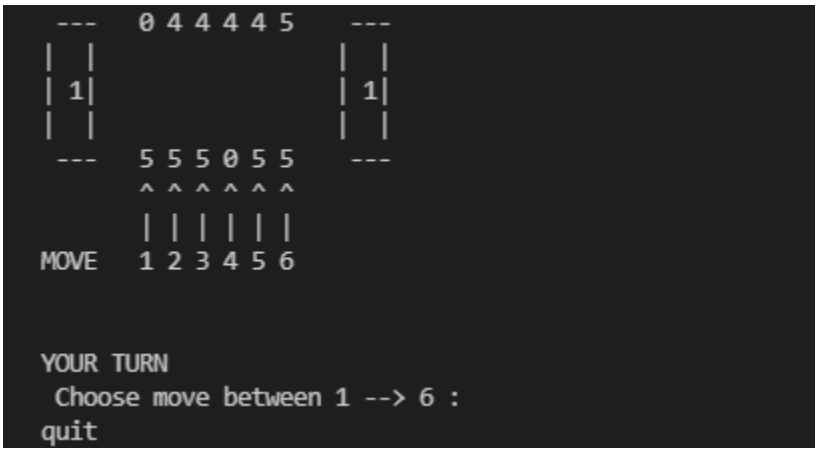
After he finishes his turn, the computer (AI) makes his move.

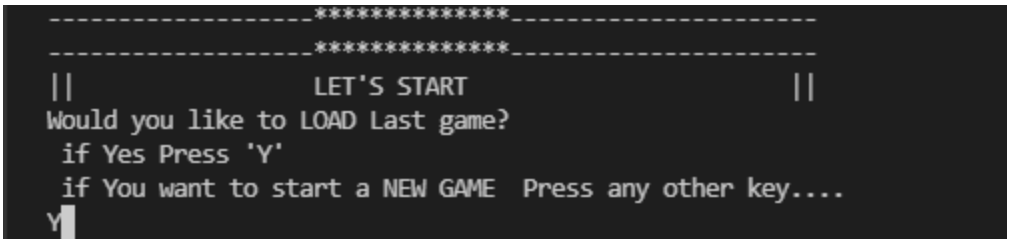
Until one of them finishes his stones.

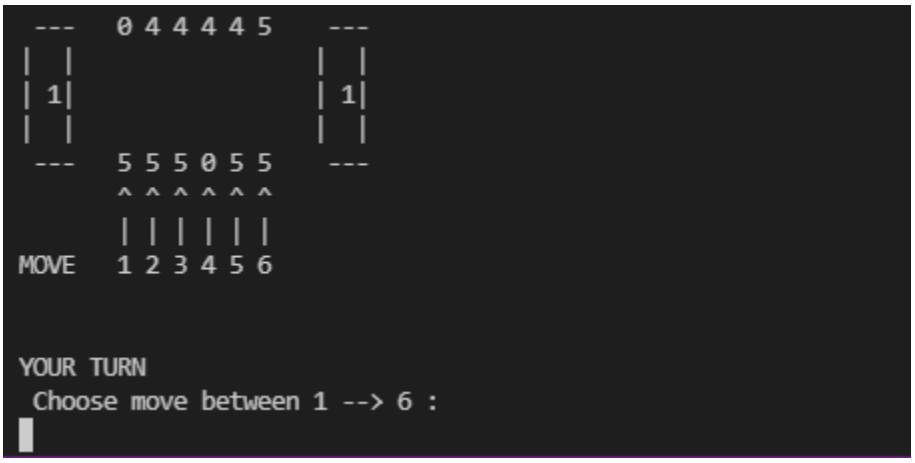


5- And there is an option for quit the game, save the last state of the board and load it again. When he starts the game again he asked whether he wants to reload the game or start a new one.









6-A summary of how the work was split among the team members:

- **Dalia**: Implemented the **helper functions** (`initialize_board` (), Is_game_over(), decide_winner( ))
- **Doaa**: Implemented **save and reload function, the helper functions** (`print_board`(),`collect_reminder_stones`(), `player1_move`(), `player2_move`(), `play_move`(), `get_valid_moves`()),
- **Dina**: Implemented the **minimax function.**
- **Zeinab**: Implemented the **evaluation functions**
- **Sara**: Implemented **stealing mode function**

7- Any additional useful documentation (including code documentation, descriptions of difficulties encountered, tricks used, etc.):

Additional useful documentation:

This video visualizes minimax function.

https://www.youtube.com/watch?v=l-hh51ncgDI&t=407s

Descriptions of difficulties encountered:

we face difficulties in giving a score to each move in the game at the first because we didn't know the moves that could lead max stones in the bucket.

The debugging part (we stay 10 hours just debugging :D).