Faculty of engineering ain shams university
computer and systems engineering dep.
4th Elec Year

CSE481
Artificial intelligence
Mancala Project Description

Submitted by:

- David john dawoud	section: 1	code: 1600531
- Girges micheal	section: 1	code: 1600446
- Ziad Tarek Hassan	section: 1	code: 1600606
- Ahmed Taha fekry	section: 2	code: 1600108
- Mina wagdy awny	section: 5	code: 1601544

Contents

Project Brief Description:

Full implementation of the famous Mancala Game

Including 4 Modes of the game:

1- Human player Vs. Human player

2- Human player Vs. Ai player

3- Ai player Vs. Human player

4- Ai player Vs. Ai player

- The Ai player uses MinMax Algorithm With Alpha-beta Pruning written from scratch.

- when you get started, you'll be asked about which mode you want to play.

- Our project supports "Stealing Mode "of Mancala.

Bonus supported:

1- various difficulties of AI player.

2- Saving and loading the current game in case of any sudden termination during the game.

3- Various checks on user's input before the game or within the game.

4- Alpha beta optimization.

5- added timer inside minmax algorithm.

6- added iterative deepening variant of minmax algorithm.

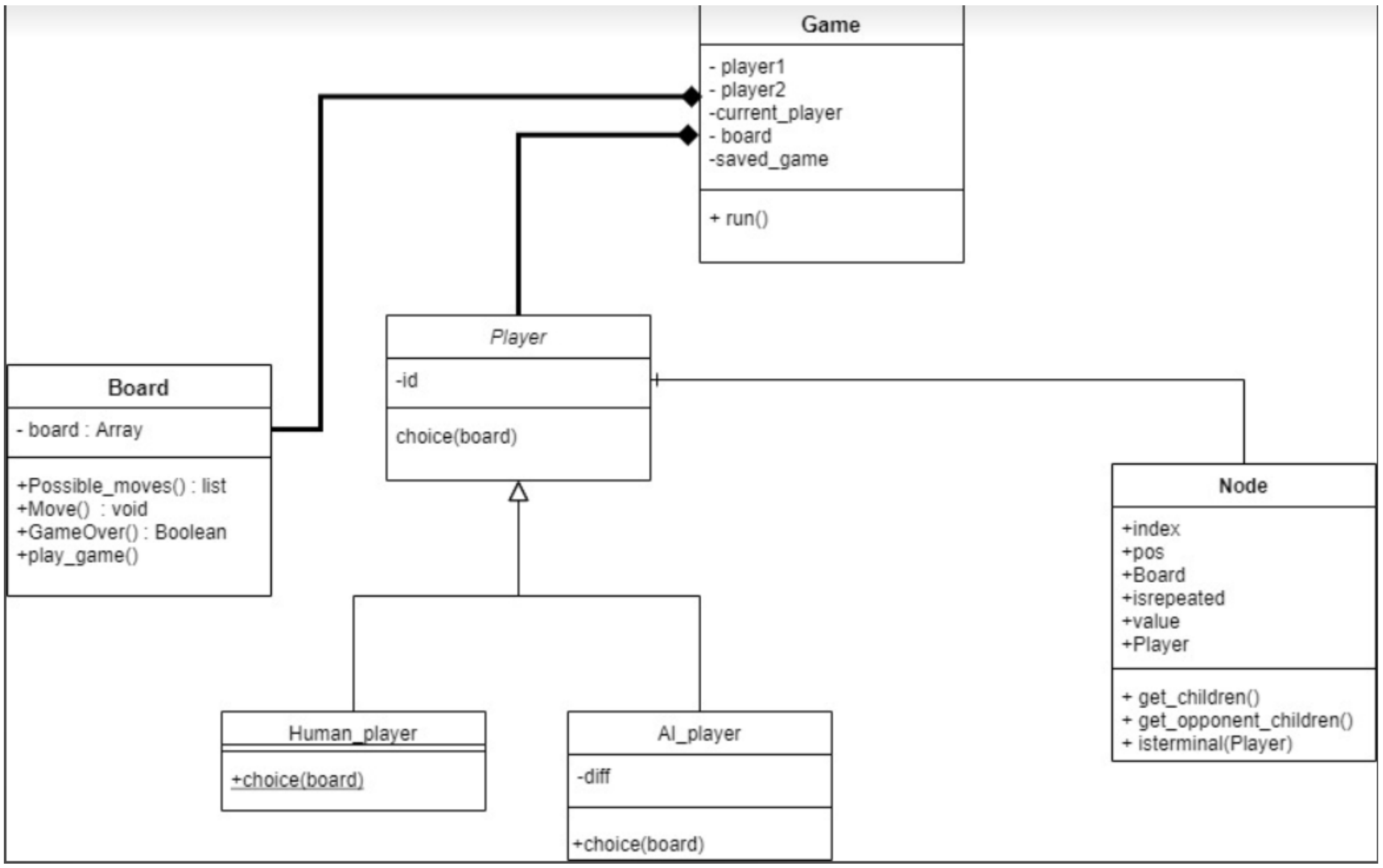
7- MinMax optimization using sorting of children.

Project detailed Explanation with bonus features:

Click here.

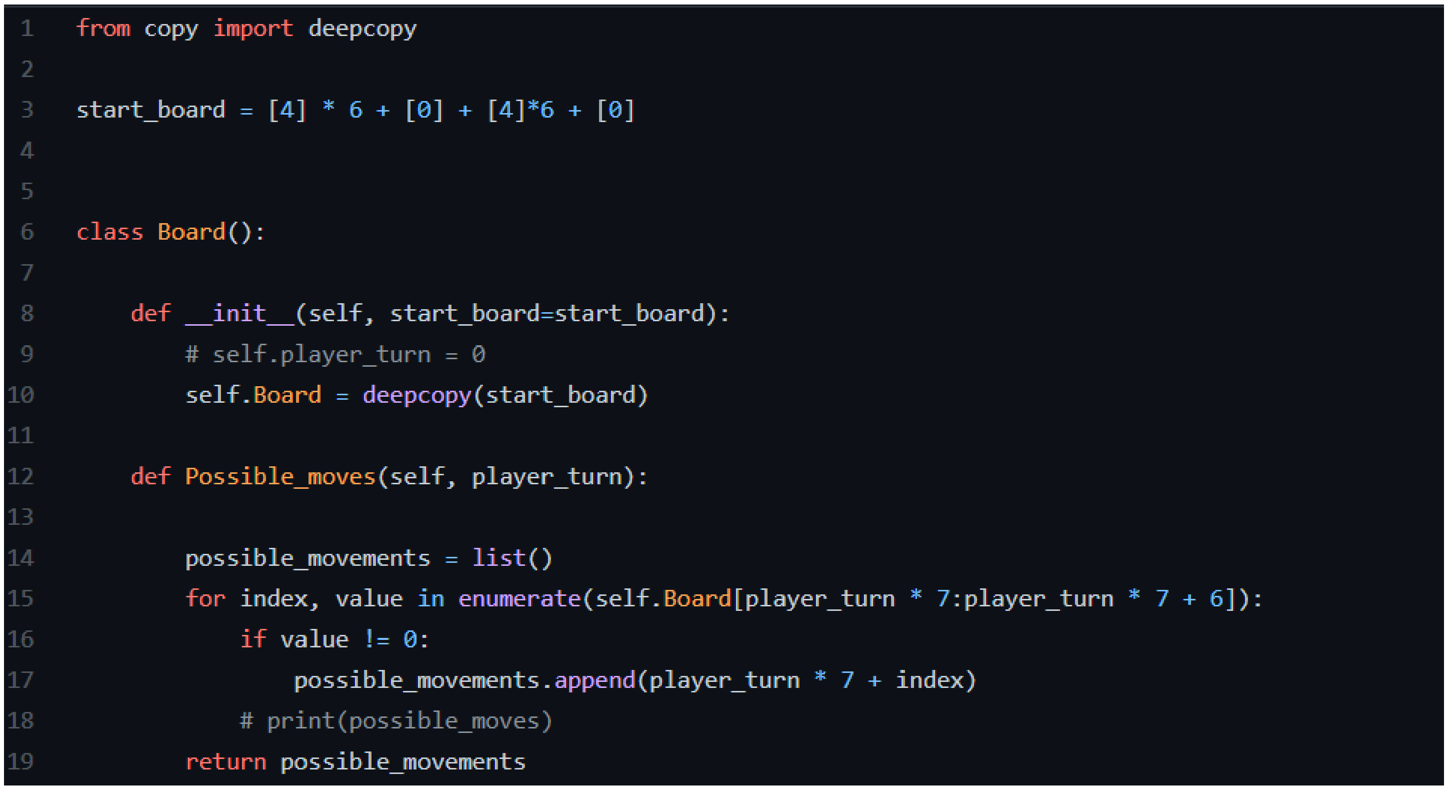
Project Repository: Click here.

Class Diagram of the complete design of our game:



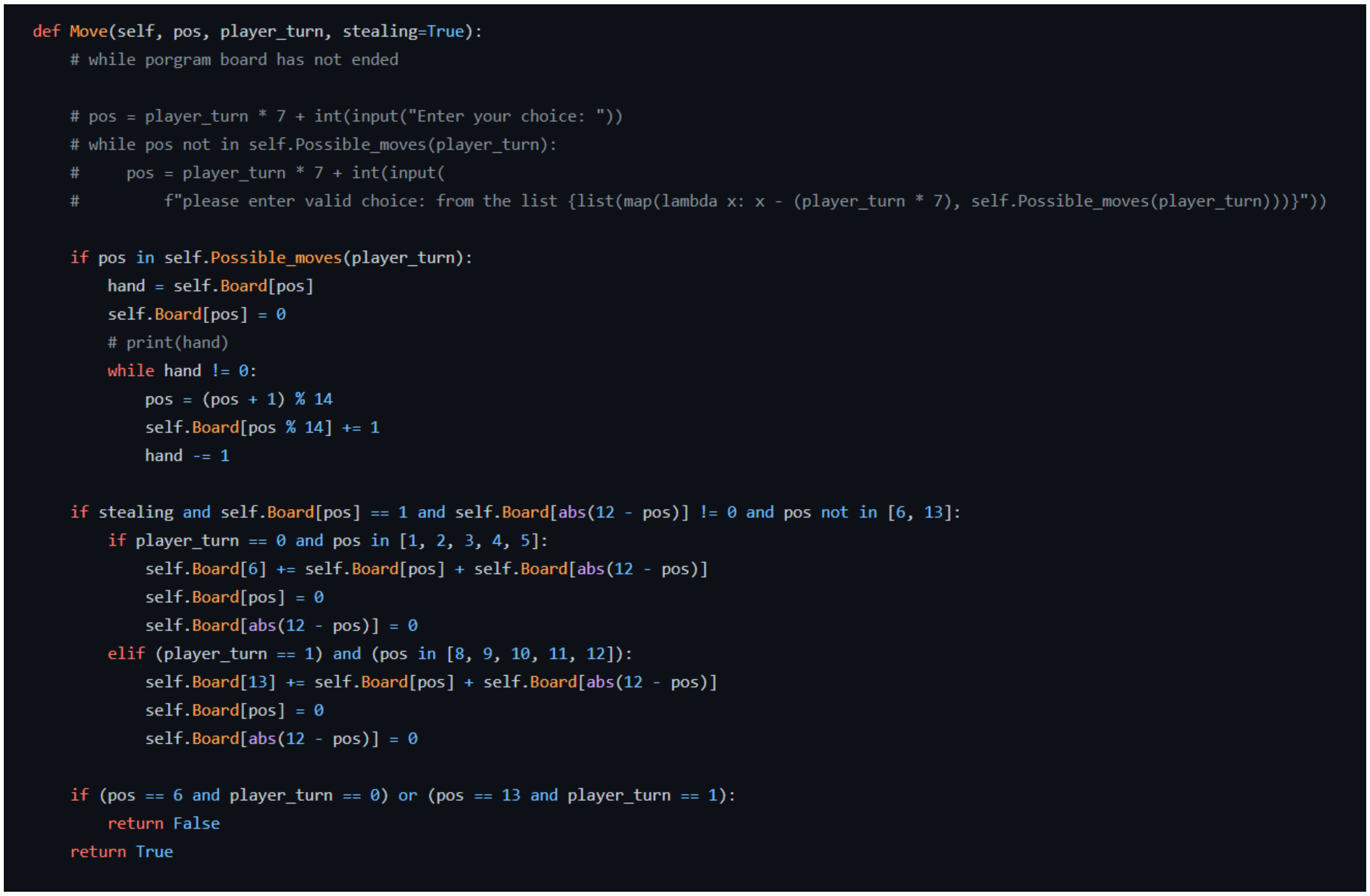
Detailed Description of Classes & Functions:

Board Class:



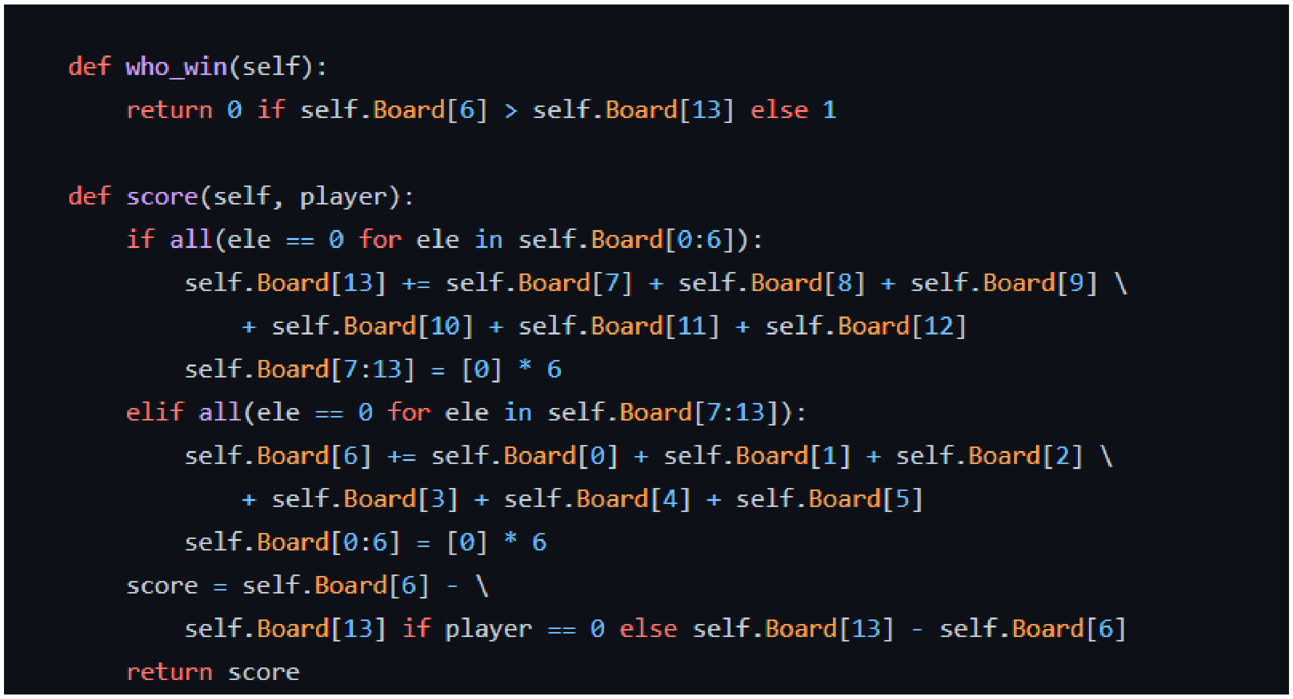
```python
from copy import deepcopy

start_board = [4] * 6 + [0] + [4]*6 + [0]


class Board():

    def __init__(self, start_board=start_board):
        # self.player_turn = 0
        self.Board = deepcopy(start_board)

    def Possible_moves(self, player_turn):

        possible_movements = list()
        for index, value in enumerate(self.Board[player_turn * 7:player_turn * 7 + 6]):
            if value != 0:
                possible_movements.append(player_turn * 7 + index)
            # print(possible_moves)
        return possible_movements
```

- Our Mancala Board is represented by a list of 14 elements
The first 7 elements are for player 1 (6 holes for stones + 1 for mancala itself) and the other 7 are for player 2.
- Our board is always printed on each play ... starting the game with value 4 in each hole and zeros for mancalas.
- Possible moves function: returns the possible moves for each player, as player 1 is allowed to play from his side of board
And also the entered played position mustn't equal 0.



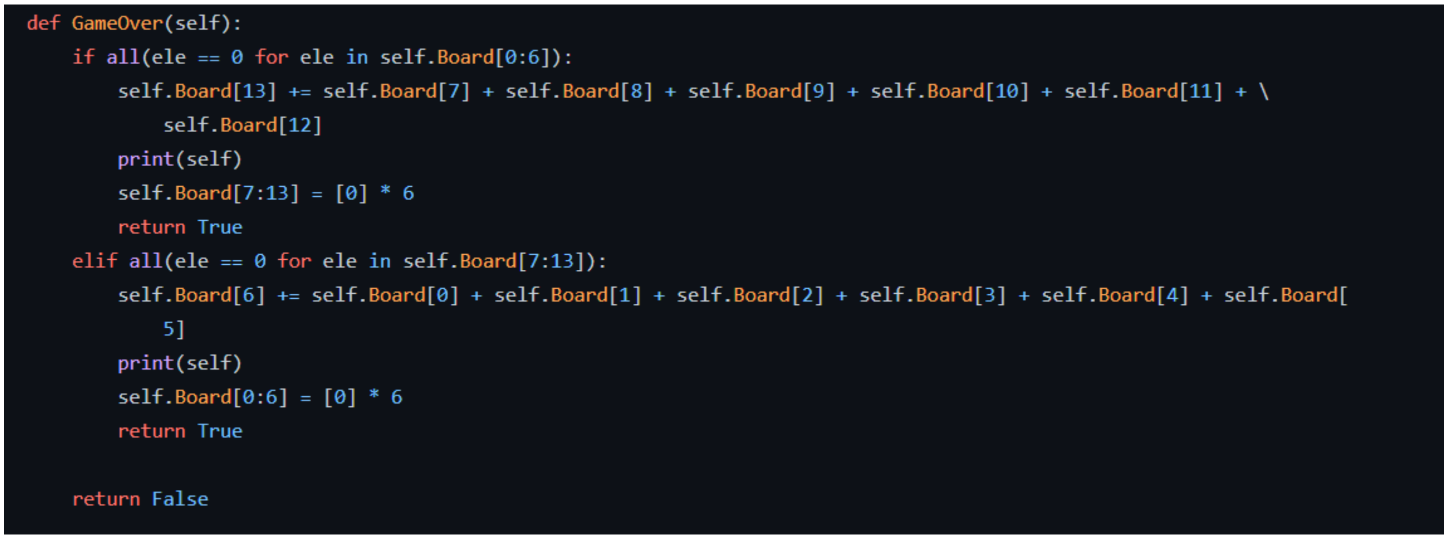
```python
def Move(self, pos, player_turn, stealing=True):
    # while porgram board has not ended

    # pos = player_turn * 7 + int(input("Enter your choice: "))
    # while pos not in self.Possible_moves(player_turn):
    #     pos = player_turn * 7 + int(input(
    #         f"please enter valid choice: from the list {list(map(lambda x: x - (player_turn * 7), self.Possible_moves(player_turn)))}"))

    if pos in self.Possible_moves(player_turn):
        hand = self.Board[pos]
        self.Board[pos] = 0
        # print(hand)
        while hand != 0:
            pos = (pos + 1) % 14
            self.Board[pos % 14] += 1
            hand -= 1

    if stealing and self.Board[pos] == 1 and self.Board[abs(12 - pos)] != 0 and pos not in [6, 13]:
        if player_turn == 0 and pos in [1, 2, 3, 4, 5]:
            self.Board[6] += self.Board[pos] + self.Board[abs(12 - pos)]
            self.Board[pos] = 0
            self.Board[abs(12 - pos)] = 0
        elif (player_turn == 1) and (pos in [8, 9, 10, 11, 12]):
            self.Board[13] += self.Board[pos] + self.Board[abs(12 - pos)]
            self.Board[pos] = 0
            self.Board[abs(12 - pos)] = 0

    if (pos == 6 and player_turn == 0) or (pos == 13 and player_turn == 1):
        return False
    return True
```

Move function is the core of playing the game, the function which is responsible for the basic movement of mancala.
At first, passing the entered position by the user, then checking
If it is a valid input, if so a value of 1 added to each neighbor until the value of the entered position on the board is 0.
Stealing mode:
If stealing mode is on, and at any time there is a hole on the board which has a value of 1 and the opposite hole on board is not empty, so add that 1 + the value of the opposite hole's value and add the result to the mancala of the player who has a value 1 in his side of the board and exactly vice versa for the other player if that case ever happened.



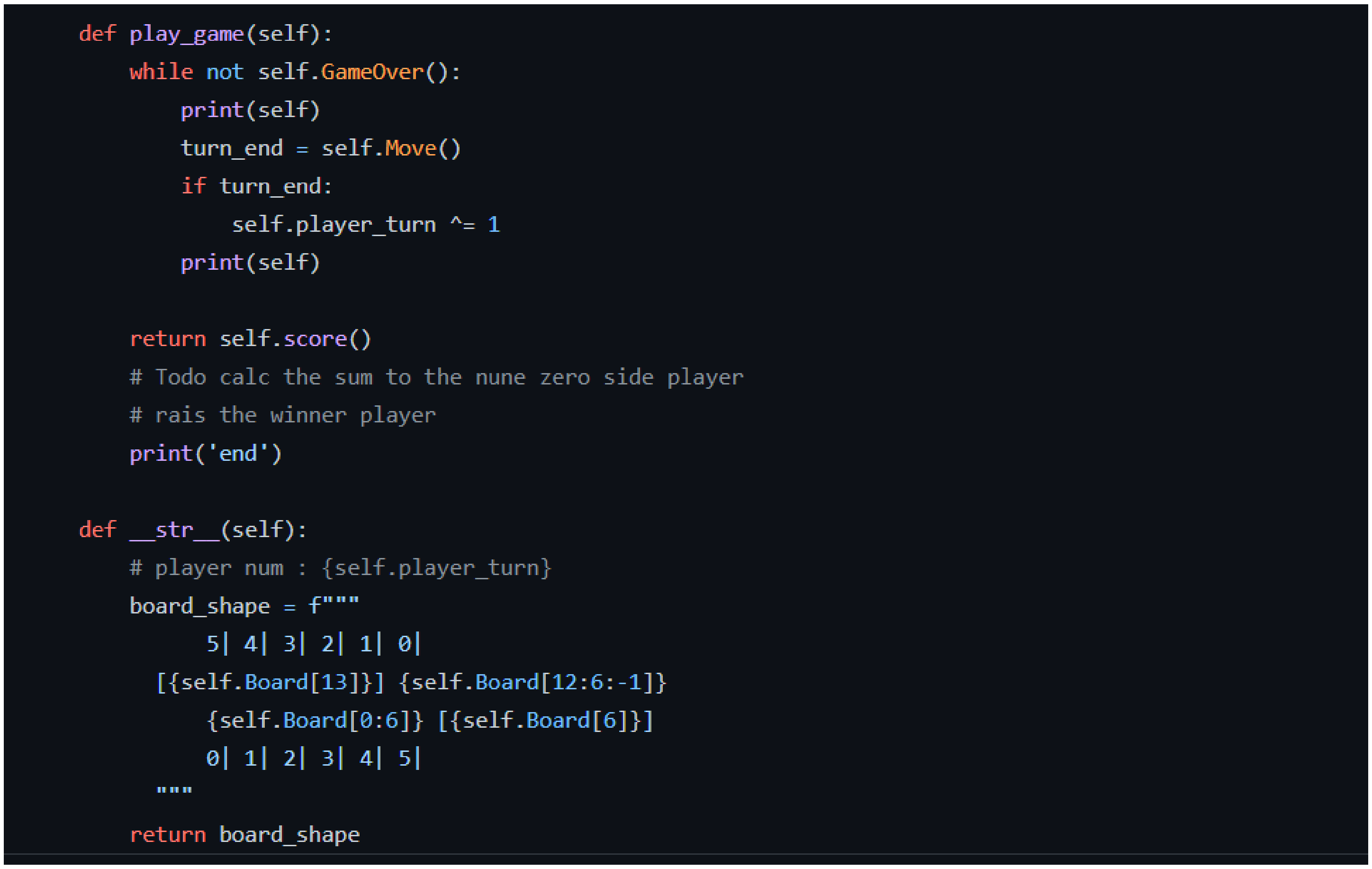
```python
def who_win(self):
    return 0 if self.Board[6] > self.Board[13] else 1

def score(self, player):
    if all(ele == 0 for ele in self.Board[0:6]):
        self.Board[13] += self.Board[7] + self.Board[8] + self.Board[9] \
            + self.Board[10] + self.Board[11] + self.Board[12]
        self.Board[7:13] = [0] * 6
    elif all(ele == 0 for ele in self.Board[7:13]):
        self.Board[6] += self.Board[0] + self.Board[1] + self.Board[2] \
            + self.Board[3] + self.Board[4] + self.Board[5]
        self.Board[0:6] = [0] * 6
    score = self.Board[6] - \
        self.Board[13] if player == 0 else self.Board[13] - self.Board[6]
    return score
```

Who_win: a simple check on which player has a higher score on his mancala, to be the winner of the game.
Score function: checks if all holes on any side of the board are empty (the end of the game) and if so, we will add all the holes' values of the other side of the board to this player's side mancala.
This function returns the difference score between the 2 mancalas to determine who wins.



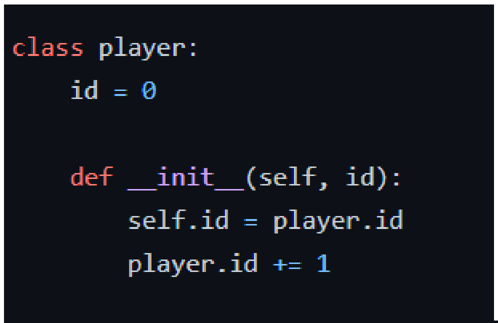
```python
def GameOver(self):
    if all(ele == 0 for ele in self.Board[0:6]):
        self.Board[13] += self.Board[7] + self.Board[8] + self.Board[9] + self.Board[10] + self.Board[11] + \
            self.Board[12]
        print(self)
        self.Board[7:13] = [0] * 6
        return True
    elif all(ele == 0 for ele in self.Board[7:13]):
        self.Board[6] += self.Board[0] + self.Board[1] + self.Board[2] + self.Board[3] + self.Board[4] + self.Board[
            5]
        print(self)
        self.Board[0:6] = [0] * 6
        return True

    return False
```

GameOver function: just check on the end of the game condition stated before and calculating the final score in each mancala.



```python
def play_game(self):
    while not self.GameOver():
        print(self)
        turn_end = self.Move()
        if turn_end:
            self.player_turn ^= 1
        print(self)

    return self.score()
    # Todo calc the sum to the nune zero side player
    # rais the winner player
    print('end')

def __str__(self):
    # player num : {self.player_turn}
    board_shape = f"""
          5| 4| 3| 2| 1| 0|
      [{self.Board[13]}] {self.Board[12:6:-1]}
          {self.Board[0:6]} [{self.Board[6]}]
          0| 1| 2| 3| 4| 5|
      """
    return board_shape
```

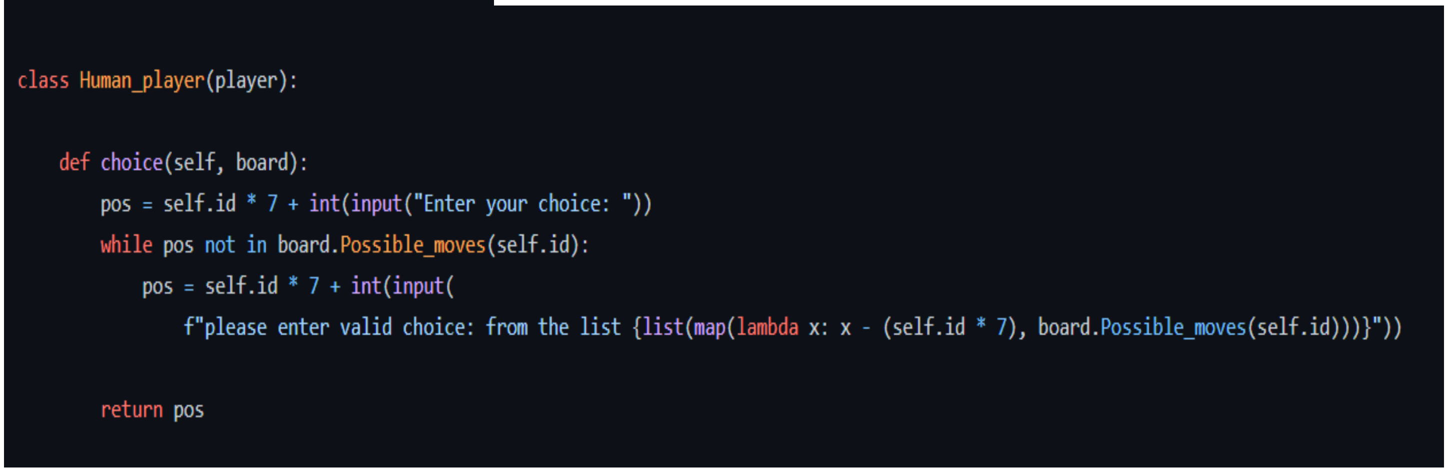
Play_game function: just applying the logic explained in all the past functions, Player move then XORING the player turn as the switch play (player 0 and 1) Then return the final score.



```python
class player:
    id = 0

    def __init__(self, id):
        self.id = player.id
        player.id += 1
```

Players Class:

When instantiating an object of player class, it automatically given an id to mark him.



```python
class Human_player(player):

    def choice(self, board):
        pos = self.id * 7 + int(input("Enter your choice: "))
        while pos not in board.Possible_moves(self.id):
            pos = self.id * 7 + int(input(
                f"please enter valid choice: from the list {list(map(lambda x: x - (self.id * 7), board.Possible_moves(self.id)))}"))

        return pos
```

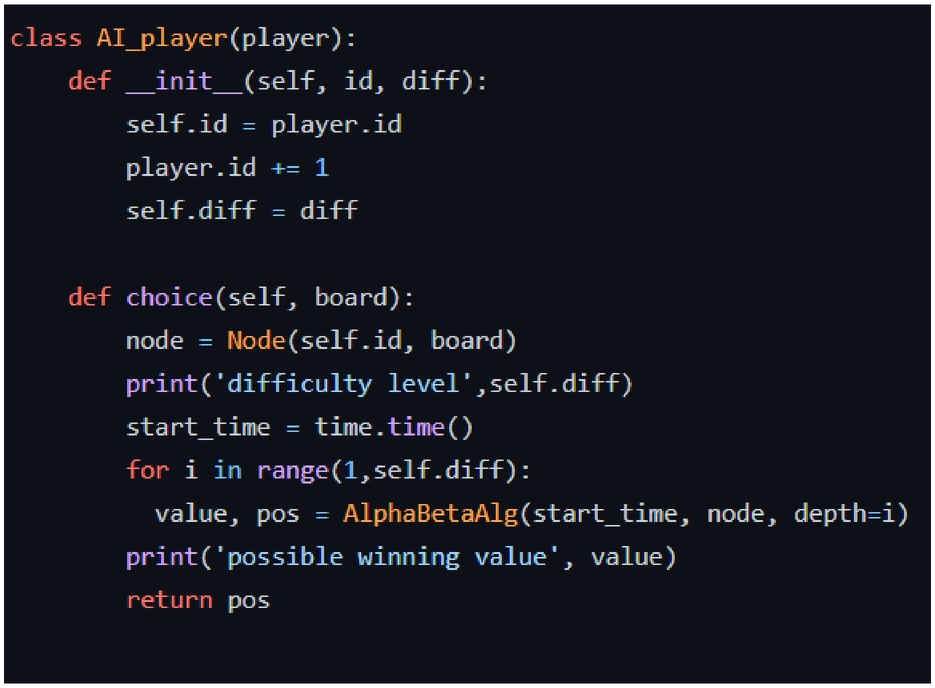
Human_player Class:

Inherit from Player class to be given an id.

Choice function:

Whatever the id of the player (0 or 1) it returns the position input by the player, and in case entering a not valid hole position.

This will fire a message to him to enter a valid one with printing the possible valid moves.



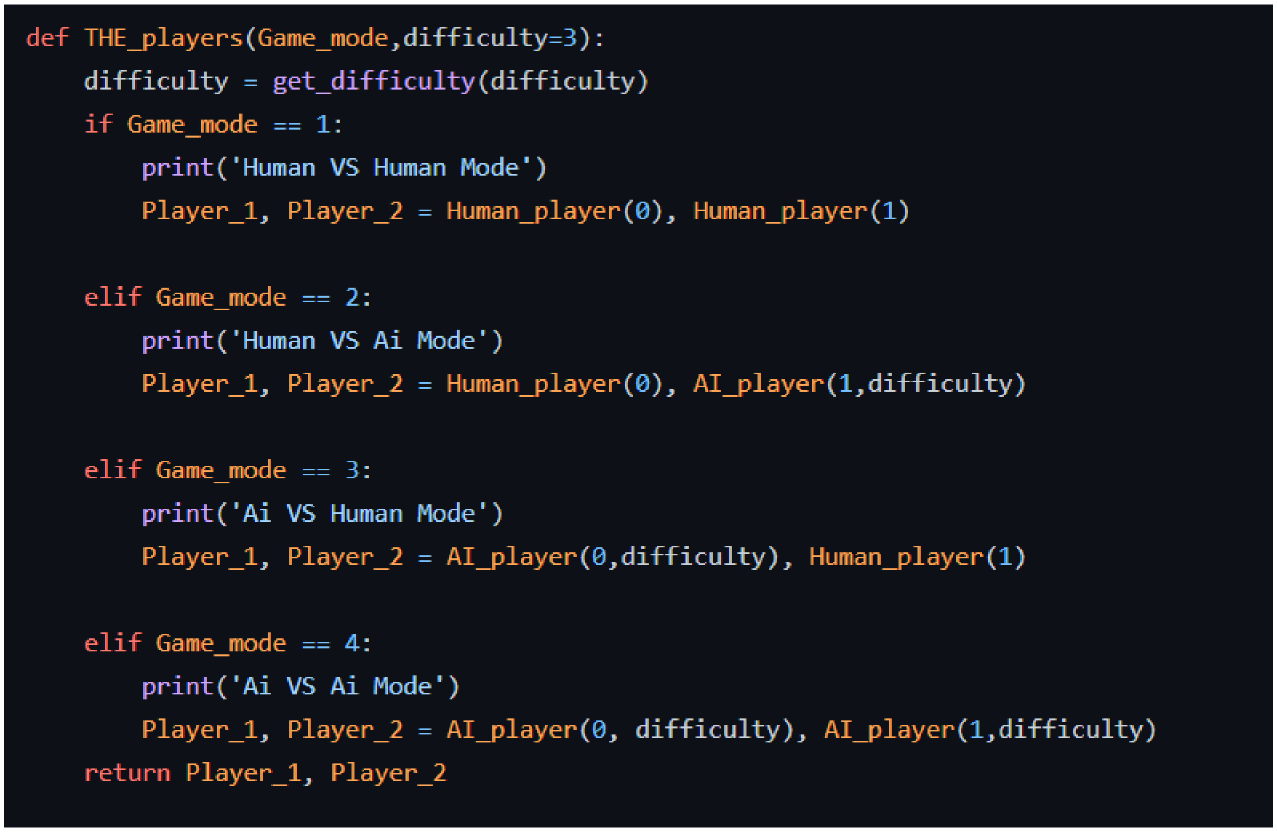
```python
class AI_player(player):
    def __init__(self, id, diff):
        self.id = player.id
        player.id += 1
        self.diff = diff

    def choice(self, board):
        node = Node(self.id, board)
        print('difficulty level',self.diff)
        start_time = time.time()
        for i in range(1,self.diff):
          value, pos = AlphaBetaAlg(start_time, node, depth=i)
        print('possible winning value', value)
        return pos
```

AI_player Class: **it also inherits from player class to be given an id.**

Choice function: when creating the AI agent, the user will be asked about the difficulty level (a Bonus Part) and based on it. The AI agent will get deeper or shallower in the searched tree which is made inside the AlphaBetaAlg function “will be described later”.

The function will print the possible winning value.

Return: position to be played by AI agent.



```python
def THE_players(Game_mode,difficulty=3):
    difficulty = get_difficulty(difficulty)
    if Game_mode == 1:
        print('Human VS Human Mode')
        Player_1, Player_2 = Human_player(0), Human_player(1)

    elif Game_mode == 2:
        print('Human VS Ai Mode')
        Player_1, Player_2 = Human_player(0), AI_player(1,difficulty)

    elif Game_mode == 3:
        print('Ai VS Human Mode')
        Player_1, Player_2 = AI_player(0,difficulty), Human_player(1)

    elif Game_mode == 4:
        print('Ai VS Ai Mode')
        Player_1, Player_2 = AI_player(0, difficulty), AI_player(1,difficulty)
    return Player_1, Player_2
```

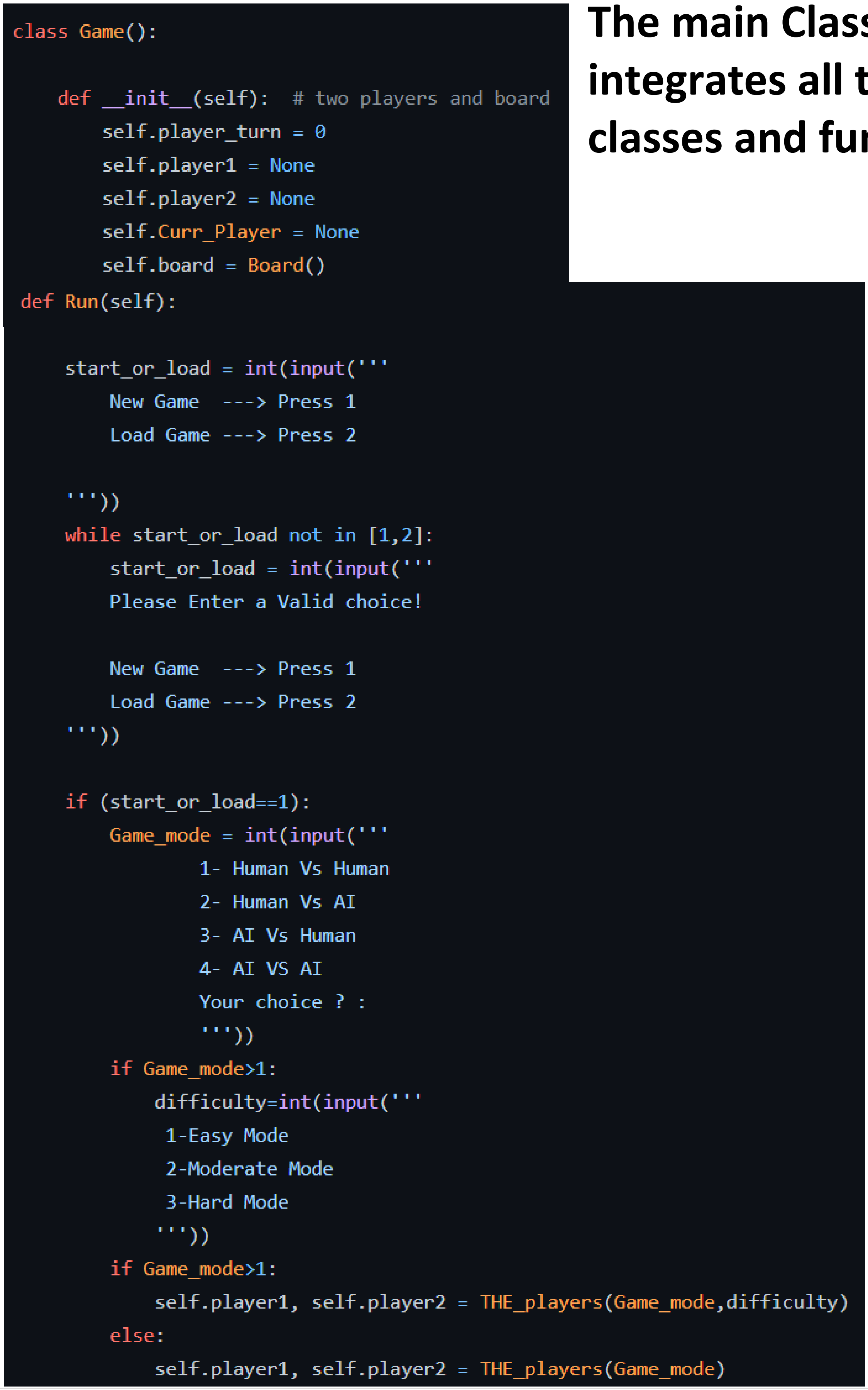
THE_players function:

Based on the “Game_mode” & “difficulty” input variables from user, Players are created with their id.

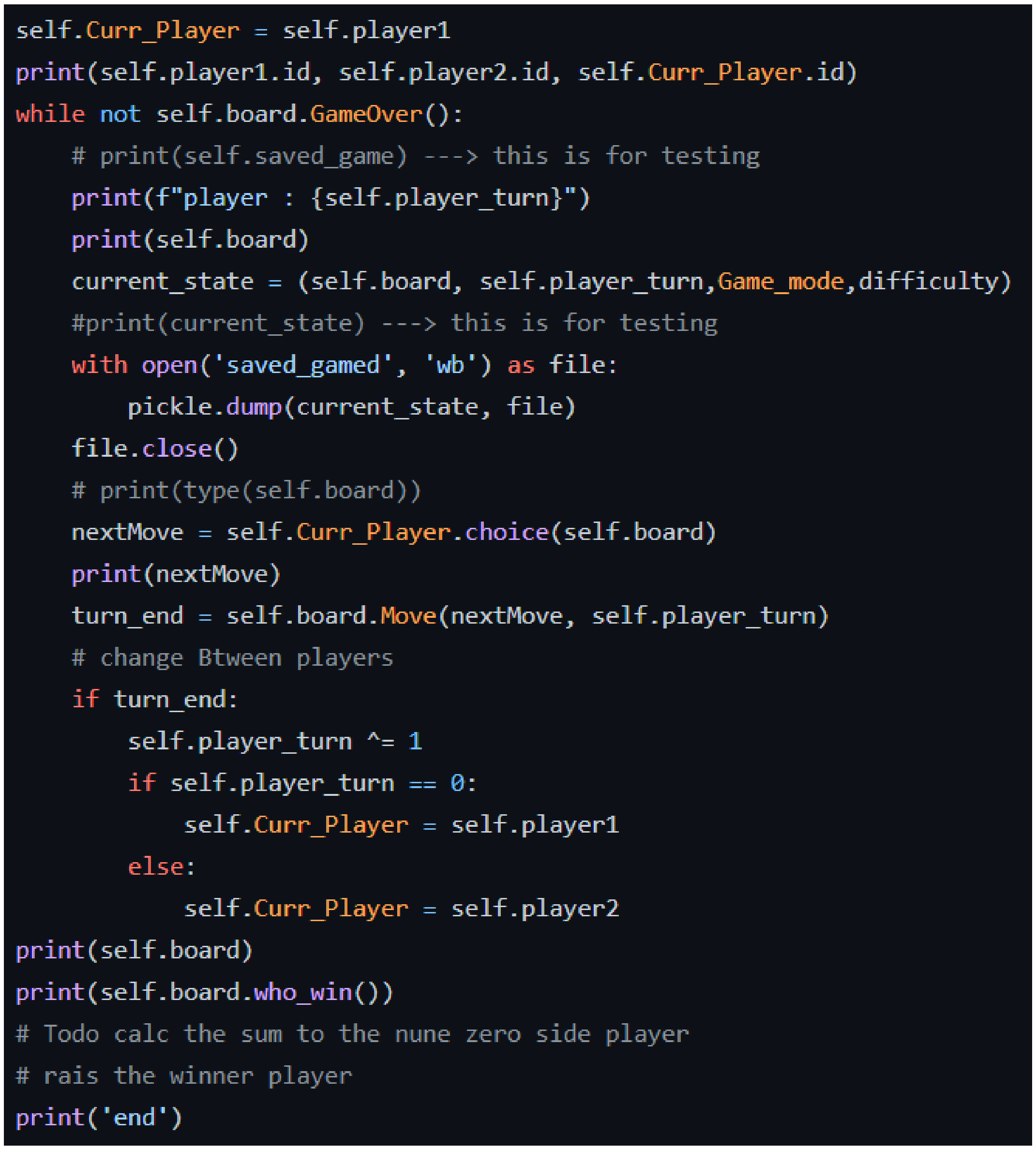
Also, for AI agent, the difficulty input variable will determine for which depth it will iterate throw the Tree.

Game Class:

The main Class which integrates all the past classes and functions of s



```python
class Game():

    def __init__(self):  # two players and board
        self.player_turn = 0
        self.player1 = None
        self.player2 = None
        self.Curr_Player = None
        self.board = Board()
 def Run(self):

     start_or_load = int(input('''
         New Game  ---> Press 1
         Load Game ---> Press 2

     '''))
     while start_or_load not in [1,2]:
         start_or_load = int(input('''
         Please Enter a Valid choice!

         New Game  ---> Press 1
         Load Game ---> Press 2
     '''))

     if (start_or_load==1):
         Game_mode = int(input('''
                 1- Human Vs Human
                 2- Human Vs AI
                 3- AI Vs Human
                 4- AI VS AI
                 Your choice ? :
                 '''))
         if Game_mode>1:
             difficulty=int(input('''
              1-Easy Mode
              2-Moderate Mode
              3-Hard Mode
             '''))
         if Game_mode>1:
             self.player1, self.player2 = THE_players(Game_mode,difficulty)
         else:
             self.player1, self.player2 = THE_players(Game_mode)
```



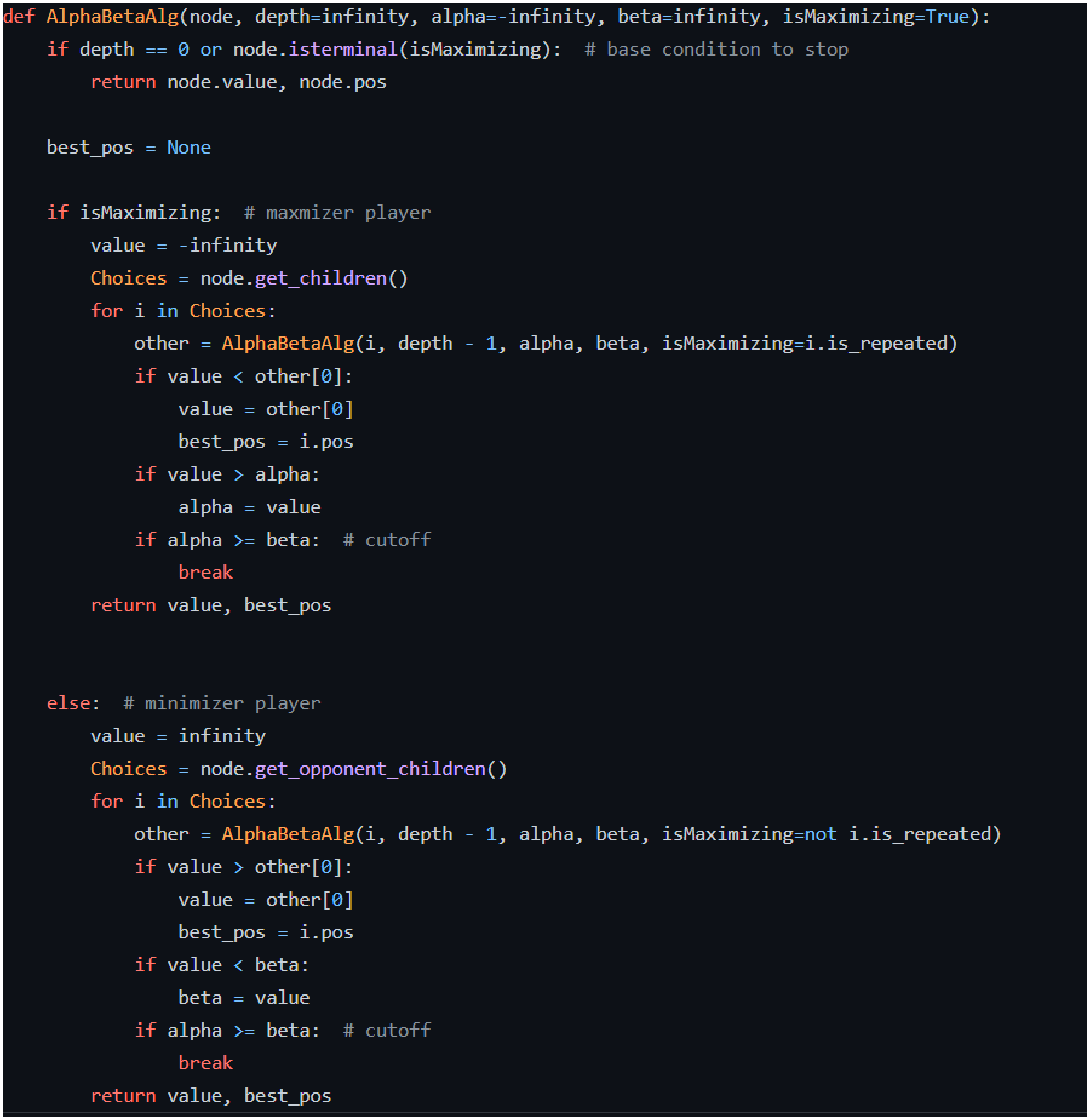
```
self.Curr_Player = self.player1
print(self.player1.id, self.player2.id, self.Curr_Player.id)
while not self.board.GameOver():
    # print(self.saved_game) ---> this is for testing
    print(f"player : {self.player_turn}")
    print(self.board)
    current_state = (self.board, self.player_turn,Game_mode,difficulty)
    #print(current_state) ---> this is for testing
    with open('saved_gamed', 'wb') as file:
        pickle.dump(current_state, file)
    file.close()
    # print(type(self.board))
    nextMove = self.Curr_Player.choice(self.board)
    print(nextMove)
    turn_end = self.board.Move(nextMove, self.player_turn)
    # change Btween players
    if turn_end:
        self.player_turn ^= 1
        if self.player_turn == 0:
            self.Curr_Player = self.player1
        else:
            self.Curr_Player = self.player2
print(self.board)
print(self.board.who_win())
# Todo calc the sum to the nune zero side player
# rais the winner player
print('end')
```

Run function:

-As shown in our code, the user will choose whether to start a new game or load the last un-finished game (A bonus Part).

-If load game is selected, the game will work automatically without any other questions.

- If new game is selected, the user will be asked about which mode of playing he wants to play.
- in case choosing any playing mode having AI model, the user will be asked about the difficulty level required.
- While not the game is over "periodically checks the game stopping condition stated before" the players will play alternatively using the utility functions explained in all the past modules.
- Using "Pickle" library for saving and loading Custom data types in python, the following data is saved on each game play:
(Board – Which player turn – Game mode – difficulty)
By loading this data, the game will be loaded Later on "in case of sudden termination for example".



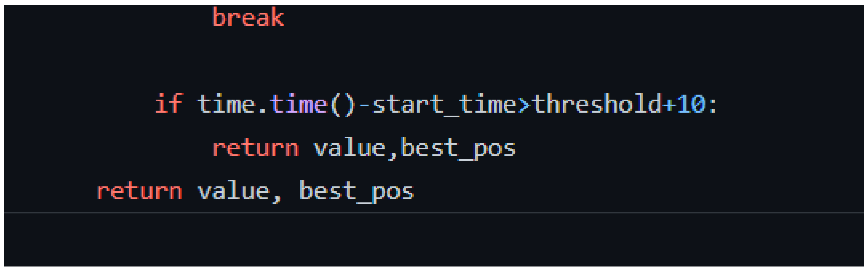
```
def AlphaBetaAlg(node, depth=infinity, alpha=-infinity, beta=infinity, isMaximizing=True):
    if depth == 0 or node.isterminal(isMaximizing):  # base condition to stop
        return node.value, node.pos

    best_pos = None

    if isMaximizing:  # maxmizer player
        value = -infinity
        Choices = node.get_children()
        for i in Choices:
            other = AlphaBetaAlg(i, depth - 1, alpha, beta, isMaximizing=i.is_repeated)
            if value < other[0]:
                value = other[0]
                best_pos = i.pos
            if value > alpha:
                alpha = value
            if alpha >= beta:  # cutoff
                break
        return value, best_pos


    else:  # minimizer player
        value = infinity
        Choices = node.get_opponent_children()
        for i in Choices:
            other = AlphaBetaAlg(i, depth - 1, alpha, beta, isMaximizing=not i.is_repeated)
            if value > other[0]:
                value = other[0]
                best_pos = i.pos
            if value < beta:
                beta = value
            if alpha >= beta:  # cutoff
                break
        return value, best_pos
```

The minmax algorithm is implemented as a function that takes 5 arguments, first one, the node that it stands on right now, second one is the depth of the node, third and fourth arguments are the alpha and beta values,

5th and last argument is a Boolean value to say if the player who is playing right now is a maximizer or a minimizer.
Inside the function there is a condition to check if the depth is zero (so we have reached Maximum depth) or if the node is terminal, so we are at the leaf nodes, then we return the node value.
Then we check to see if the player is a maximizer, if so , we define alpha with a value of negative infinity, and an empty list to put in it the best sorting of the children in case of a maximizer player, then we loop over children , checking to see the max value returned from those children using recursion, and put it in the alpha value if this value is greater than the old alpha, otherwise it remains the same, then there's a condition to check if the value of alpha is greater than or equal beta, then cut off occurs.
This function at the end returns the value, which is the alpha value since it is a maximizer, and also returns a list sorted in descending order to place the largest value on the left to get the best pruning which is a bonus feature.

Then we check to see if the player is a minimizer, if so , we define beta with a value of positive infinity, and an empty list to put in it the best sorting of the children in case of a minimizer player, the we loop over children , check to see the minimum value returned from those children using recursion, and put it in the beta value if this value is less than the old beta, otherwise it remains the same, then there's a condition to check if the value of alpha is greater than or equal beta, then cut off occurs.

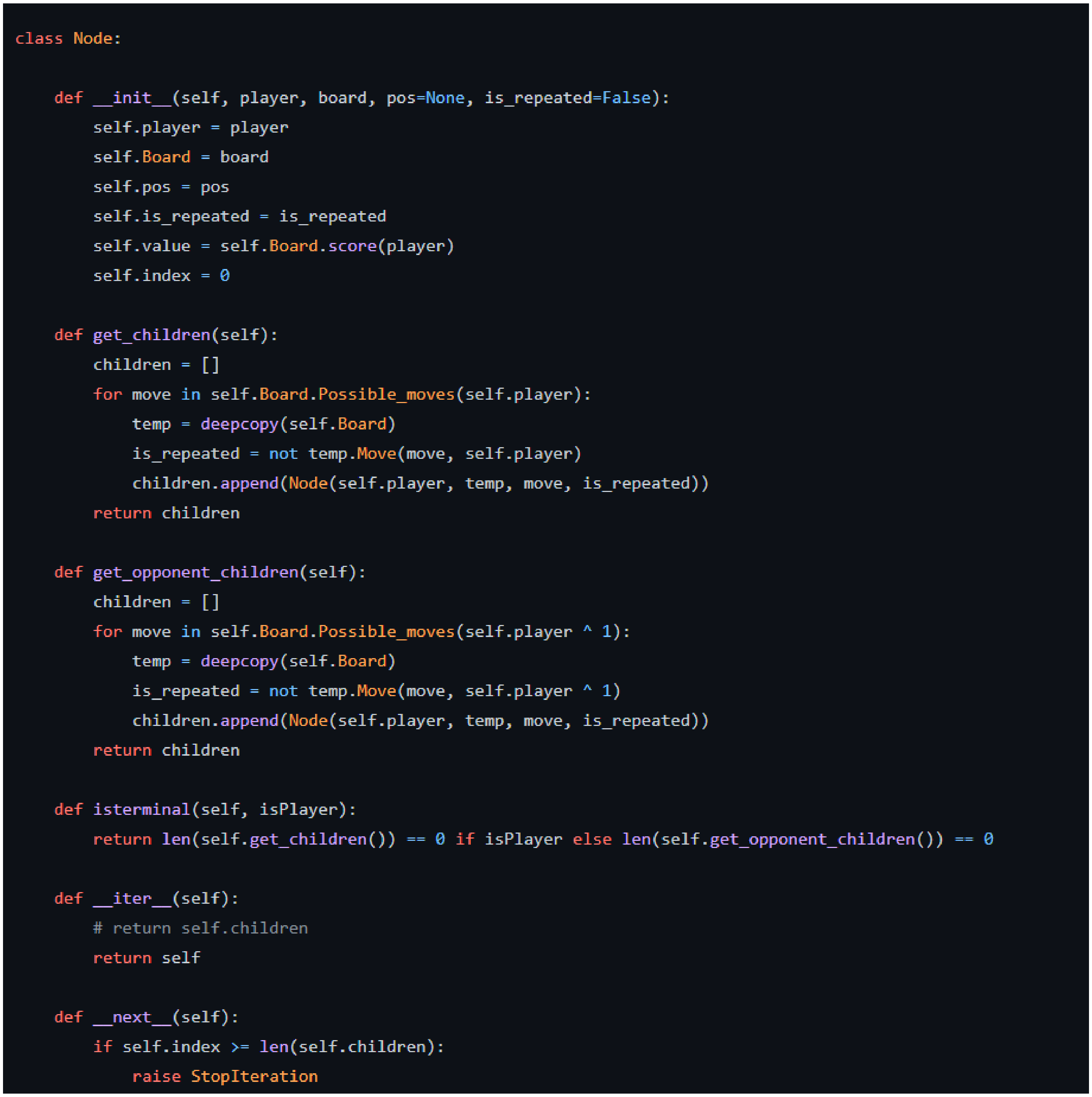
This function at the end returns the value, which is the beta value since it is a minimizer, and also returns a list sorted in ascending order to place the smallest value on the left to get the best pruning which is also a bonus feature.



```
            break

        if time.time()-start_time>threshold+10:
            return value,best_pos
    return value, best_pos
```

we have implemented iterative deepening version of the minmax algorithm, such that if the search process exceeds certain time, it will return with the best move found so far (A Bonus Feature).

Node Class:



```python
class Node:

    def __init__(self, player, board, pos=None, is_repeated=False):
        self.player = player
        self.Board = board
        self.pos = pos
        self.is_repeated = is_repeated
        self.value = self.Board.score(player)
        self.index = 0

    def get_children(self):
        children = []
        for move in self.Board.Possible_moves(self.player):
            temp = deepcopy(self.Board)
            is_repeated = not temp.Move(move, self.player)
            children.append(Node(self.player, temp, move, is_repeated))
        return children

    def get_opponent_children(self):
        children = []
        for move in self.Board.Possible_moves(self.player ^ 1):
            temp = deepcopy(self.Board)
            is_repeated = not temp.Move(move, self.player ^ 1)
            children.append(Node(self.player, temp, move, is_repeated))
        return children

    def isterminal(self, isPlayer):
        return len(self.get_children()) == 0 if isPlayer else len(self.get_opponent_children()) == 0

    def __iter__(self):
        # return self.children
        return self

    def __next__(self):
        if self.index >= len(self.children):
            raise StopIteration
```

This Node Class is Responsible for “drawing” the Search Tree for the AI to Search Using the Min-Max Algorithm, with Alpha-Beta Pruning.

The Search Tree is drawn by 2 Functions, `get_children` and `get_opponent_children`, as both moves exist in the tree.

This File also includes a Function called `isterminal`, which determines whether the node has any children of not.

The Node class contains the following attributes:

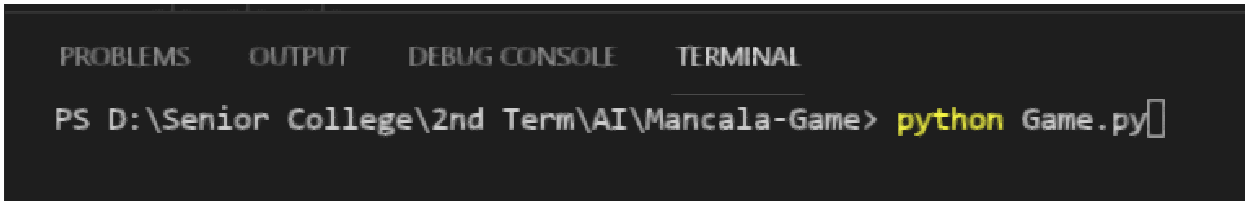
- `player`**: the id of the player who will play the next move (0 or 1)**
- `board`**: the current board**
- `pos`**: the move that leads to the board.**
- `is_repeated`**: a flag to know whether the next move will be the player's move or the opponent's move.**
- `value`**: the current value of the board, based on the player of the node, and the score function.**

The Node class also contains the following functions:

- `get_children`**: returns a list of nodes, one for each possible move that can be made by the player of the node**
- `get_opponent_children`**: returns a list of nodes, one for each possible move that can be made by the opponent player**
- `isterminal`**: returns a flag determining whether this is the final move of the game or not, depending on the player parameter.**

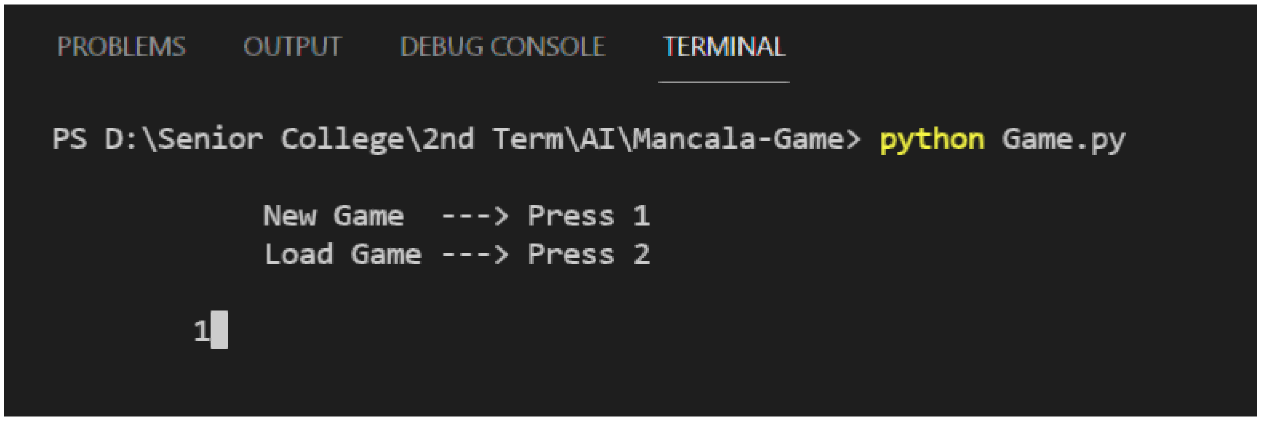
User Guide:

1. **Make sure you have python installed.**
2. **Extract the zip file downloaded from the repo**
3. **Open a terminal in the folder where you extracted the project, and write "python Game.py"**



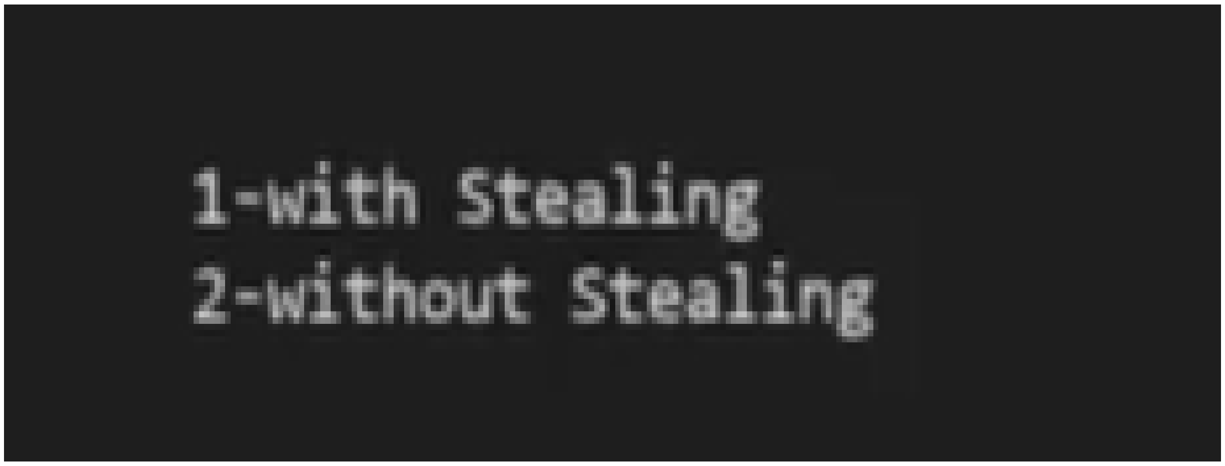
```
PS D:\Senior College\2nd Term\AI\Mancala-Game> python Game.py
```

4. **In the terminal, First enter 1 or 2 to select new game or load game.**



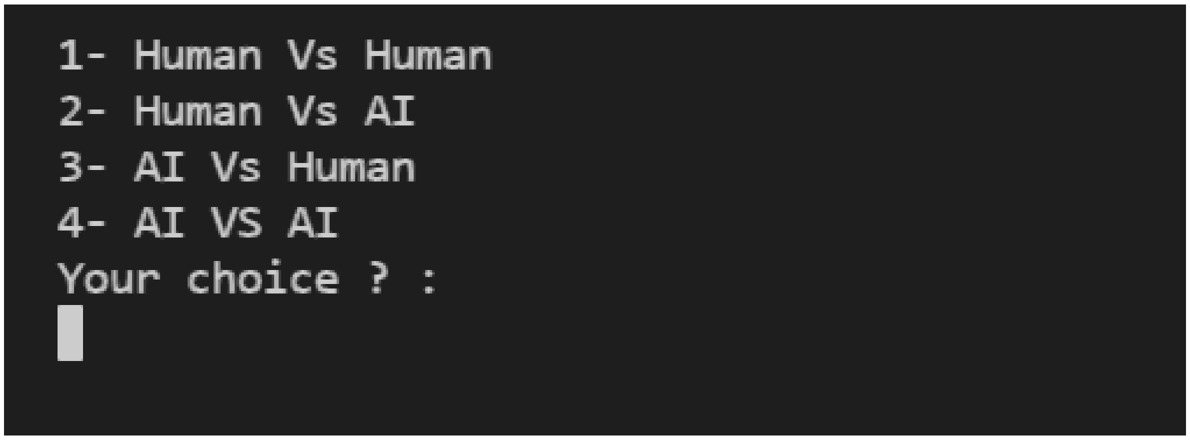
```
PS D:\Senior College\2nd Term\AI\Mancala-Game> python Game.py

            New Game  ---> Press 1
            Load Game ---> Press 2

        1
```

5. **Choose between with Or without stealing modes.**



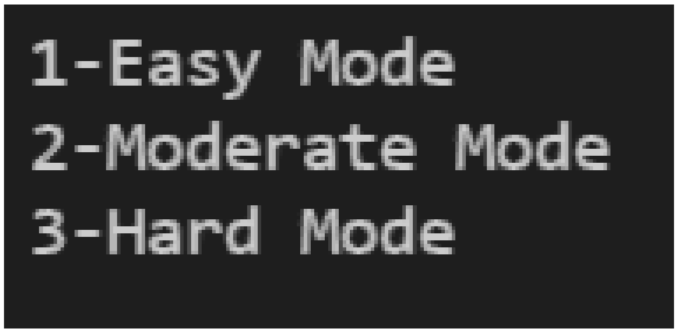
```
1-with Stealing
2-without Stealing
```

6. **If new game is chosen, Enter 1,2,3 or 4 to select game mode**



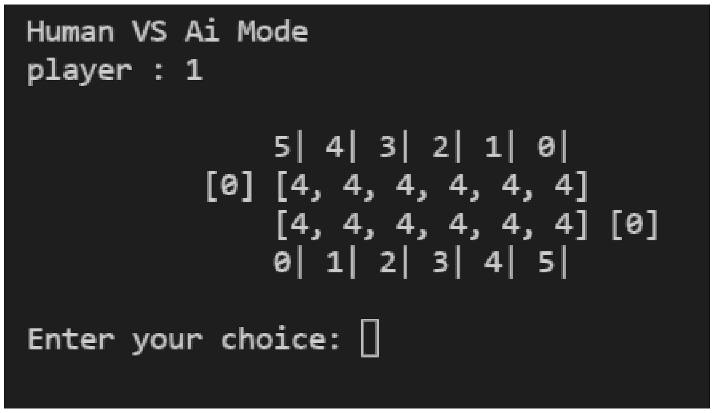
```
1- Human Vs Human
2- Human Vs AI
3- AI Vs Human
4- AI VS AI
Your choice ? :
```

7. **If you chose any mode other than Human vs Human, Enter 1,2 or 3 for difficulty setting.**



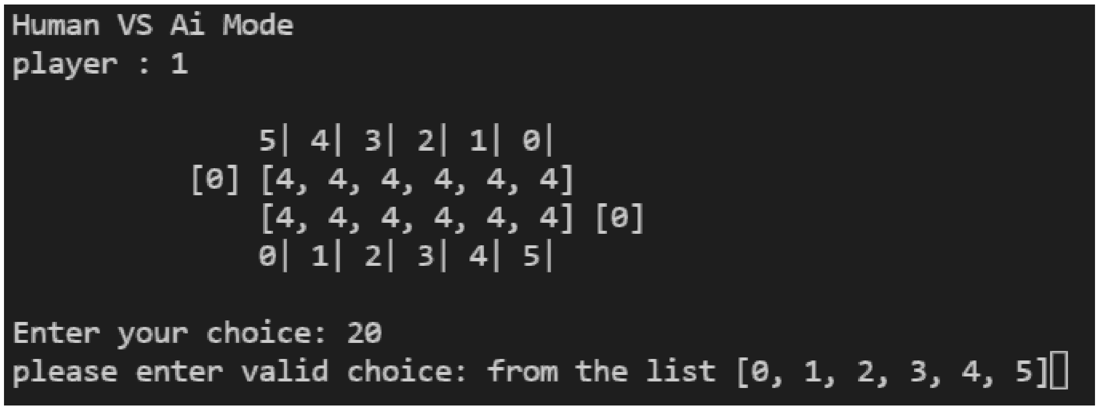
```
1-Easy Mode
2-Moderate Mode
3-Hard Mode
```

8. **Enter your choice position from valid positions.**



```
Human VS Ai Mode
player : 1

           5| 4| 3| 2| 1| 0|
       [0] [4, 4, 4, 4, 4, 4]
           [4, 4, 4, 4, 4, 4] [0]
           0| 1| 2| 3| 4| 5|

Enter your choice:
```

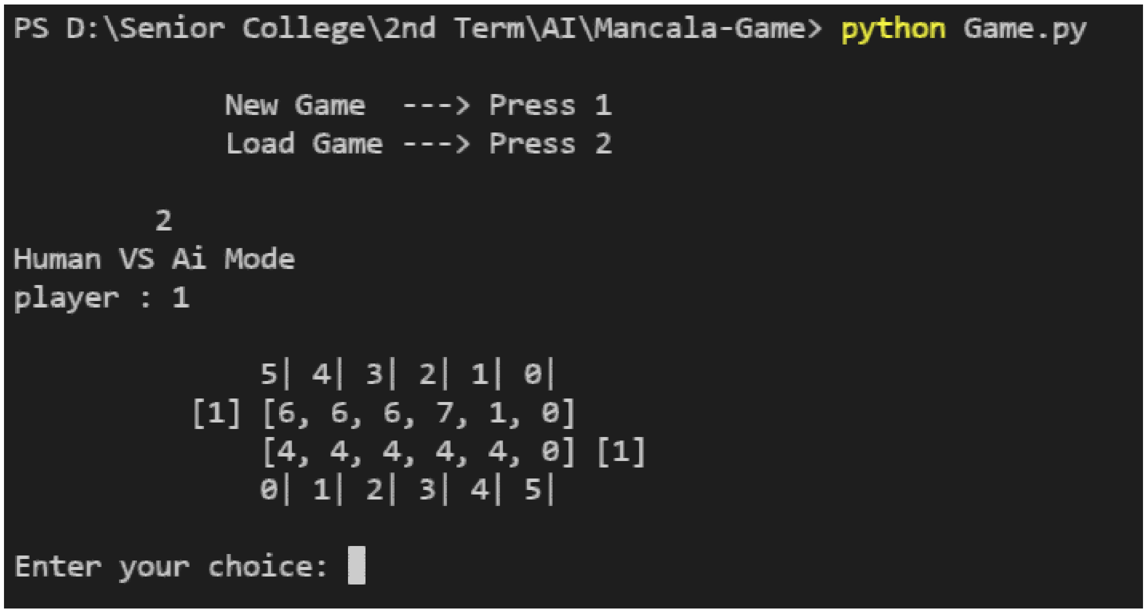
9. If your choice is invalid, a list of possible choices will appear



```
Human VS Ai Mode
player : 1

               5| 4| 3| 2| 1| 0|
           [0] [4, 4, 4, 4, 4, 4]
               [4, 4, 4, 4, 4, 4] [0]
               0| 1| 2| 3| 4| 5|

Enter your choice: 20
please enter valid choice: from the list [0, 1, 2, 3, 4, 5]
```

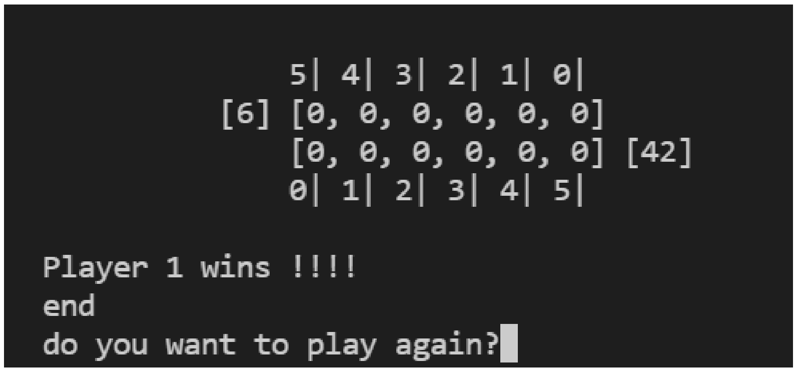
   a. To exit a game before it is finished, simply close the script, or press Ctrl-C.

10. In case you left a game before it is done, you can simply load the last state by choosing load game after you restart the script.



```
PS D:\Senior College\2nd Term\AI\Mancala-Game> python Game.py

            New Game  ---> Press 1
            Load Game ---> Press 2

        2
Human VS Ai Mode
player : 1

               5| 4| 3| 2| 1| 0|
           [1] [6, 6, 6, 7, 1, 0]
               [4, 4, 4, 4, 4, 0] [1]
               0| 1| 2| 3| 4| 5|

Enter your choice:
```

11. **When a game is done, the winner will be displayed, and the script will ask if you want to play again, enter y to play again, or any key to exit.**



```
           5| 4| 3| 2| 1| 0|
       [6] [0, 0, 0, 0, 0, 0]
           [0, 0, 0, 0, 0, 0] [42]
           0| 1| 2| 3| 4| 5|

Player 1 wins !!!!
end
do you want to play again?
```

Project Work:

Girges micheal: implemented (Game.py - Board.py) files.
Ziad Tarek: Contributed in ''MinMax'' Algorithm with optimization as well as AI algorithm.
David john: Contributed in the "Node" File, the "AI Algorithms" File, as well as the integration of the code.
Ahmed Taha Fekry: various difficulty level support, iterative deepening optimization with timer and Integration of the project Files.
Mina wagdy: Integration, General Testing with Corner Cases, Support Saving and Loading of the game and Documentation of the whole project.