2021

Mancala Game

NAMES:

MINA TALAAT ZAKI	1601535
MINA ADEL AZIZ	1601536
MICHEL BOULES BOTROS	1501547
MINA MIKHAEEL YACOB	1601541
MINA HAMDY BADER	1601532

description the game:

Rules:

1) the game board consist of 14 holes 6 for each player and 2 mancalas.
2) player who stores larger number of stones in his/her mancala wins the game.
3) game has two modes: stealing/non-stealing.
4) each player chooses a hole from its valid holes, take the stones in this chosen hole, and add one stone in each neighboring hole.
5) if the last stone in this turn lands in player's mancala, the player takes another turn.
6) In stealing mode, if the last stone landed in one of player's valid holes which contains zero stones, the player steals all stones from opposite opponent hole and added it to its mancala.
7) each player should not add any stone in his opponent mancala.
8) the game ends when all holes of any player become empty.
9) at the end of the game all remaining stones in holes of each player is added to his mancala.

Implementation:

Play:

arguments :

1) List : which holds the number of stones in each position right now
2) Input_pos : the next position to be chosen by any player.
3) Player : whose turn is right now
4) Mode : with stealing or without

Output:

1) Copy_list: updated list
2) Player : who is the next player to play
3) New_turn: a flag to indicate whether there is a player who is going to play again.

After this function is executed, the list is updated after the player chose the input position.

What does this function do?

We have three cases :

1) Normal case : when the player chose a position, and no special case will happen ex: a new turn
2) New turn case : happen when the last stone is placed in the player's mancala.
3) Skip case : the stone must skip the opponent's mancala when passing by it

- after we make sure that the player chose a valid position and a non-empty position, we identify what case will happen and we can easily identify that by knowing the input position and the number of stones in that position

- according to our case right now different code will be executed

- in the normal case the list is updated normally

- if the last stone is placed in the mancala the player is given a new turn

- if the stone passed by the opponent's mancala it will skip it without putting a stone in it

The previous steps are for non-stealing mode.

In case of stealing mode

The same steps are done but a new feature is added.

When the last stone is placed in an empty position , this stone will be placed in the mancala as well as the stones in the opposite position of the opponent.

NOTE:

If the player chose a non-valid position an error message is printed → "wrong entry".

If the player chose an empty position another error message is printed → "choose nonempty hole".

minimax algorithm:

arguments:

board, depth, alpha, beta, player, mode, new_turn(flag to indicate wherever the player has another turn).

outputs:

best next move of Ai bot.

the code has three phases.

- if we reached depth zero or the game ended

  we call heuristic function ,send the board_now to get the heuristic of this state.then,we return best next move for this player and the value of heuristic.

- if player == 2 (maximizer level)
  - we should loop over all valid moves for this player at this situation., so we call **all_valid_moves(board_now).**
  - we try each valid move for this player by calling **play(board_now,the_chosen_hole, mode of playing).**
  - we check if new turn flag is one, so we don't change player.
  - recursive call of minimax algorithm but with new board return from this move.
  - we are in maximizer level, so we change in alpha value, so we choose the max between value of alpha now and value returned from minimax algorithm.
  - if the value of alpha changed so we have new best next move for our Ai bot.
  - if value of alpha is more than beta, then a cutoff occurs. Then, we stop evaluating.
  - we return best next move for this player and max between value return from minimax and value reached so far.
- if player == 1
  - we should loop over all valid moves for this player at this situation., so we call **all_valid_moves(board_now).**
  - we try each valid move for this player by calling **play(board_now,the_chosen_hole, mode of playing)** .
  - we check if new turn flag is one, so we don't change player
  - we are in minimizer level, so we change in beta value, so we choose the min between value of alpha now and value returned from minimax algorithm.
  - if value of alpha more than the value of beta. then, cutoff occurs.
  - we return best next move for this player and max between value return from minimax and value reached so far.

All_valid _moves :

arguments:

Board_now, player.

outputs:

list contains all valid holes for this player.

What does this function do?

determines valid moves based on the current player playing.

Game ended:

arguments:

Board_now.

outputs:

True or false

What does this function do?

determines if the game ended when one side is full of zeros and returns true and false.

Heuristic:

arguments:

board_now.

outputs:

heuristic value.

What does this function do?

calculates the heuristic function which is in our case the difference between both mancalas.

Main algorithm:

- initializes the board [0, 4, 4, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4]
- start with first player.
- the player starts to choose to play against Ai's bot or another human player.
- the player selects mode(stealing or not).
- if player choose to play against Ai , he should choose difficulty level(easy ,medium, high).
- player choose to load latest saved game or start a new game.
- if player choose load(l) we open a txt file and load game state (board, player,mode,difficulty,playervsAi)
- if player vs player so we ask the player to enter number of hole or choose to save the game.
    - if s is entered then we open a txt file and save game status.
    - we call **play(board, chosen_hole, player, mode of game)** .
    - function return board after these move and which player to play next move.
    - we check if the game ends, so we decide which player won.
- if player vs Ai we ask player to enter number of hole or s to save game:
    - if s is entered then we open a txt file and save game status.
    - we call **play(board, chosen_hole, player, mode of game)** .
    - function return board after these move and which player to play next move.
    - we check if the game ends, so we decide which player won.
    - we call **minimax(a copy of board, depth , alpha, beta, player, mode, new_turn).**
    - it returns best next move for our Ai bot.
    - we call **play(board, best_move, player, mode of game)** to play Ai turn .
    - we check again if game ended. then, decide which player won.

Bonus features:

1) support different difficulty.
2) support save and load.

User Guide:

player vs player:

non_stealing:

```
choose the mode( 0 : PvP / 1 : PvAi)
```

0 -> to play player vs player.

1 -> to play against Ai bot.

```
choose the mode( 0 : non stealing / 1 : stealing)
```

0 -> non stealing

```
to load game press l or p to play new game
```

l -> to load latest saved game.

p -> start new game.

```
choose the position(player 1) OR to save game press s
```

choose number from 1 to 6 .

example:

```
       1    2  3   4   5  6

choose the position(player 1) OR to save game press s 5
       13  12  11  10  9   8
+----+----+----+----+----+----+----+
|    | 4 | 4 | 4 | 4 | 4 | 4 |    |
|  0 |----+----+----+----+---| 1  |
|    | 0 | 0 | 0 | 0 | 0 | 1 |    |
+----+----+----+----+----+----+----+
       1    2  3   4   5  6
```

stealing:

```
choose the mode( 0 : PvP / 1 : PvAi)
```

0 -> to play player vs player.

1 -> to play against Ai bot.

```
choose the mode( 0 : non stealing / 1 : stealing) 1
```

1 -> stealing

```
to load game press l or p to play new game
```

l -> to load latest saved game.

p -> start new game.

```
choose the position(player 1) OR to save game press s
```

choose number from 1 to 6 .

```
      13  12  11  10  9   8
+----+----+----+----+----+----+----+
|    | 4 | 4 | 4 | 4 | 4 | 4 |    |
|  0 |----+----+----+----+---| 0  |
|    | 0 | 1 | 0 | 0 | 2 | 0 |    |
+----+----+----+----+----+----+----+
      1    2  3   4   5  6

choose the position(player 1) OR to save game press s 2
```

example :

enter 2

```
      13  12  11  10  9   8
+----+----+----+----+----+----+----+
|    | 4 | 4 | 4 | 4 | 4 | 4 |    |
|  0 |----+----+----+----+---| 0  |
|    | 0 | 1 | 0 | 0 | 2 | 0 |    |
+----+----+----+----+----+----+----+
      1    2  3   4   5  6

choose the position(player 1) OR to save game press s 2
```

```
choose the position(player 1) OR to save game press s 2
      13  12  11  10  9   8
----+---+---+---+---+---+---+---+
    | 4 | 4 | 0 | 4 | 4 | 4 |   |
  0 |---+---+---+---+---+---| 5 |
    | 0 | 0 | 0 | 0 | 2 | 0 |   |
----+---+---+---+---+---+---+---+
      1   2   3   4   5   6

choose the position(player 2) OR to save game press s
```

you stole all fours stones from your opponent.

player vs Ai bot:

```
choose the mode( 0 : PvP / 1 : PvAi)
```

1 -> player vs Ai

```
choose the mode( 0 : non stealing / 1 : stealing)
```

0 -> non stealing

1 -> stealing

```
choose the difficulty( 0 : easy / 1 : medium / 2 : hard)
```

difficulty level:

0 -> easy

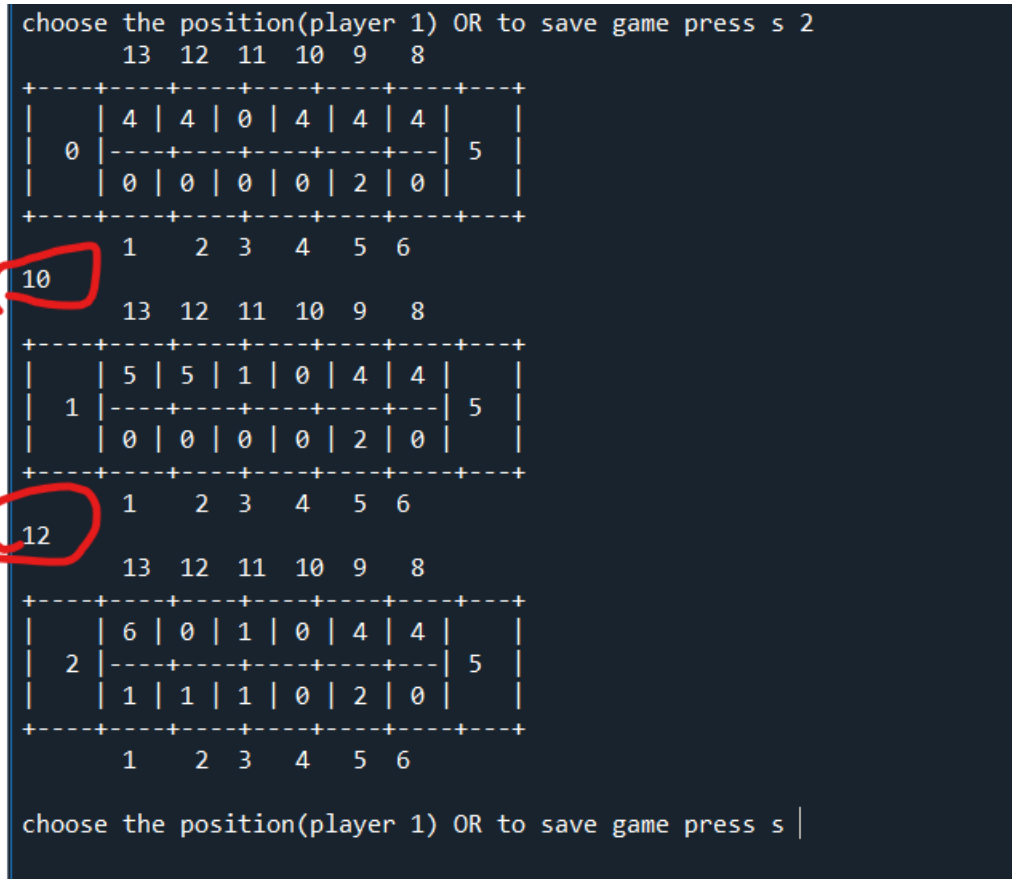
1 -> medium

2 -> hard

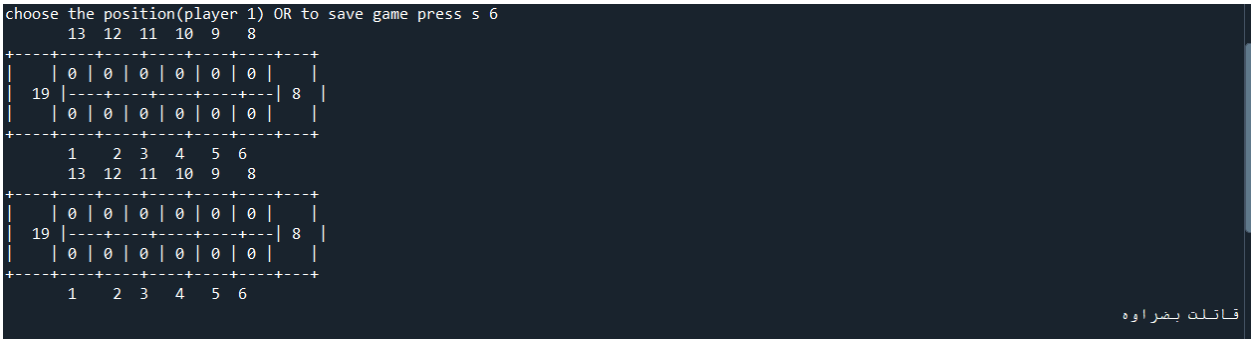
```
choose the difficulty( 0 : easy / 1 : medium / 2 : hard) 0

to load game press l or p to play new game p
       13  12  11  10  9   8
+----+----+----+----+----+----+----+
|    | 4 | 4 | 4 | 4 | 4 | 4 |    |
|  0 |----+----+----+----+---| 0  |
|    | 0 | 1 | 0 | 0 | 2 | 0 |    |
+----+----+----+----+----+----+----+
       1    2  3   4   5  6

choose the position(player 1) OR to save game press s
```

choose number from 1 to 6:

```
to load game press l or p to play new game p
       13  12  11  10  9   8
+----+----+----+----+----+----+----+
|    | 4 | 4 | 4 | 4 | 4 | 4 |    |
|  0 |----+----+----+----+---| 0  |
|    | 0 | 1 | 0 | 0 | 2 | 0 |    |
+----+----+----+----+----+----+----+
       1    2  3   4   5  6

choose the position(player 1) OR to save game press s 2
```



Ai plays two turns according to these moves.



task for each team member:

Michel Attallah: No stealing algorithm and board interface

Mina Adel: With stealing algorithm

Mina Mikhael: Min/max algorithm

Mina Hamdy: Helper functions (Determine when game ends, valid moves, and heuristic function calculation)

Mina Talaat: Combination of algorithms and integration