Mancala Game

Program:

Course Code: ***CSE481***

Course Name:

Artificial Intelligence

Examination Committee

Dr. Manal Morad Zaki
Eng. Ahmed Fathy
Eng. Ola Hamdy

Ain Shams University
Faculty of Engineering
Spring Semester – 2020

Student Personal Information for Group Work

Student Names:	Student Codes:
Aly Moustafa El-Kady	1600844
Aya Tarek El-Ashry	1600370
Lama Zeyad Ibrahim	16E0134
Yara Mohamed Zaki	1601672

Plagiarism Statement

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student Name: **Aly El-Kady/ Aya El-Ashry/ Lama Zeyad/ Yara Zaki** **Date:** **12/6/2021**

Table of Contents

Game description

Mancala is a two-player turn-based board game. It consists of 6 pits, 24 stones and a mancala for each player. Each mancala must be on the right side of the player. The player should sit opposite to each other and play in a counterclockwise direction.

As setup, each pit should contain 4 stones except for the mancala. On each turn, the player will take all the stones from any of his/hers pit and deposit one stone in each pit. The player should skip his/her opponent's mancala. If the player's last stone lands in his/her own mancala, the player may take another turn.

The Mancala game has two modes: stealing and no stealing. Stealing mode is If the player's last stone lands on an empty pit, the player should capture the stones in his/her own pit and the stones in his/her opponent's pit directly across the player's pit.

Game is over when one of the player's pits are all empty, by then the other player can capture all the stones left in his/her pits and place it in his/her mancala. The winner is the player with the most stones in his/her mancala.

Implementation

The Mancala project is implemented using Python3 by Spyder IDE, the mancala algorithm is based on minimax decision rule with alpha-beta pruning. The player plays against an AI player using the console. The implementation consists of 3 source files: mancala, minimax and main.

Each of these codes perform certain functions. The mancala file consists of 4 functions: print_board, getwinner, gameover and move. The minimax file consists of a class called Node which contains 4 functions: getStates, buildTree, minimax and print_moves. Lastly, the main file integrates both last 2 files and implement the game flow.

Functions

I. Mancala Project Functions

i. ToString()

ToString takes as an argument the list - which is the board and returns it as a string.

ii. Center_Drawing_List(), Center_Drawing_String(), Center_Drawing_String_Circles(), Center_Drawing_String_Null()

These functions are used to utilize the terminal printing.

iii. Print_board()

print_board takes as an argument the board and prints it for the player to visualize the state of the board after every game played.

iv. gameOver()

gameOver takes as an argument the board and checks if one of the 2 players has all his/her pits empty which means the game is over. It returns 1 if game is over and 0 otherwise.

v. GetWinner()

getWinner takes as an argument the board. It's used to calculate the final score when the game is over, and add any left stones to the player's mancala.

vi. Move ()

move takes as an argument the board, the index and the boolean stealing. It moves the mancala of a certain index following the rules of the game. It can also work by the 2 game modes, if stealing - send stealing = True, otherwise stealing = False. It returns the state of the board after moving the mancala of the argument index and the next player that should play (if no one can play it returns the next player = -1)

The function contains all the game rules covered.

vii. NewGame()

Newgame function prints the initial board of the game. It takes the first player, stealing mode and stealing mode as an input. It returns the board, player, stealing mode and depth limit. It is used later in the main function, its returns is given to **move()** function if player 2 is his/her turn or playturn() function

viii. SaveGame()

Savegame takes next board, next player, stealing and depth limit as arguments. These arguments are saved from the previous game. Basically, it saves them in lastgame.txt as they can be recalled in restore function

ix. RestoreGame()

RestoreGame opens the lastgame.txt file saved by SaveGame. It append the variables saved in text file in game list then it returns game list.

II. Minimax Functions

i. Class Node

This class represents the tree of a certain depth limit of the current state of the board. It initializes the root node with the current state, sets it as a maximizer node, has an empty list of possible states that can be reached from this board, an empty list of next players that should play depending on which path is to be chosen, and sets the alpha and beta to -infinity and infinity.

ii. Getstates ()

Function *getStates* is a method that belongs to class Node. It takes as an argument the stealing mode whether it is true or false. It first checks if the game is OVER, using the *gameOver* function and returns nothing if true. If the node is a maximizer, it loops over index 0 to 5 and calls the *move* function to get all possible states from the node's board. If the node is not a maximizer, it loops over index 7 to 12 and calls the *move* function to get all possible states from the node's board. If there is a state, it creates a Node object from this board and checks if the next_player is the AI it sets the node as a maximizer and appends the possible_state in the list of possible_states and appends the next_player in the list of next_players that were initialized in the node. If there is no state, the move function returns the next player as -1 so as to not append the current board.

iii. buildTree()

Function *buildTree* is a recursive method that belongs to class Node. It builds the tree of the current board up to a certain depth limit (argument passed to the function). It first checks the stopping condition which is whether the depth limit reached zero, if it is true then return and do nothing. Otherwise, get all possible states from the current board using the *getStates* method, and iterate over all the current states that can be reached for this current board and call the *buildTree* function for each state, but decrease the depth limit by 1 for every level. The tree is eventually built successfully with each child ranging from 0 to 6.

iv. Minimax ()

Function *minimax* is a recursive method that belongs to class Node. It checks whether the node is a maximizer or not, in both cases it calls the *minimax* function until it reaches the stopping condition which is whether there are possible states for the current node or not; if not then it is a terminal node, and we calculate and return the score. Back to the original if, the function then compares the score calculated with the best value of the other children's nodes - whether it is the maximum or the minimum. If the score calculated is better than the best, we set it as the best new score, the current state as the new best state and the next player of this path. The function then checks the alpha and beta values to break the loop if alpha is bigger than or equals to the beta. Function then returns the score, the best board state, and the next player.

v. Printmoves()

Function *printMoves* is a method that belongs to class Node. It simply loops over the direct children of the current node to print them. It was implemented only for error checking.

vi. PlayTurn()

Function to initialize a Node object of the current state of the board and call the methods needed to play a turn (*buildTree, minimax*). It takes as an argument the board, the depth limit, and the game mode (stealing or no stealing).

III. Main

Finally, the main function which is the core and integrates all the files. You must run the main function to start the game. It starts with welcoming message, asking if you want to start new game or load a previous game then asking for the difficulty level and whether you want to start playing or let AI play first.

The function basically consists of while loop which keeps on looping till the **gameover() is true** . It checks if player 1 (AI) is next player then it calls the **Playturn()** function implemented in minimax file returning the next player and next board. If player 2 is the next player then it calls **move()** which is implemented in mancala_project file returning new board and new player.

After each turn, it prints out if you wish to save this game and continue later. If you type Y for yes, it calls savegame(). You can then rerun the main and choose load your previous game.

Finally, When the game is over. We announce the winner by calling **getwinner()** implemented in mancala_project file.

Bonus Features

The extra Features added are:

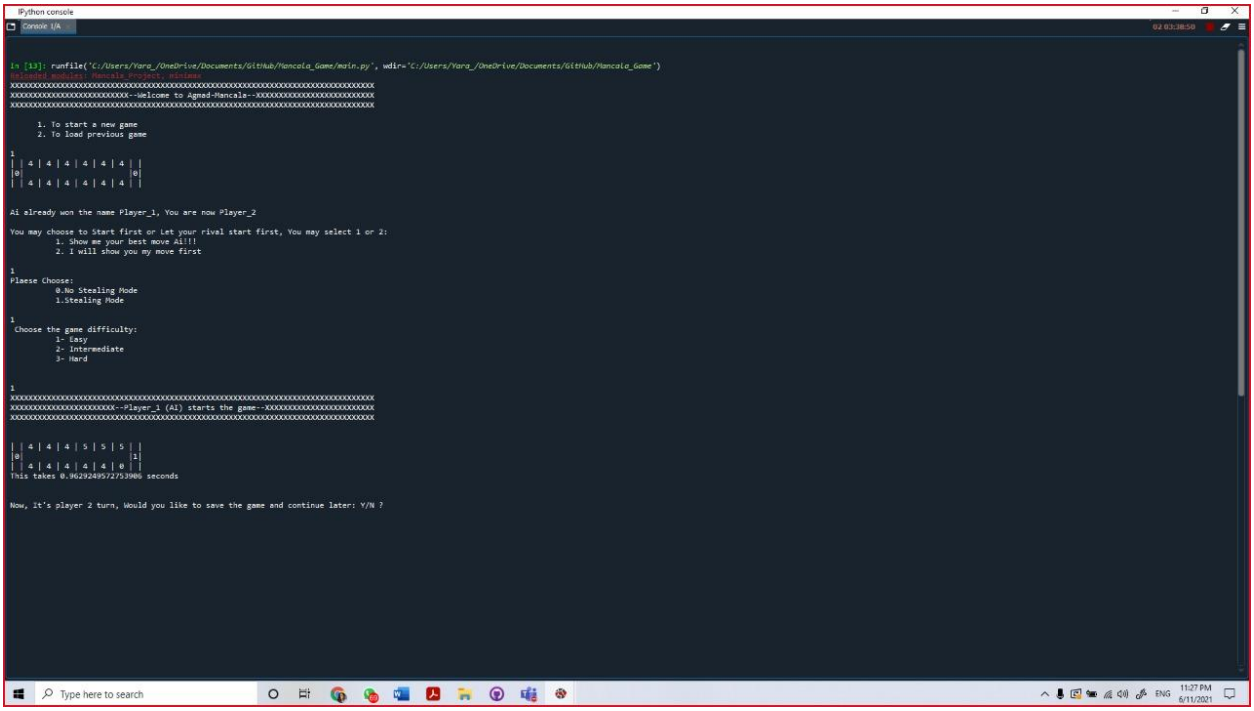
1- Saving and loading a previous game. It is displayed after each turn.
2- various difficulty levels: easy, intermediate, and hard. They depend on the depth limit of the game tree. The easy mode goes up to 7 levels which takes around 1 seconds, The intermediate mode goes up to 8 levels taking around 5 seconds and the hard mode goes up to 9 levels taking around 23 seconds.

User Guide with Snapshots

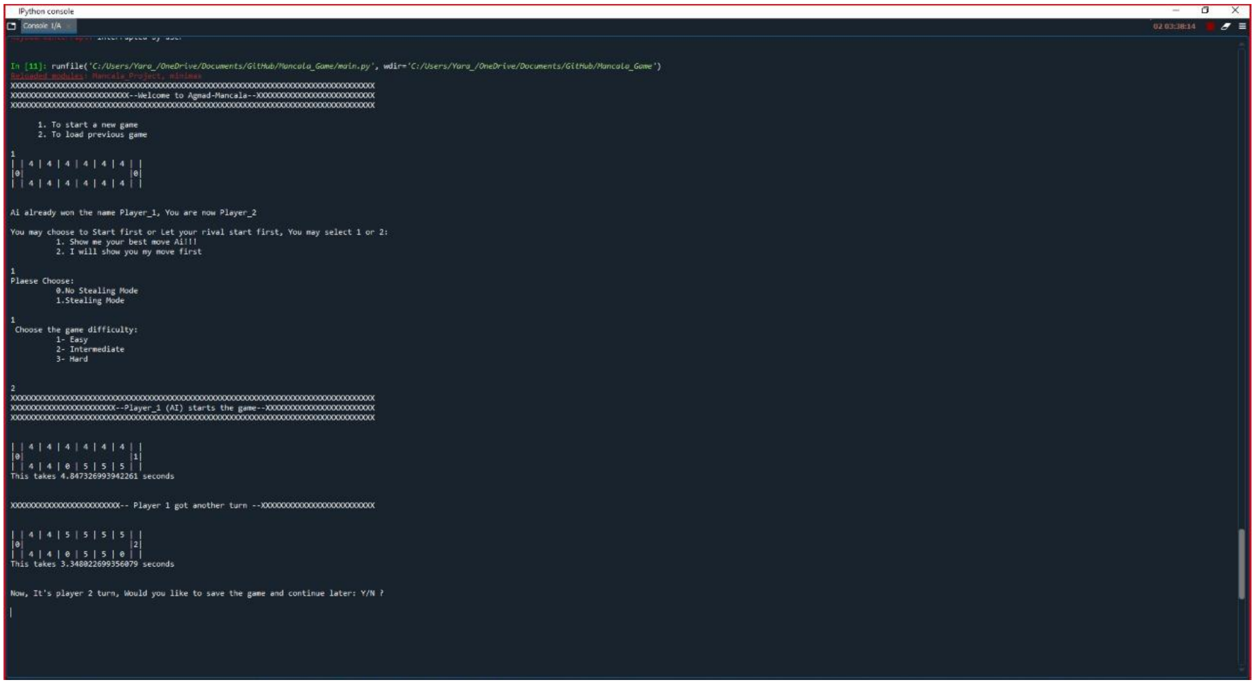
First, when you run the game, it asks you whether you want to start a new game or to load a previous game, then it prints the initial board in the 2 cases then it asks who you want to start the game whether it's the AI or it's you, then to choose the mode (Stealing or No Stealing), then to choose the game difficulty (Easy, Intermediate, Hard)

- For each difficulty level, we calculate the time which the function takes to reach the best state:

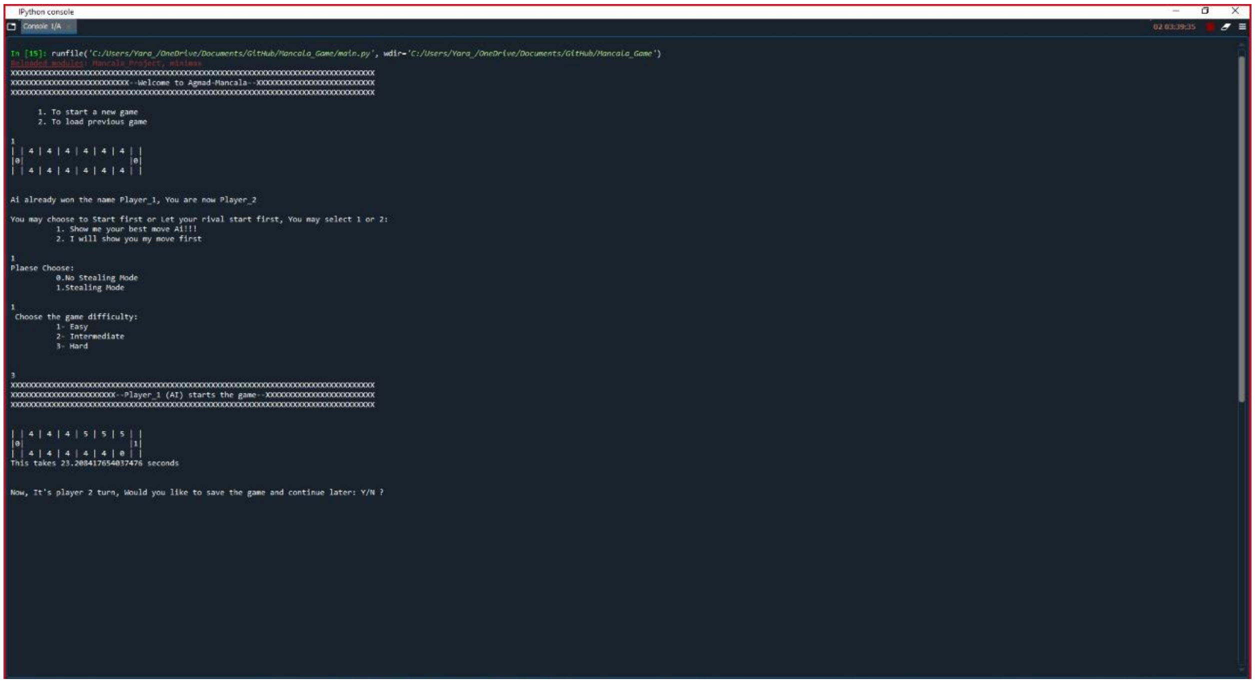
1- **For Easy difficulty (Depth=7)** ➔it takes less than 1 second.



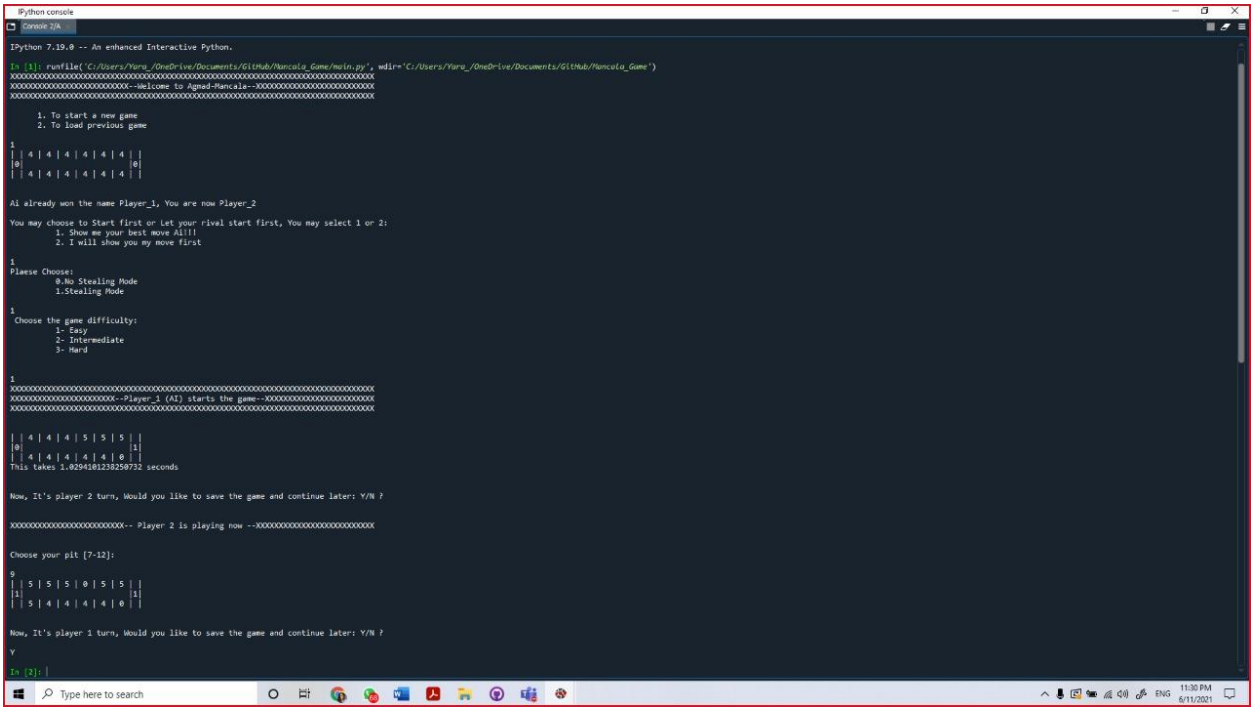
2- **For Intermediate difficulty (Depth=8)** ➔ it takes less than 5 seconds.

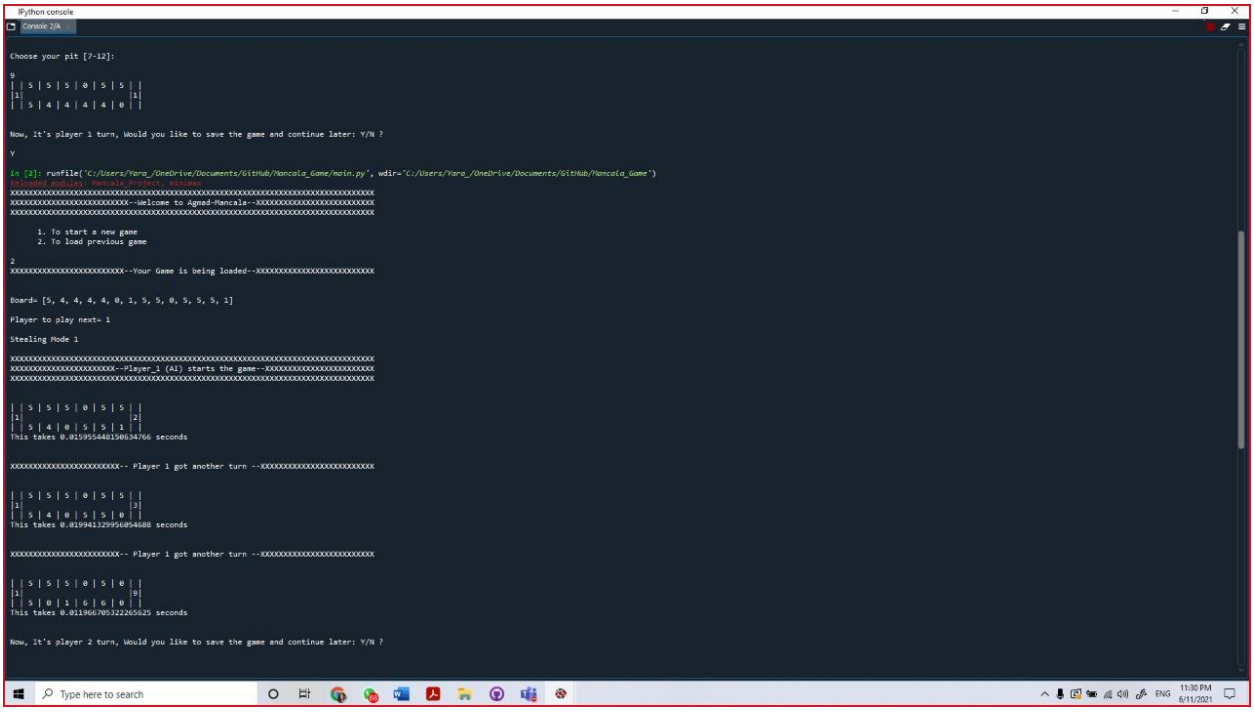


3- **For Hard difficulty (Depth=9)** ➔ it takes around 24 seconds.



- **For Saving and loading a previous game**





Work distribution

Aly Mostafa:

- Implemented move, print_board, save, restore function.
- Unit testing.
- Integration testing.

Lama Ziad:

- Implemented minimax file.
- Unit testing.
- Integration testing.

Aya Tarek:

- Implemented the main.
- Implemented getwinner function.
- Unit testing.
- Integration testing.

Yara Zaki:

- Implemented GameOver function.
- Implemented the main.
- Unit testing.
- Integration testing.

additional documentation

GitHub Repo: https://github.com/alymostafa1/Mancala_Game

YouTube Video Link: https://www.youtube.com/watch?v=xORjiU3u55g

Illustration Video for our AI model VS Website model:
https://drive.google.com/file/d/1t1cF2x3FQI3ic5A642rpGfNwuEYvVDzR/view?usp=sharing