RESEARCH & PROJECT SUBMISSIONS

Program: Computer & Control Systems Department

Course Code: ***CSE481***

Course Name:

Artificial Intelligence

Examination Committee

Dr. Manal Mourad

Ain Shams University
Faculty of Engineering
Spring Semester – 2020

Student Personal Information for Group Work

Student Names:	Student Codes:
Adham Hesham Hamed	1600227
Reem Mohamed Abd El-Rouf Mady	1600593
Amr Yasser Mahmoud El-Gamal	1600949
Ahmed Abd El-Salam Helaly	1600125

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 been 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: **Adham Hesham / Reem Mohamed / Amr Yasser / Ahmed Abd El-Salam** **Date:** **12/6/2021**

Submission Contents

Table of Contents

Brief Description of the game and implementation.

Game Description:

The object of the game is to capture the most stones.

Setup: layout the board horizontally between two players and place 4 stones into each of the 12 small pockets. Board is divided into two rows of 6 pockets each. Each player controls the 6 pockets on the side closest to them. The two larger pockets are called Mancalas. Each player owns the Mancala to the right of their row. Pick a player to go first. On your turn pick up all the stones in any one of the pockets on your side of the board. Moving counterclockwise from the pocket picked, deposits one of the stones in each pocket you pass until the stones run out. If you run into your own mancala, deposit one stone in it. If you run into your opponent's mancala, skip it. If the last piece you drop is in your own mancala, take another turn immediately. If the last piece you drop is in an empty pocket on your side of the board, you capture that piece and any pieces in the hole directly opposite of it. Always place all captured pieces in your mancala. The game ends when all six spaces on one side of the board are empty. The player who still has pieces on his side of the board when the game ends, capture all those pieces and puts them into their mancala. Count all the pieces in each mancala. The player with the most pieces, wins.

Github link: https://github.com/adhamhesham97/Mancala

Youtube link: https://www.youtube.com/watch?v=Ln8SYJpMLxQ

Implementation:

- We implemented our game using python language.
- The board appears in the console.
- We implemented two game modes; stealing and no stealing.
- The user gets to choose to **play with the computer** or not (**computer vs computer**)
- We implemented the option to save the game by saving the current board, the player's turn, and the game mode.
- The load function as well to load the previous saved game.
- We implemented the Time limit and play now options.
- We implemented a computer vs computer game

The program first asks the user if he/she wants to play first. The player choose a pocket in his turn while the computer chooses a pocket using the minimax alpha beta algorithm, ending up with the best pocket to play with for this turn.

The game continues till someone wins or a tie.

Detailed description of the implemented functions

For the game to function with several modes and players options, we implemented the code via multiple functions where each function plays an important role in playing the mancala game.

Functions that have been used are :

- **Save (board,player,stealing) :**
  It basically takes the current game state of the board, player and the stealing and saves them in a certain place in the memory with a path annotated as 'f' using the " pickle.dump() " function from the imported library " pickle ".
- **Load() :**
  It loads the last state of the game's board, player and game mode saved in the memory and returns their values in the variables (board, player, stealing).
- **move(board, pocket, stealing=True) :**
  this function does four things to play a certain move by a certain player, which takes us to the first thing : identifying which player is playing based on the current given pocket number (if the pocket is from 0 to 5, this means it's the first player that's playing, otherwise it's the second player). The second thing is to actually move the stones without defying the game rules like skipping the side pockets (mancalas). Basically we start by storing the number of stones in the current pocket in a variable " stones " and setting a skip variable initially with zero while setting the next pocket as the current pocket, this skip variable is then set to 1 in case we're standing on one of the other players side pocket (mancala). The third thing this function does is taking the game mode into consideration (weather we're playing with stealing or not). The fourth thing is enabling the ability for a player to play again if they stopped at their mancala. Finally the function returns the board state (how many stones in each pocket on the board) and the next player.
- **isValidMove(board, pocket) :**
  This function simply returns weather the current pocket is empty or not so that the move is a valid one.
- **score(board, player) :**
  takes the board state and player number and calculates this player's score.
- **findWinner(board) :**
  this function basically calculates which player has more stones in their mancala, if player one has more stones, player one wins and the value returned is '1', if player two has more stones, player two wins and the value returned is '2'. In case the two players have the same number of stones in their mancala, this means it's a draw game and the value returned is '3'. The board state is returned after it has been reset along with the winning player.

- **minimax(board, player, stealing, depth, alpha, beta, maximizingPlayer) :**
  this is one of the most important functions which is responsible for choosing the best scenario for a certain player using the minimax and alph/beta method where it maximizes the chosen player's chance to win the game. This algorithm sets weather each level is a maximizer or a minimizer, starts at the top level (level 0) with apha = - infinity and beta = infinity then goes down to the left and starts changing their values based on the level's type (maximizer or minimizer). The function first check if the game's time is up, then check if there are no possible moves before starting the minimax algorithm.
- **bestPocket(board, player, stealing=True, depth=10) :** this function uses the minimax function and applies the fact that it chooses the best move to be played and get the maximum score possible at each move. First, it shows the player the time limit, then calls the minimax function and checks if the next step has higher score than the current one. If the next step's score is higher, this move will be the next move, otherwise it'll be skipped and see the next one and then applies the same thing. If the keyboard is pressed, the function prints "Ctrl-C is pressed".
- **Display(board,x):**
  It takes the previous board and the player number. And prints out the current board.
- **getInput(board,player,stealing):**
  takes the current board, player number & game mode.
  it displays the board to the user on the console
  and asks him/her to choose the location of the pocket to play or to save the game.
  if he enters "s", the game is saved.
  if the users enters an invalid character or chooses an empty one, he's asked to reenter the pocket again.
  this function returns **-1** if the user chose to save the game or returns the **pocket number.**
- **inputStealing():**
  this function asks the user to enter the game mode, returns a boolean "stealing" whether it's true or false.
- **inputPlaying():**
  asks the user whether he/she wants to play with the computer or not.
  if yes, it then asks if he/she wants to save the gam (flag=0 -> not saved, flag=1 -> saved)
  it returns a Boolean "playing": represents whether the player is going to play or not
- **inputPlayerNumber():**
  This function returns the order of the players (who is going to play first)
  if the game is saved before, then it gets the first player number
- if it is not saved then it asks the user if he wants to play first
- **inputimeLimit():**

  Asks the user if he wants to change the time limit. if yes then the user enters a new limit if not then it remains equals to 10 secs. it then returns the time limit.

- **playerVScomputer():**

  this function is mainly the game where the user plays with the computer, first it checks which user to play first and whether the user wants to play or no.
  and then the game starts each one plays in his turn. When it's the user turn he's asked to choose a pocket and the the board changes and a new one is shown in the console. When it's the computer turn a minimax algorithm is done, so we end up with the best pocket to choose. After the game Is finished, it prints who wins in the end.

- **computerVScomputer():**

  this function plays the game by the computer against itself, it doesn't ask the user about any info to enter. It just continue to play by choosing the best pocket to pick every time using the minimax alpha beta algorithm and then it returns who wins, player 1 or 2 or if it's a tie

Bonus:

1. Save / load
2. Time limit, play now, and Iterative deepening in MINIMAX
3. Computer vs computer

User Guide with snapshots

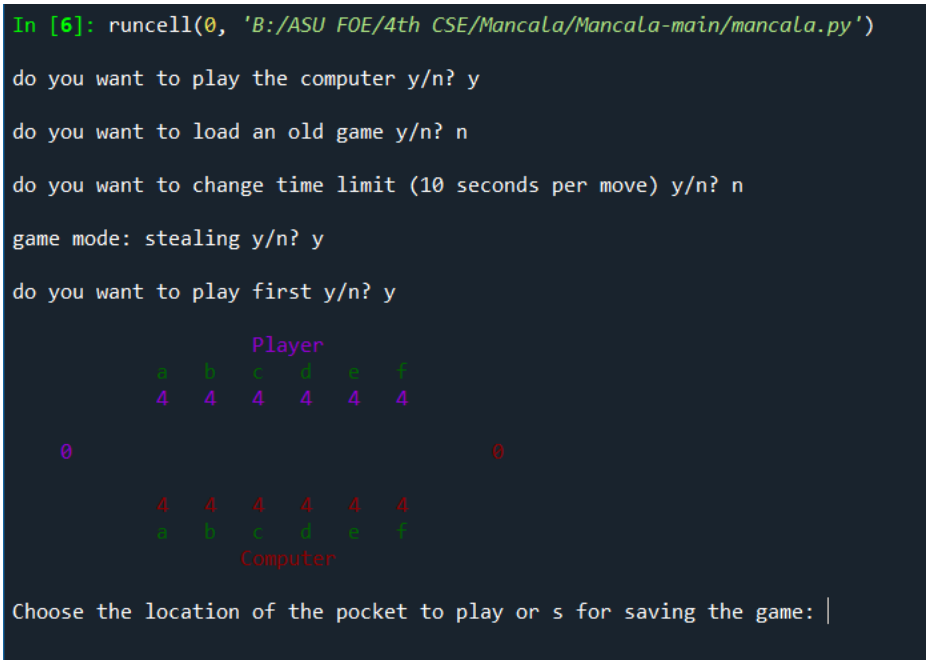
The user guide is represented in the console.

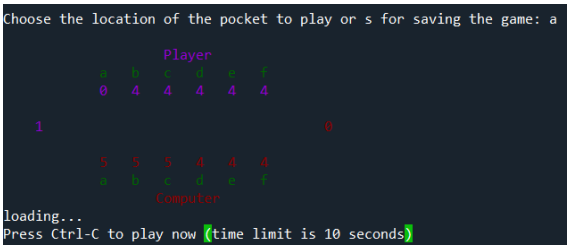
First the user is asked whether to play with the computer or not

Then if he wants to load a previous game and to change the time limit if he wants to.

Game mode and If he wants to play first.

Then the board is shown to the user, and he's asked to choose a pocket.





Then its time for the computer to play.

If I entered a wrong pocket or if I chose an empty pocket, it then asks me to enter a right one

```
computer played pocket a

                    Player
            a   b   c   d   e   f
            0   4   4   4   4   4

    1                                   0

            0   6   6   5   5   5
            a   b   c   d   e   f
                   Computer

Choose the location of the pocket to play or s for saving the game: 3

Enter the right pocket: a

Enter the right pocket:
```

If we chose to let the computer play itself, it continues to play turn after turn

```
In [7]: runfile('B:/ASU FOE/4th CSE/Mancala/Mancala-main/mancala.py', wdir='B:/ASU FOE/4th
CSE/Mancala/Mancala-main')

do you want to play the computer y/n? n

do you want to change time limit (10 seconds per move) y/n? n

game mode: stealing y/n? n

                   Player1
            a   b   c   d   e   f
            4   4   4   4   4   4

    0                                   0

            4   4   4   4   4   4
            a   b   c   d   e   f
                   Player2
loading...
Press Ctrl-C to play now (time limit is 10 seconds)
player 1 plays pocket d

                   Player1
            a   b   c   d   e   f
            5   5   5   0   4   4

    1                                   0

            4   4   4   4   4   4
            a   b   c   d   e   f
                   Player2
loading...
Press Ctrl-C to play now (time limit is 10 seconds)
```

```
                Player1
          a   b   c   d   e   f
          1   2   4   3   8   0

    14                                9

          0   0   0   0   0   7
          a   b   c   d   e   f
                Player2
loading...
Press Ctrl-C to play now (time limit is 10 seconds)
player 2 plays pocket f

                Player1
          a   b   c   d   e   f
          0   0   0   0   0   0

    38                                10

          0   0   0   0   0   0
          a   b   c   d   e   f
                Player2
player 1 wins
```

If we entered an invalid input we detect it.

```
do you want to play the computer y/n? y

do you want to load an old game y/n? g
invalid input

do you want to load an old game y/n?
```

Here we saved the game and reload it again

```
Choose the location of the pocket to play or s for saving the game: s

                Player
          a   b   c   d   e   f
          0   4   4   4   4   4

    1                                 0

          0   6   6   5   5   5
          a   b   c   d   e   f
                Computer

Choose the location of the pocket to play or s for saving the game: s

In [13]: runcell(0, 'B:/ASU FOE/4th CSE/Mancala/Mancala-main/mancala.py')

do you want to play the computer y/n? y

do you want to load an old game y/n? y

do you want to change time limit (10 seconds per move) y/n? n

                Player
          a   b   c   d   e   f
          0   4   4   4   4   4

    1                                 0

          0   6   6   5   5   5
          a   b   c   d   e   f
                Computer

Choose the location of the pocket to play or s for saving the game:
```

Work Distribution

Adham Hesham SalahElDin Hamed: move(), minimax(),score(), bestPocket(), playerVScomputer, computerVScomputer, inputStealing(), inputPlaying(), inputTimeLimit(), inputPlayerNumber(),isValidMove()

Reem Mohamed AbdElRaouf Mady: save(), load(),display(), findWinner(), getInput(),

Ahmed AbdElSalam Mhamoud Helaly: report, testing

Amr Yasser Mahmoud ElGaml: report, testing, video