Ain Shams University

Faculty of Engineering

CSE481: Artificial Intelligence

Intelligent Mancala Game

Members:

Name	ID
Manal Ahmed Mohamed	1601449
Nesma Mohamed Atef	1601579
Hala Mohamed Ahmed Shaheen	1601655
Yassmen Mahmoud Ali	1601685

1. A brief description of the game and of your implementation including any bonus features included.

   A brief description of the game: Mancala Game is a two-player game, each one has 6 pockets in front of him and his mancala, each pocket has 4 stones in it, and he is allowed to choose any pocket of the 6 to make his move (as long as this pocket is not empty), The goal is to collect the biggest number of stones in your own mancala.
   The game has 2 modes, with stealing and without stealing, without stealing means that the player cannot steal stones from his opponent pockets no matter what the situation, with stealing means that the player can steal from his opponent pockets ONLY IF he placed his last stone in an empty pocket (of his own pockets) and the opposite opponent pocket's has stones in it, so the player

will take the stone he put in the empty place and the stones in the opposite opponent's pocket and put them in his mancala.

A brief description of our implementation: we first build a human vs human version to make sure that we understand the game rules right, then we moved to the minimax with alpha-beta pruning to make the AI chooses the best available move in a given depth, then we update the board based on the game rules and the mode of the game.

BONUS Features :

1- we provided multiple difficulty levels(easy -meduim - hard) , in the easy level we use depth = 2, so the AI decision will be some how shallow, in higher levels we increase the depth , so the AI performs better moves.

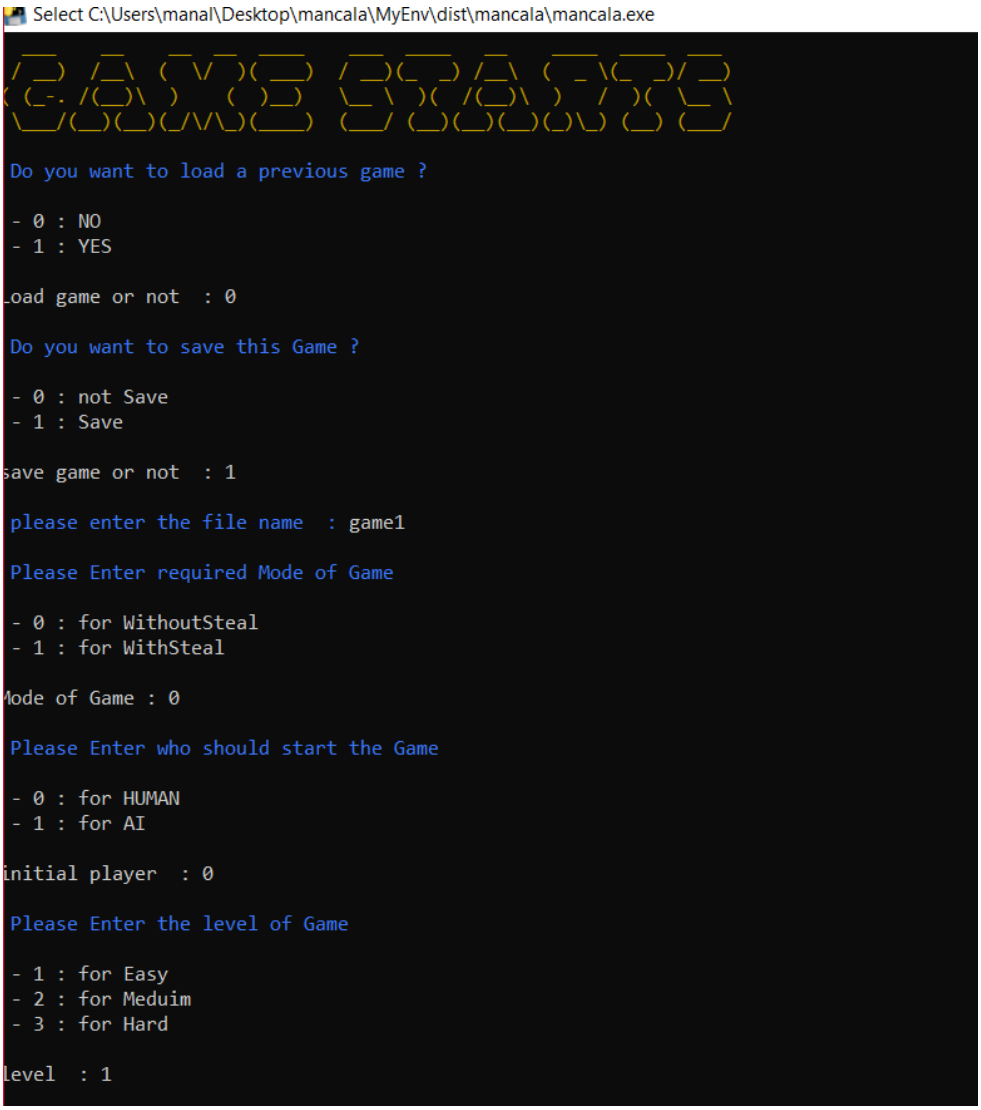
2- we provided a utility function for saving and loading the game including the all previous information of the game [the player who start playing, mode (with/without stealing)- level].

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

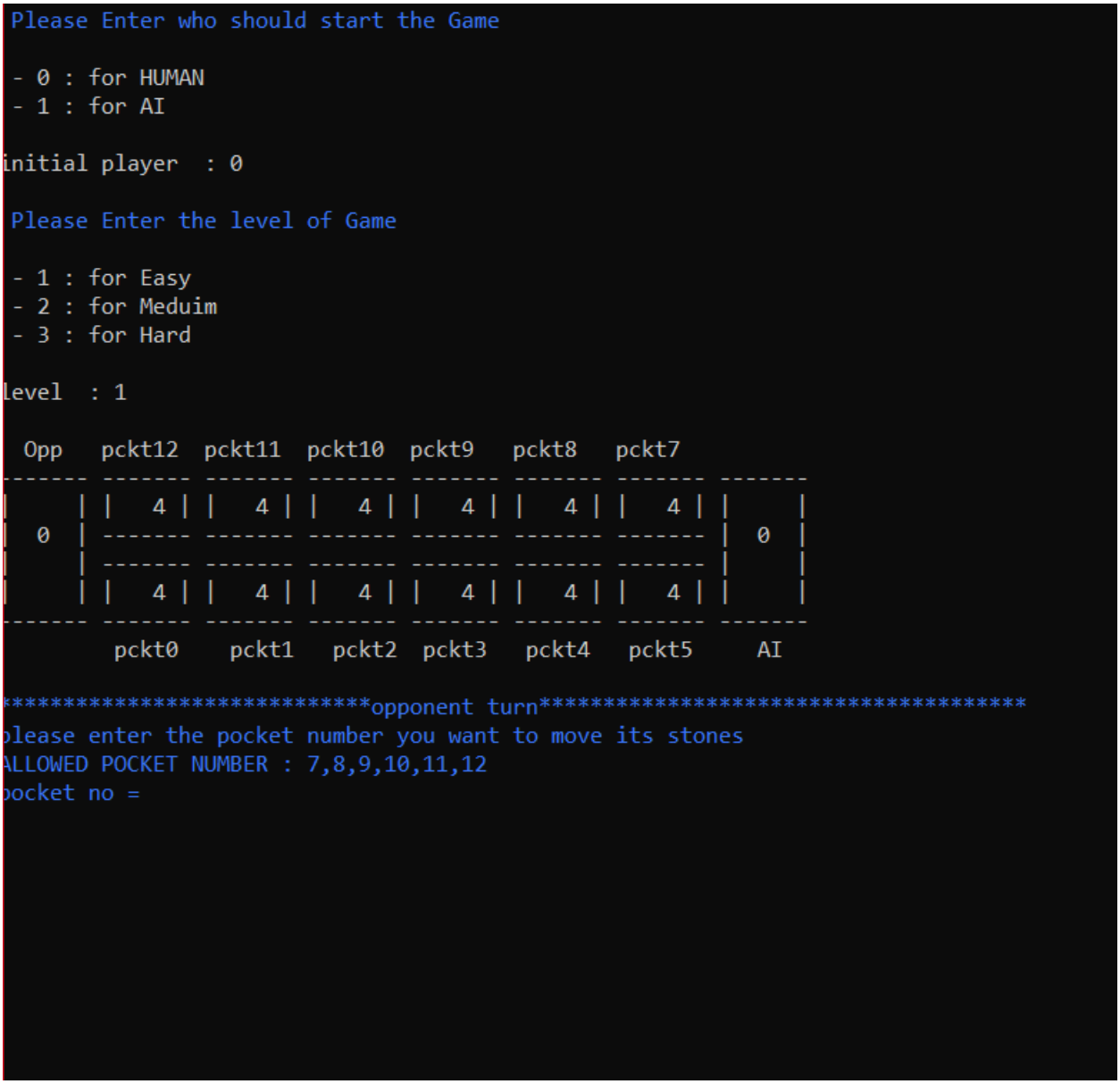
We first used a very shallow evaluation function, which gives score 1 when the AI's mancala has more stones than the opponent's mancala, -1 when the AI's mancala has less stones than the opponent's mancala, 0 if they both have the same number of stones in each mancala, then we improved it by using a more effective evaluation function which returns a score of AI's mancala number of stones – opponent's mancala number of stones, it will return a positive number if the AI is the winner and a negative number if the AI is the loser, 0 if they are equal (which never happens if the AI starts the Game because he plays optimally) " IF the AI plays first, we never win in any game mode (stealing-no stealing)/easy-medium-hard"

3. A user guide with snapshots.

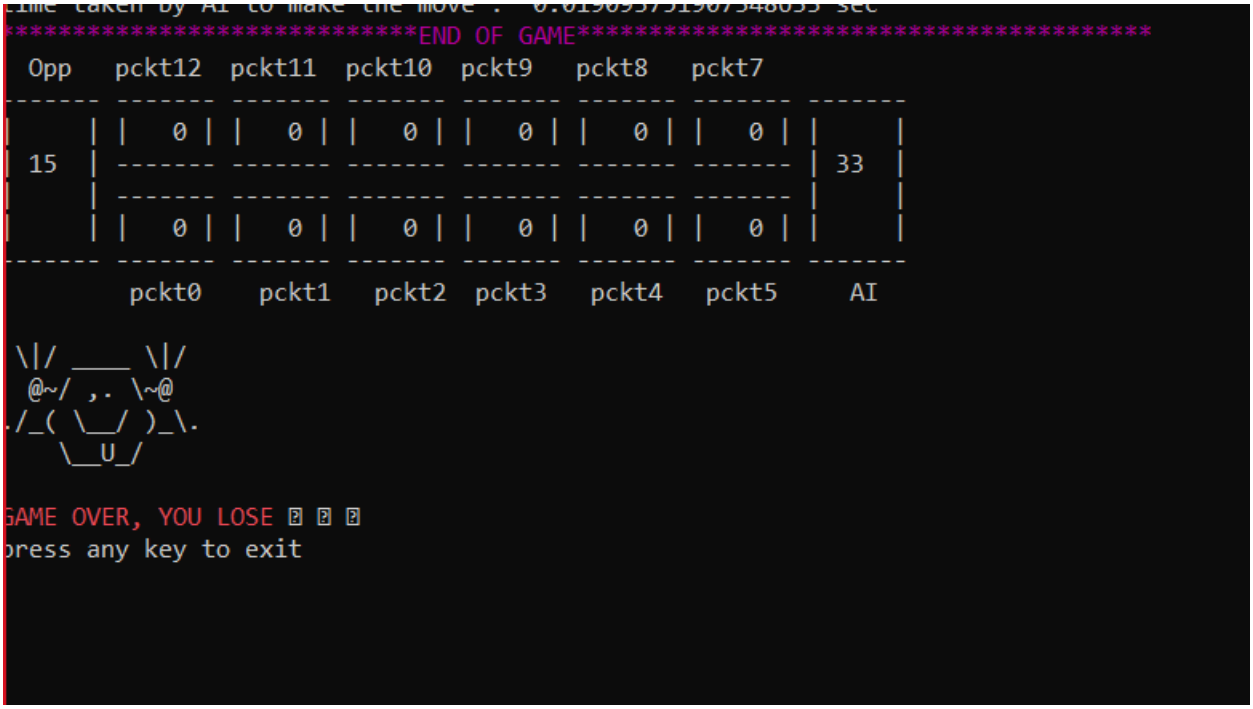
When you first run the executable file, the game options will appear to choose what you want:



Based on your choices the game will start and the mancala board will appear



Continue playing the game until you win or lose



Then press enter to exit the game

4. A summary of how the work was split among your team members (who did what exactly)

   First, we made a brainstorming about the overall code implementation then we divided the code into functions and divided those functions on us

Name	Functions
Manal Ahmed Mohamed 1601449	Minimax playAgain evaluation functions Savegame Loadgame printMancala
Nesma Mohamed Atef 1601579	updateBoard isSteal updateBoardWithSteal isMancalaIndex lastStoneIndex
Hala Mohamed Ahmed Shaheen 1601655	AIMove endgame printSteal 3 levels function Checkwinner printWinner
Yassmen Mahmoud Ali 1601685	oppMove stopPlay isPocketEmpty AllPocketsEmpty

	Winner Loser

5. Any additional documentation you might find useful (including code documentation, descriptions of difficulties encountered, tricks used, etc.)

   The code includes good use of comments for every function using multiline comments:

```
def printMancala(mancala_board):
    """
    parameters: the current mancala board you want to print
    it prints the mancala in a good way to see how the current state of the mancala is
    returns: None
    """
```

```
def evaluate(board):
    """
    parameters: mancala current board
    it returns the difference of the number of stones in the AI's mancala and opponent's mancala,
    +ve means AI is the winner, -ve means AI is the loser
    return: the score (AI's mancala number of stones - opponent's mancala number of stones)
    """
```

```
def playAgain(board,pocket_index):
    """
    parameters: mancala board, the played pocket index
    this function checks if the last stone is placed in the player's mancala so he should play again
    retrun: True if he should play again, false if not
    """
```

```
def minimax(board,mode,depth,depthMax,player,alpha,beta):
    """
    paramters: mancala board, mode of the game, depth of the tree, player: true if AI false if
    opponent, alpha and beta parameters
    this is the main function of the whole game, it returns the best score that AI can acheive by
    making a certain move, it searches the tree to a certain depth
    determined based on the level of the game, it uses alpha beta pruning to cut off the
     unnecessary branches to make the search faster
    return: best score
    """
```

```
def saveGame(fileName,board,player,mode,level):
    """
    parameters: the file name you want to save the game in, the board you want to save,the last
     played player, the mode of the game and the level
    it takes the paramters and save them in a list game in a text file
    returns: None
    """
```

```
def loadGame(fileName):
    """
    parameters: the file name you want to load the game from
    it opens the file and gives the saved list again to continue playing from the stopped position
    returns: gamelist
    """
```

```
def endGame(board):
"""
Check if condition of end of game is met or not
parameters:mancala board
returns: True if all pockets of both players are empty,false otherwise """
```

---

def printWinner(board):

"""

check for condition of winning and display a message showing who is the winner

parameters: board

returns: none

"""

---

def printSteal(prevBoard,currBoard,pocket_index):

"""

checks if any of two players steal or not and display the message showing which player is stealing

parameters: prevBoard:board before move

currBoard:board afer move

pocket_index:index of the played pocket causing the move

returns:none

"""

---

def AIMove(board,mode,depthMax,printBoard):

"""

performs AI move based on decision evaluated by minimax algorithm and evaluation/utility function

parameters: board:mancala board

mode:stealing or without stealing

depthMax:maximum depth of AI tree

printBoard:boolean represents when the board needs to be printed to avoid redundant prints

returns:best score and updated board

"""

---

def easyLevel(board,mode,printBoard):

def meduimLevel(board,mode,printBoard):

def hardLevel(board,mode,printBoard):

"""

difficulty levels : is all about varying depth to make AI make better movement

level: Easy - Meduim -hard

"""

---

def isMancalaIndex(player,lastIndex):

"""

checks whether the lastIndex due to the player move is its mancala index

parameters: 1- player:opponent or AI

2- lastIndex:last move performed by player

returns: true if it is the player mancala index , false if not

"""

---

def isSteal(board,lastIndex,player):

"""

checks for the condition of stealing : the laststone is placed in an empty pocket belonging to the same player

and the opposite pocket of the other player is non-empty

parameters: 1- board:mancala board

2- lastIndex:last move performed by player

3- player:opponent or AI

returns: true if condition of stealing is met otherwise false

"""

---

def lastStoneIndex(board,pocket_index):

"""

find the last stone index due to move performed by player

parameters : 1- board:mancala board

2- pocket_index : pocket played by player

returns : the last position due to this move

"""

---

def updateBoardWithSteal(board,pocket_index):

"""

update mancala if the condition of steal is met

parameters: 1-board:mancala board

2-pocket_index : pocket played by player

returns: the updated board

"""

---

def updateBoard(board,pocket_index,mode):

"""

update board due to move and including if condition of stealing is met

parameters: 1-board:mancala board

2-pocket_index : pocket played by player

returns: the updated board

"""

---

def isPocketEmpty(board,pocket_index):

"""

check whether the pocket is empty or not

parameters: 1- board:mancala board

2- pocket_index : the index of the pocket

returns: true if the pocket at that index is empty or not

"""

```
def AllPocketsEmpty(board,player):
"""
check whether the all mancala pocket are empty or not
parameters: 1- board:mancala board
        2- player:opponent or AI
returns: true if the all pockets are empty or not

"""
```

```
def stopPlay(board,player):

"""

returns whether stop condition is true or false

parameters: 1- board:mancala board

        2- player:opponent or AI

returns: true if the stop played condition is met otherwise false

"""
```

```
def oppMove(board,mode,endOfGame):

"""

perform the opponent move based on the choosen pocket

parameters: 1- board:mancala board

        2- mode:with/without stealing

        3- endOfGame:boolean represents whether the game ends or not

returns: the updated board after opponent move

"""
```
