Mancala

12.06.2021

Name	ID
Ibrahim Hassan Ibrahim Mohamed	1600008
Eslam Ahmed Abdelhamid	1600252
Khaled Atef Abdelaziz Mansour	1600504
Taha Mohamed Taha Rageh	1600714
Omar Abdel-Baset Abdel-Maqsoud	1600888

Game Description

General description

Mancala is a generic name for a family of two-player turn-based strategy board games played with small stones, beans, or seeds and rows of holes or pits in the earth, a board or other playing surface. The objective is usually to capture all or some set of the opponent's pieces.

Our Implementation

The game is implemented in python respecting modularization and OO standards to run on linux specially **Ubuntu 18.4 LTS**.

1- Either player or computer could start

2- user can choose to play stealing mode

3- User can save game to load it and continue later

4- Time limit is supported

5- Force play now is supported

Implementation

I. Mancala Class

The Mancala class is encapsulate the game related variables and methods

```
class Mancala():
    """docstring for Mancala"""
    def __init__(self, Initial_State = None,is_Stealing = True, turn = 'max'):
        super(Mancala, self).__init__()
        self.state = []
        if Initial_State is None:
            self.state = [4,4,4,4,4,4,0,4,4,4,4,4,4,0]
        else:
```

```
            self.state = Initial_State

        self.Stealing = is_Stealing
        self.turn = turn
        self.valid_move = True
        self.previous_move = -1
        self.cost = self.cost_Function()
```

The class has 4 methods:

is_end() : this method checks whether the game has ended or not

cost_Function() : calculate the cost of the given object

print() : print the game state in the command line

move() : change the state of the game based on the selected move

II. Tree

The Tree is built recursively by first scanning through the available moves and then recursively appending the new states as children to the original state. The recursion stops when the depth is decremented to zero

```
    def createTree(self,root,depth):

        if depth == 0:
            return root
        Valid_moves = range(0,6) if root.mancala.turn == 'min' else range(7,13)
        for move in Valid_moves:
            Next = root.mancala.move(move)
            child_Node = self.createNode(Next.state,Next.turn,move)
            root.future_states.append(self.createTree(child_Node,depth - 1))
        return root
```

To get the best move we use mini_max with alpha-beta pruning method. We traverse the tree by DFS and for each node we calculate its score by calling the same function recursively for all of its childs and take the max value to be the cost (alpha)of this node if the node is maximizer and the min value to be the cost (beta) of this node if the node is minimizer we prune nodes that when alpha is bigger than or equal beta

```
def alpha_beta(self,node, alpha, beta):
        if node.future_states == []:
```

```
            return node.mancala.cost

        if node.mancala.turn == 'max':
            temp = float('-inf')
            for child in node.future_states:
                child.mancala.cost = self.alpha_beta(child, alpha, beta)
                temp = max(temp, child.mancala.cost)
                alpha = max(alpha, temp)
                if alpha >= beta:
                    node.pruned = 1
                    break  # beta cutoff
            node.mancala.cost = temp
            return temp

        elif node.mancala.turn == 'min':
            temp = float('inf')
            for child in node.future_states:
                child.mancala.cost = self.alpha_beta(child, alpha, beta)
                temp = max(temp, child.mancala.cost)
                beta = min(beta, temp)
                if beta <= alpha:
                    node.pruned = 1
                    break  # alpha cutoff
            node.mancala.cost = temp
            return temp
```

III. Libraries

The external libraries used in this code are:

1 - csv: to save or load the game in a csv file.

2 - os: which is also used in saving or loading.

3 - datetime: to save the time of the saved game.

4 - time: to calculate the time of the computer movement to ensure that it won't exceed 30 seconds.

5 - signal: gives alarm that it reached 30 seconds that the computer must stop playing after.

Bonus

I. Play Now and Limited Time modes

User can force ai model to play now by pressing Ctrl^C

A keyboard interrupt exception are thrown and handled to return last calculated move

Similarly Time limit mode are supported by fire a signal that throw a time limit exception that's interrupt the move calculations (loop) and return last determined value

```
class TimeoutException(Exception):   # Custom exception class
    pass


def timeout_handler(signum, frame):   # Custom signal handler
    raise TimeoutException
```

```
    move = get_move(current_state, 1)
    signal.signal(signal.SIGALRM, timeout_handler)
    signal.alarm(30)
    try:
        while True:
            temp = get_move(current_state, i)
            move = temp
    except TimeoutException:
        return move
```

II. Four Levels of Difficulties

Four level of difficulties are supported by changing the max depth of tree (in case of not choosing either time limit or play now mode)

```
def get_move(state, depth=LEVELS[level]):
    try:
        s = state.copy()
        root = node(s, 'max')
        create_tree(root, depth)
        alpha_beta(root, float('-inf'), float('inf'), 'max')
```

III. Save and Load Game

Game states (board, stealing, interrupt mode, date) are saved/loaded in csv file

User Guide

```
ibrahem@VAIO MINGW64 /f/4th/manacala/Mancala_AI (main)
$ python whole.py
do you want to load game ? (yes = press 1, no = press 0)
0
do you want stealing mode ? (yes = press 1, no = press 0)
1
do you want to start ? (yes = enter 'min', no = enter 'max')
min
choose level of difficulty : 1 - 2 - 3 - 4
3
force play method:
0- let the kid take his time
1-CTRL+C
2- 30s time limit
0
player -->   0 [4, 4, 4, 4, 4, 4]
computer -->   [4, 4, 4, 4, 4, 4] 0
it is your turn Enter a number from 1 to 6 or 0 to save and exit
1
this is your play
player -->   0 [4, 5, 5, 5, 5, 0]
computer -->   [4, 4, 4, 4, 4, 4] 0
--------------------
this is my move
time:  1.96339750289917
player -->   0 [4, 5, 5, 5, 5, 0]
computer -->   [4, 4, 0, 5, 5, 5] 1
--------------------
this is my move
time:  1.5096261501312256
player -->   0 [4, 5, 5, 6, 6, 1]
computer -->   [4, 4, 0, 5, 0, 6] 2
--------------------
it is your turn Enter a number from 1 to 6 or 0 to save and exit
0
```

1. first user is asked if he wants to load a saved game or not

Load list

```
index ['time', 'stealing', 'level', 'irp']
1 ['2021-06-06/08/21 05:49:17', '1', '3', '1']
2 ['2021-06-06/08/21 05:49:37', '1', '3', '1']
3 ['2021-06-06/12/21 21:22:31', '1', '1', '2']
enter game number to be loaded:
```

2. If he want to start a new game then he is asked if he want “stealing mode” or not
3. Then he is asked if he wants to start or wants the AI to start he should write “max” To let AI start or write “min” so that he can start
4. Then the user is asked to choose level of difficulty which affects the depth of the formed tree (level 1: depth=1 ,level 2: depth=3 ,level 4: depth=7,level 5: depth=10)
5. Then the user is asked to select a mode for force play (0: wait until AI find the best move, 1: when pressing ctrl c the AI will play (play now button), 2: AI will play after 30 sec)

Play now mod example (CTRL+C)

```
player -->   0 [4, 4, 4, 4, 4, 4]
computer -->   [4, 4, 4, 4, 4, 4] 0
it is your turn Enter a number from 1 to 6 or 0 to save and exit
2
this is your play
player -->   0 [5, 5, 5, 5, 0, 4]
computer -->   [4, 4, 4, 4, 4, 4] 0
--------------------
this is my move
3
^Ctime:  33.592453956604004
player -->   0 [5, 5, 5, 5, 0, 4]
computer -->   [4, 4, 0, 5, 5, 5] 1
--------------------
this is my move
^Ctime:  8.811818361282349
player -->   0 [5, 5, 6, 6, 1, 5]
computer -->   [4, 4, 0, 5, 5, 0] 2
--------------------
```

```
5^Ctime:  1.7935364246368408
player -->   0 [5, 5, 5, 6, 1, 5]
computer -->   [4, 4, 0, 5, 0, 6] 2
--------------------
it is your turn Enter a number from 1 to 6
5
this is your play
player -->   1 [6, 0, 5, 6, 1, 5]
computer -->   [5, 5, 1, 5, 0, 6] 2
--------------------
this is my move
^Ctime:  2.115905284881592
player -->   1 [6, 0, 5, 6, 1, 5]
computer -->   [5, 0, 2, 6, 1, 7] 3
--------------------
this is my move
^Ctime:  0.837761640548706
player -->   1 [6, 0, 5, 6, 1, 5]
computer -->   [0, 1, 3, 7, 2, 8] 3
--------------------
```

Time limit mode example

```
2- 30s time limit
2
player -->   0 [4, 4, 4, 4, 4, 4]
computer -->   [4, 4, 4, 4, 4, 4] 0
it is your turn Enter a number from 1 to 6 or 0 to save and exit
1
this is your play
player -->   0 [4, 5, 5, 5, 5, 0]
computer -->   [4, 4, 4, 4, 4, 4] 0
--------------------
this is my move
time:  30.000553846359253
player -->   0 [4, 5, 5, 5, 5, 0]
computer -->   [4, 4, 0, 5, 5, 5] 1
--------------------
this is my move
time:  30.000282287597656
player -->   0 [4, 5, 5, 6, 6, 1]
computer -->   [4, 4, 0, 5, 0, 6] 2
--------------------
```

6. Then the game begins in this example the user played first and his play is showed then the AI played two times as he chose the move that will make him play twice by dropping last stone in his mancala

7. At each time the player's turn comes he is asked if he wants to save the game by pressing zero.

Roles

Name	Role
Eslam Ahmed Abdelhamid	Implemented bonus features
Khaled Atef Mansour	Implemented mancala class
Ibrahim hassan	Alpha-beta with pruning , get move
Omar abdelbaset	Create Tree , Node class , is_end()
Taha Mohamed Taha	Cost function, code assembly