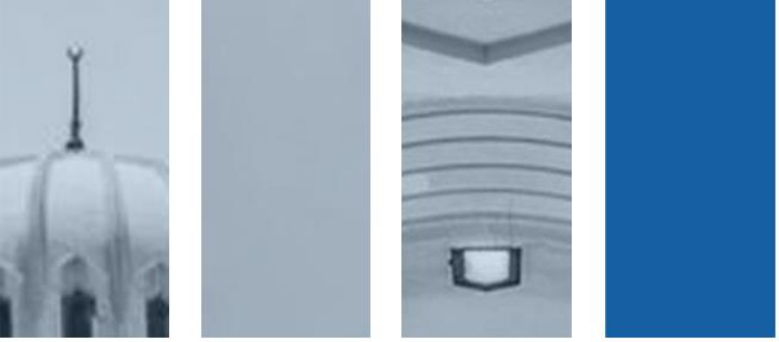


RESEARCH & PROJECT SUBMISSIONS

Program: UG2003

Course Code: CSE 481

Course Name: Artificial Intelligence

Ain Shams University
Faculty of Engineering
Spring Semester – 2020

Student Personal Information

Student Name:	Student Code:
Mohamed Wael Mohamed Alsayed	1501323
Ahmed Mohamed Ebrahim Mohamed	1600155
Mahmoud Othman rabea abdelmohsen	1501358
Mohamed Medhat Mohamed Ali Gado	1501299
Omar abdelshafy Mahmoud	1500909
Mahmoud Ahmed elsayed metwaly	1502016

Please note that Mahmoud Ahmed asked eng Ola to join the time as a extra member because he searched for a team a lot and he did not find one

Submission Contents

01

First Topic

Mancala game

1. Game description

Mancala is a board game that can be played with many variations but for our work here we only examine the version with 2 players.

The board has two sides, each side has 6 holes and a cup. The game starts with 4 pieces in each of those holes and 0 pieces in the cup. The goal of the game is to have more pieces on your side than the opponent.

2. Game rules

- Each player must select a position and move all the pieces in that position, placing a piece in every hole he passes through.
- Player can't move any pieces from the cups.
- If the last piece was placed in a cup, the player gets another turn.
- This continues until no more pieces exist on either side of the board.
- When the game ends, we count all the pieces on both sides. Player with more pieces on his side wins the game.

02

Second Topic

Implementation details

We implemented the game in 2 versions C++ and python.

In C++ version the game was divided into 3 classes: “board”, “tree”, and “game”.

1) The “board” class

```
class board {
private:
    short *arr; bool mode;
    int p1, p2;
public:
    board(bool mod = 0) { ... }
    flag play(short player, short pos) { ... }
    void print() { ... }
    short *state() { ... }
    void copy(board* new_board) { ... }
    short calc_eval() { ... }
    bool empty() { ... }
};
```

- the “board” constructor initizalizes the array with 4 pieces in each position except the cups
- the “play” function takes in 2 parameters (player, position). Player can be either player 0 or 1. Position is a range from (0-5). The function return a flag structure which are 2 boolean values indicating if the move the player did was legal and if the last piece landed in a cup. These flags are used further in the implementation.
- The “print” function prints the board in its current state.
- The “state” function returns a pointer to the array.
- The “copy” function copies the board parameter and sets the internal array to the board values.
- The “calc_eval” function evaluates which player has won by calculating the sum on each side of the board.
- The “empty” function returns 1 if either side of the board is empty indicating an end of the game.

```
int main() {
    board test;
    test.print();
    cout << test.empty() << endl;
    test.play(0, 2);
    test.print();
    test.play(1, 2);
    test.print();
}
```

```
C:\Users\moham_000\Desktop\to_deliver\Debug\to_deliver.exe
0,4,4,4,4,4,4,
4,4,4,4,4,4,0,

0
0,4,4,4,4,4,4,
4,4,0,5,5,5,1,

1,5,5,5,0,4,4,
4,4,0,5,5,5,1,
```

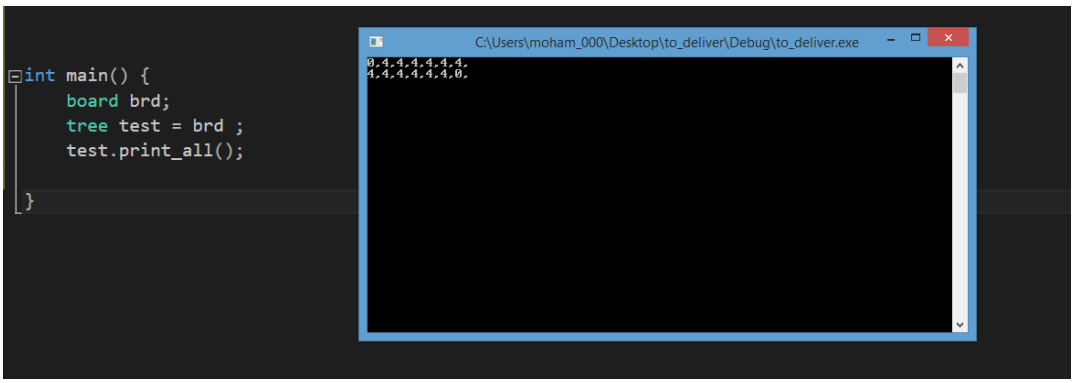
- A screenshot showing the code on the left and the expected output on the right.

2) The "tree" class

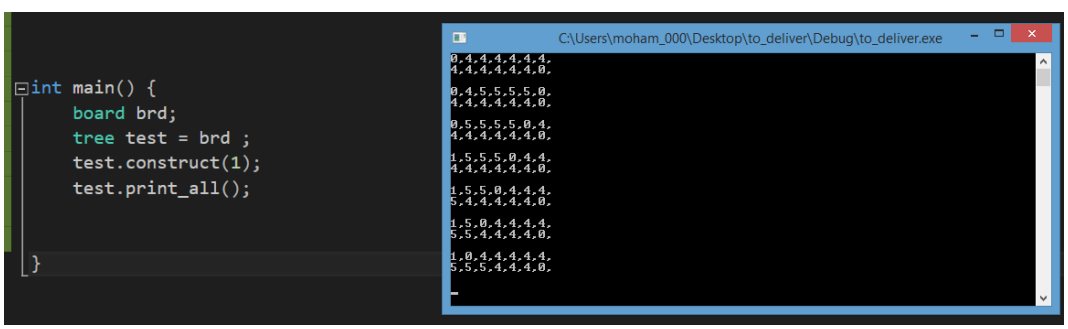
```
class tree {
private:
    node *tree_ptr, *temp_ptr; board BOARD_NOW;
    void print(node *start) { ... }
    void get_children(node *start, short player) { ... }
    void traverse(node *root, int level) { ... }
    void eval_leafs(node *start) { ... }
    void eval_driver() { ... }
    bool isleaf(node *start) { ... }
    short eval(node *nd) { ... }
    void minimax(node *start, bool ismaxlevel) { ... }
    void calc(node *start, bool ismax) { ... }
public:
    tree(board input) { ... }
    void print_all() { ... }
    void construct(int max_depth) { ... }
    void minimax_driver() { ... }
    short get_nextmove() { ... }
};
```

- The "tree" constructor takes as input a board parameter and sets the starting node with the values within the board.
- The "print_all" function is normally used for debugging to see all the contents of the tree.
- The "construct" function is used to construct the tree to a certain depth.
- The "minimax_driver" function applies the minimax algorithm on the tree.
- The "get_nextmove" function returns the nextbest move for the AI. This function is normally called after constructing the tree and running the :minimax_driver"
- Functions in the private section are used by the publicly accessed function in their inner workings.

- The “calc” function is used to calculate the max or min of children nodes given a parent node
- The “eval” function will evaluate the winning of a certain node given that node’s pointer.
- The “is_leaf” function returns a 1 if the node in question is a leaf of the tree.
- The “get_children” function is responsible for generating the possible children of a given parent node.
- The “print” function is given a starting root node and it recursively prints all the tree connected to that root.



- Screenshot showing the code on the left and the expected output on the right.
- The tree has only one node in it which is the starting node, and the board is printed.

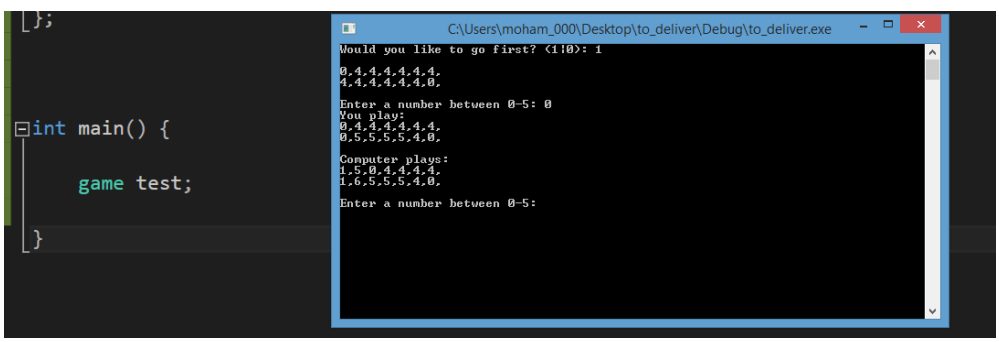


- Screenshot showing the code on the left and the expected output on the right.
- The tree has its root node shown first and 6 node with 6 possible moves for the AI to select from based on the “eval” function.

3) The “game” class

```
class game {
    bool turn; board BOARD_NOW; short MOVE; flag player_flags; short winner;
public:
    game() :MOVE(-1), player_flags({ 1,0 }) {
        cout << "Would you like to go first? (1|0): "; cin >> turn; cout << endl;
        tree AI_GAME_TREE = BOARD_NOW;
        if (!turn) {
            AI_GAME_TREE.construct(6);
            BOARD_NOW.play(1, AI_GAME_TREE.get_nextmove()); // AI moving
            cout << "Computer plays: " << endl;
            BOARD_NOW.print();
        }
        while (!BOARD_NOW.empty()) {
            while (player_flags.islegal != 1 || player_flags.isincup == 1) {
                BOARD_NOW.print();
                cout << "HERE" << endl;
                cout << "Enter a number between 0-5: "; cin >> MOVE;
                cout << "You play: " << endl;
                player_flags = BOARD_NOW.play(0, MOVE);
                BOARD_NOW.print();
                }
            AI_GAME_TREE = BOARD_NOW;
            AI_GAME_TREE.construct(6); BOARD_NOW.play(1, AI_GAME_TREE.get_nextmove());
            cout << "Computer plays: " << endl;
            BOARD_NOW.print();
            MOVE = -1;
            player_flags.islegal = 0;
            player_flags.isincup = 1;
        }
        winner = BOARD_NOW.calc_eval();
        if (winner > 0) {
            cout << "Computer wins!" << endl;
        }
        else if (winner == 0) {
            cout << "A DRAW! MY AI IS AS SMART AS YOU OR YOU ARE AS DUMB AS IT" << endl;
        }
```

- The game class has only a constructor and acts as a framework for the flow of the game.
- The class makes use of all the previous classes to drive a player vs. AI game.
- The implementation doesn’t consider that 2 players will be playing the game.



- Screenshot showing the code on the right and the output on the left.
- After entering a number between (0-5) on prompt the computer makes his move and this continues until the board is empty on either side, that is when the game ends and the game calculates who wins.

In python's version the state is represented as 4 lists, the computer and player sides represented as 2 lists each has 6 elements represents the number of rocks in each one, the other 2 has the value for player's cup and another for computer's cup.

The utility function is getting the move that maximizes the number of rocks in the computer's cup, and in players' turn, it is the move that minimizes the rocks in the computer's cup.

It has 2 modules.

1) ai module

Which contains 5 functions

-comp_move and player_move which takes an integer and a state and returns the new state and a pool to determine if the player should play again or not if the last rock ended in his cup

-best_comp_move which gets the best move for the computer which maximize his cup

-best_player_move which gets the best move for the player which minimize the computer's cup

-get_best_move: which gets called in computer's turn and takes an integer from 1 to 3 which represents the difficulty and the depth of the tree, the searching is done breadth-first with max depth 3

2) main module

-Which has the main game loop

-save and load functions to save current game state

-is_game_over which takes a state and returns a pool to determine if the game ends, and the final game state

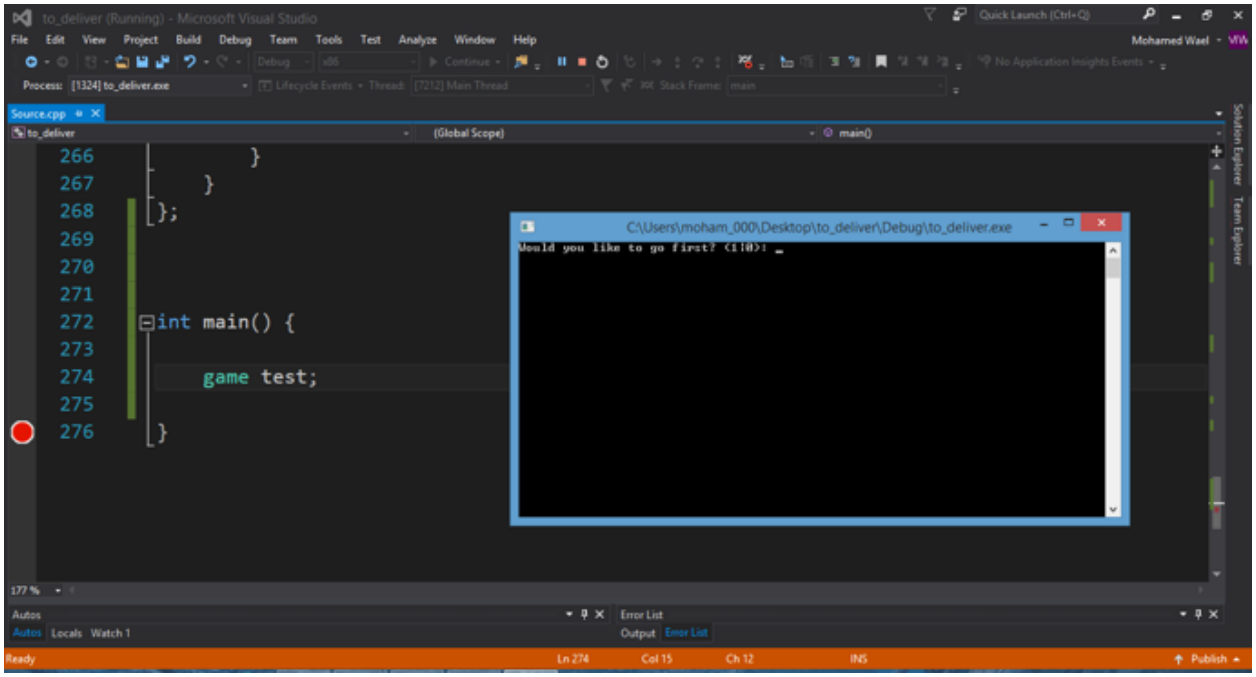
The game is over if the sum of ether side is zero, and in this case, the sum of the other size is added to the belonging cup

03

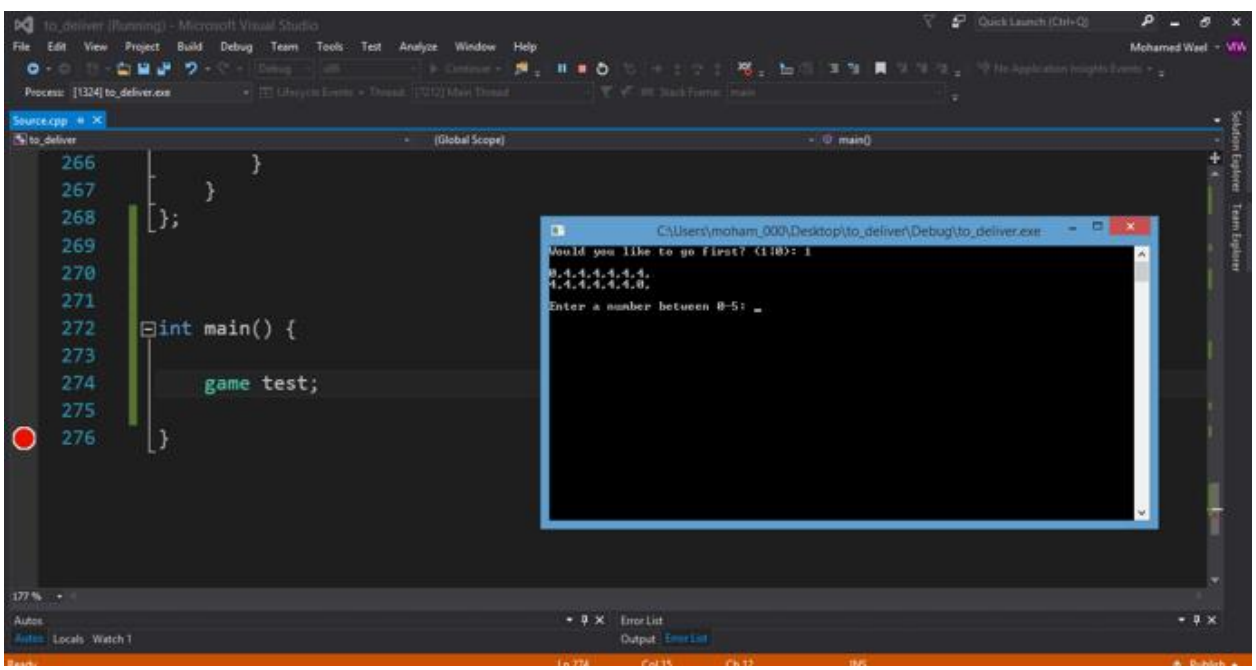
Third Topic

User Guide

C++ version

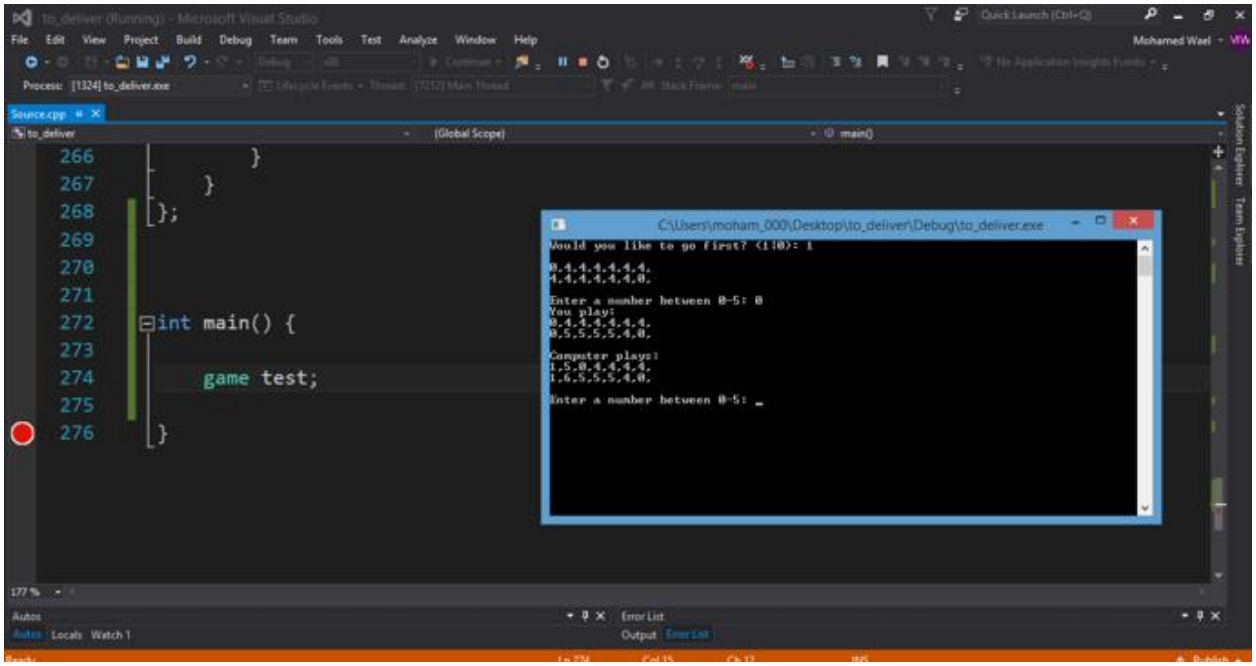


- Game prompts to see if you want to go first.
- Typing (1) for yes , (0) for no and pressing enter

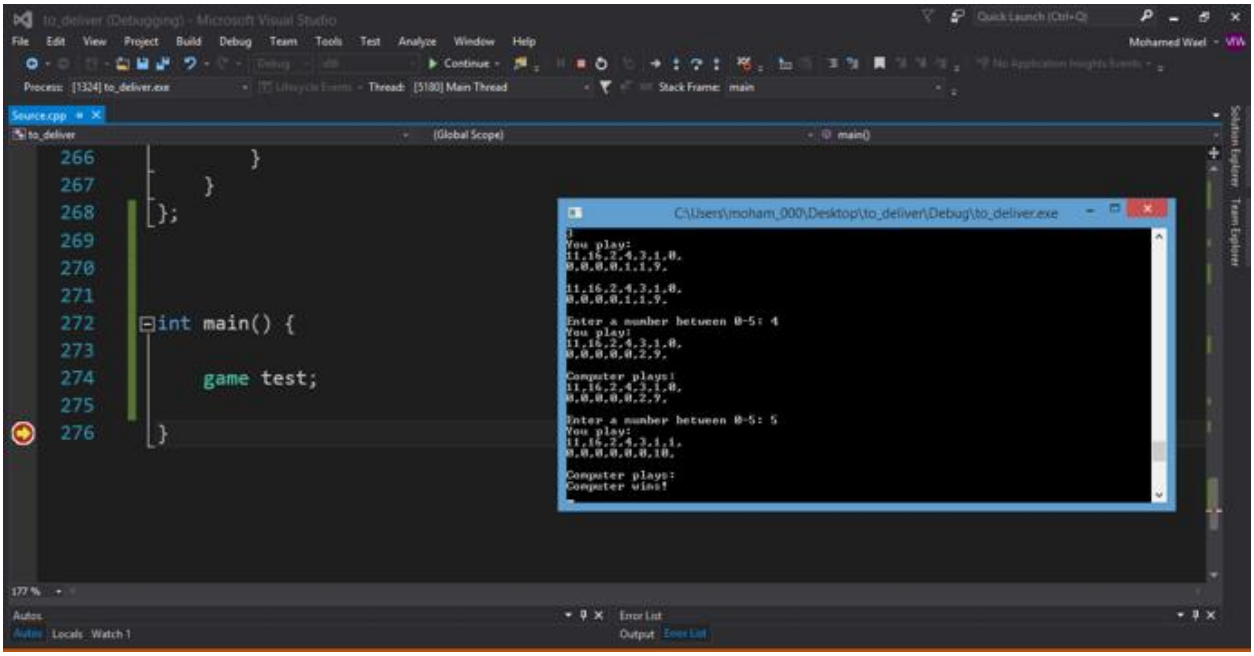


- In this case I typed 1.

- The game shows the board at the current state and prompts me to enter a number between 0-5 to move my pieces.

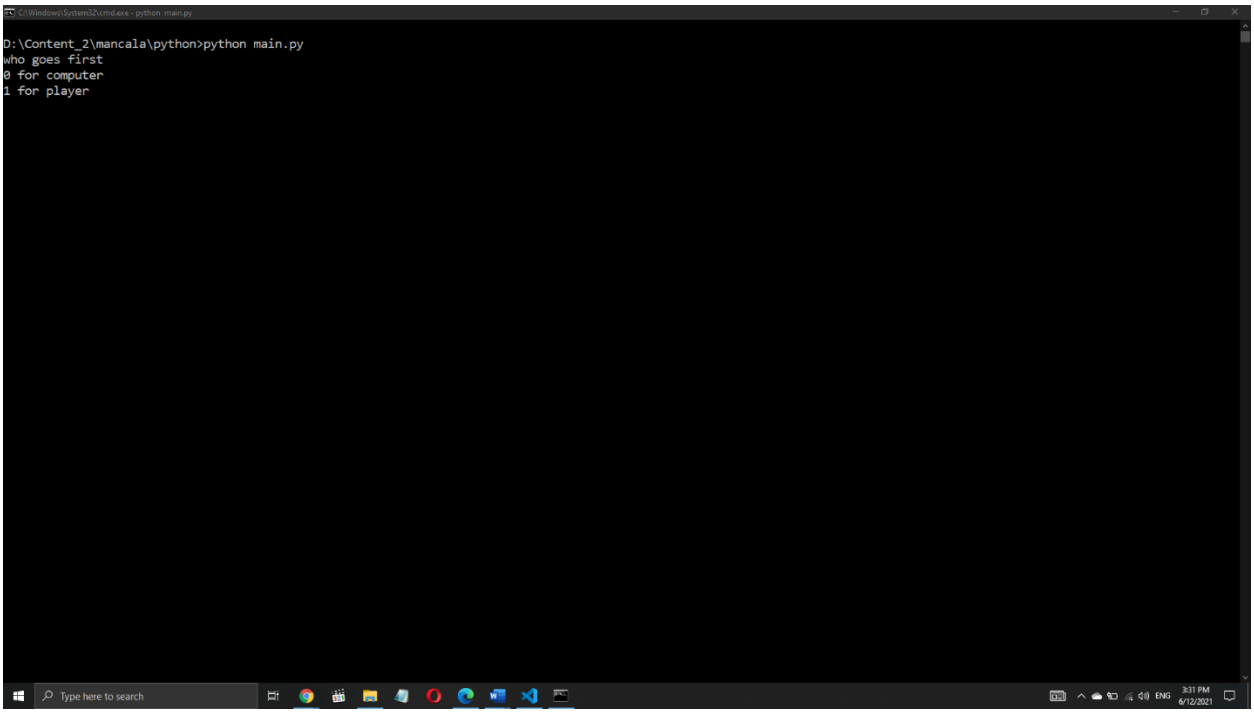


- In this case I move the pieces in position 0
- The game shows your move and then the computer's move and prompts you to play another move.

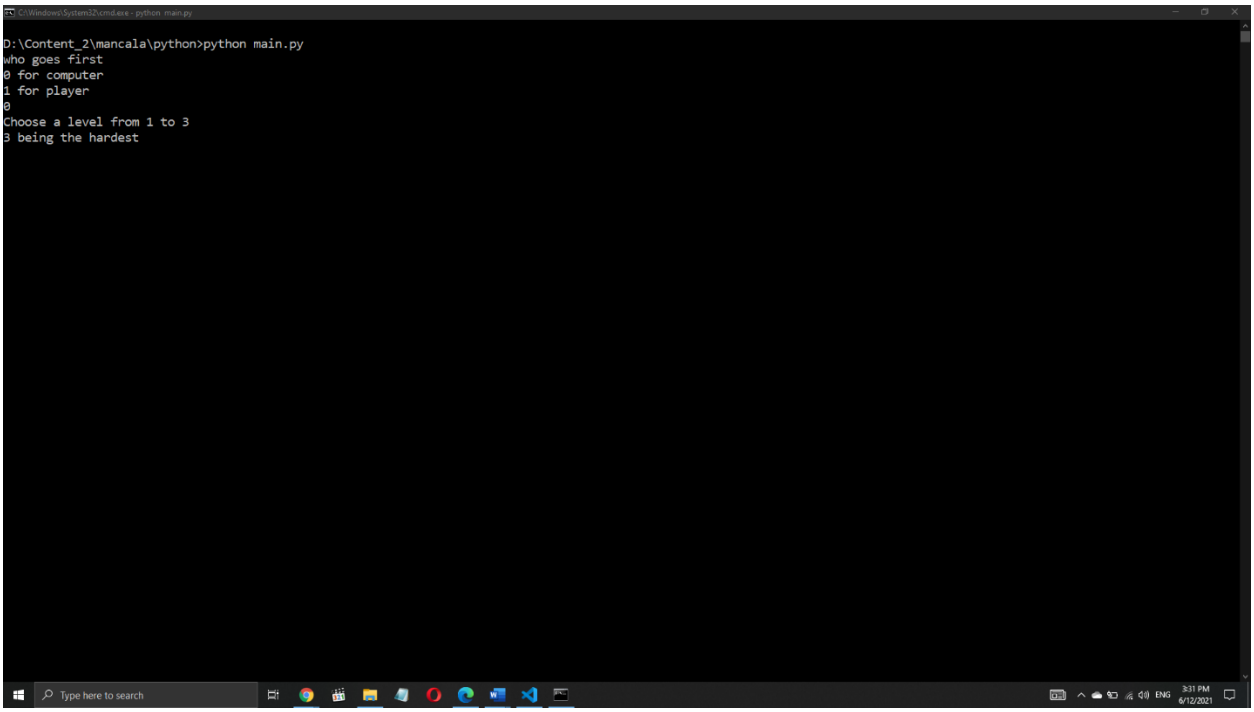


- After a long game, the computer finally won.

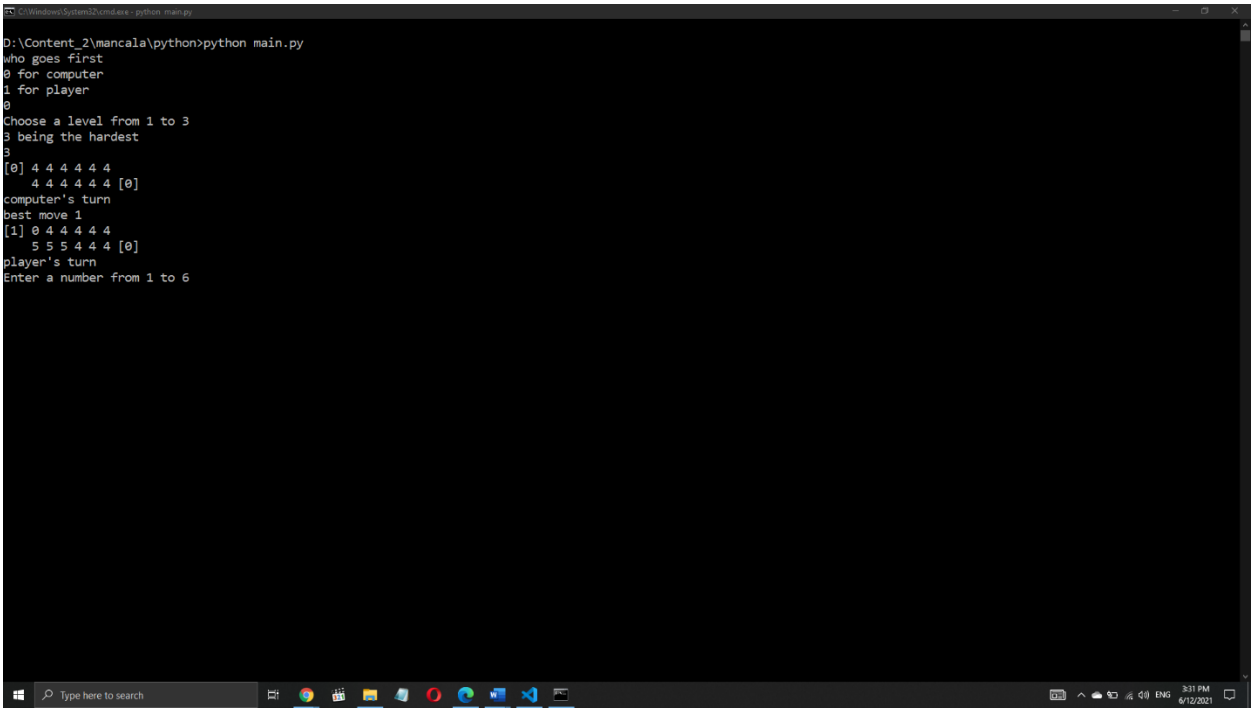
Python version



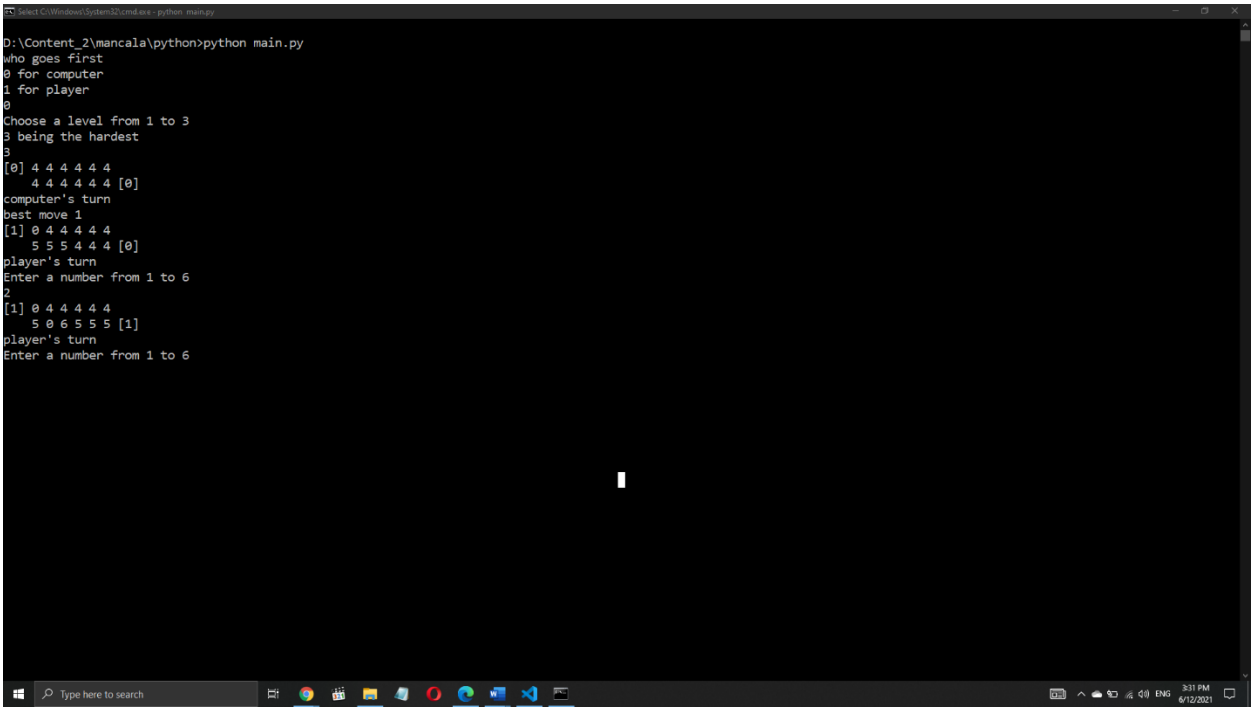
- The program prompt to see which goes first



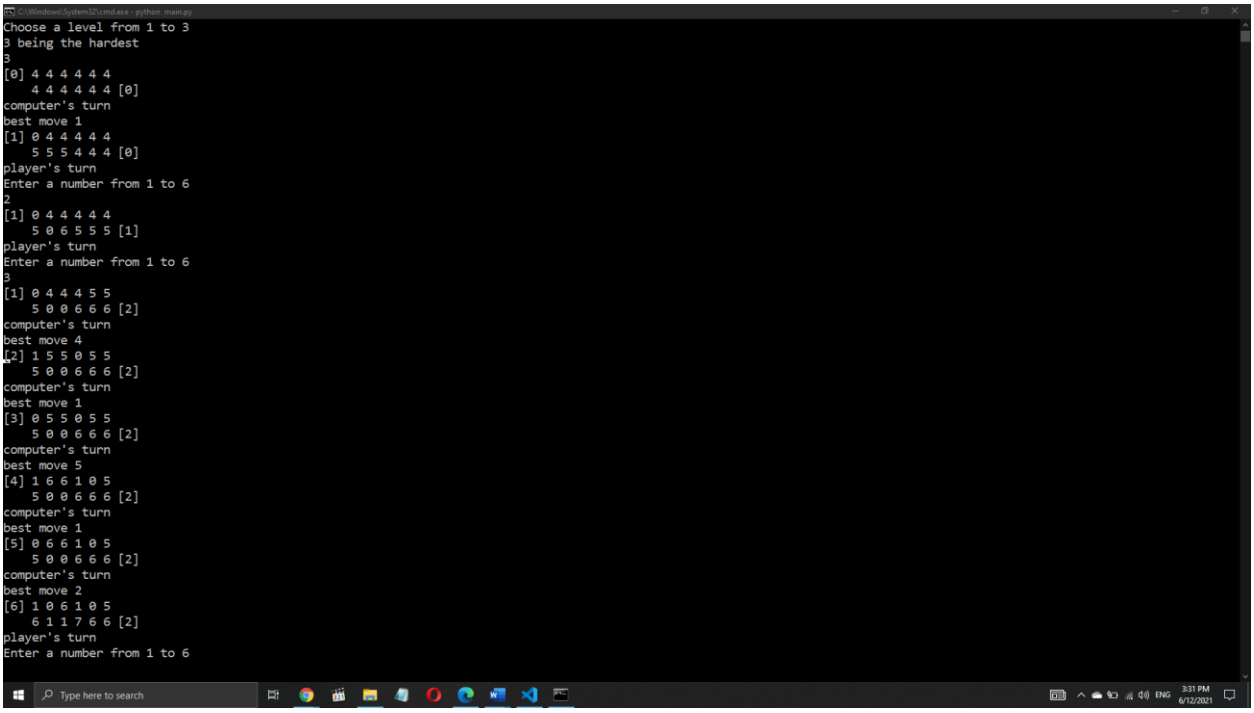
- The user chooses a level from 1 to 3



- The computer starts and the player is asked to enter a number from 1 to 6 to choose from his side



- The player can play again because the last rock ended up in his cup



```
Choose a level from 1 to 3
3 being the hardest
3
[0] 4 4 4 4 4 4
    4 4 4 4 4 4 [0]
computer's turn
best move 1
[1] 0 4 4 4 4 4
    5 5 5 4 4 4 [0]
player's turn
Enter a number from 1 to 6
2
[1] 0 4 4 4 4 4
    5 0 6 5 5 5 [1]
player's turn
Enter a number from 1 to 6
3
[1] 0 4 4 4 5 5
    5 0 0 6 6 6 [2]
computer's turn
best move 4
[2] 1 5 5 0 5 5
    5 0 0 6 6 6 [2]
computer's turn
best move 1
[3] 0 5 5 0 5 5
    5 0 0 6 6 6 [2]
computer's turn
best move 5
[4] 1 6 6 1 0 5
    5 0 0 6 6 6 [2]
computer's turn
best move 1
[5] 0 6 6 1 0 5
    5 0 0 6 6 6 [2]
computer's turn
best move 2
[6] 1 0 6 1 0 5
    6 1 1 7 6 6 [2]
player's turn
Enter a number from 1 to 6
```

- The computer determines his best moves to maximize his cup

04

Fourth Topic

Work distribution and experience

Work load and distribution

Mohamed Wael Mohamed	Board class
Mahmoud Othman rabea	Tree class
Ahmed Mohamed Ibrahim	Tree class
Mohamed Medhat Mohamed	Ai module python
Omar abdelshafy	Game class
Mahmoud ahmed	Main module python

Project experience

- It needed more time to flush out all the bugs.
- The minimax tree needed a variation since players who got their pieces in the cup got another turn. This detail was lost on us at first and caused a very tiring rewriting of the code base and logic.
- Overall was fun to work on, but truly needed more time to refine it.

python youtube video: https://youtu.be/4e4Slr1FbcM

C++ youtube video: https://youtu.be/lgOLku4TfZY

Project repo link: https://github.com/mahraee/MancalaGame.git