AI Project Report

Team Members:	IDs:
Peter Isaac Saad Iskander	1600407
Karim Salah Sadek Mahmoud	1601003
Mira Yohanna Maxwell	1601518
Thomas Joseph Kamil Youssef	1600440
Beshoy Alber Fahmy Abdel-Malek	1600414

Implementation:

We implemented a terminal-based interaction of the mancala.

The start scene:

```
 Start game:
    1 - Player vs Player
    2 - Player vs AI
    3 - AI vs AI
 L - Load Game
 O - Options
 E - Exit
```

You can load a game, edit the game options or exit, here is what the options menu looks like:

```
Difficulty Level:  Next step in the evolution (1 turn may take over a minute)
Stealing:  On
Verbose mode:  Off
D - Change Difficulty
S - Switch Stealing
V - Switch Verbose Mode
E - Back to main menu
```

Changing the difficulty :

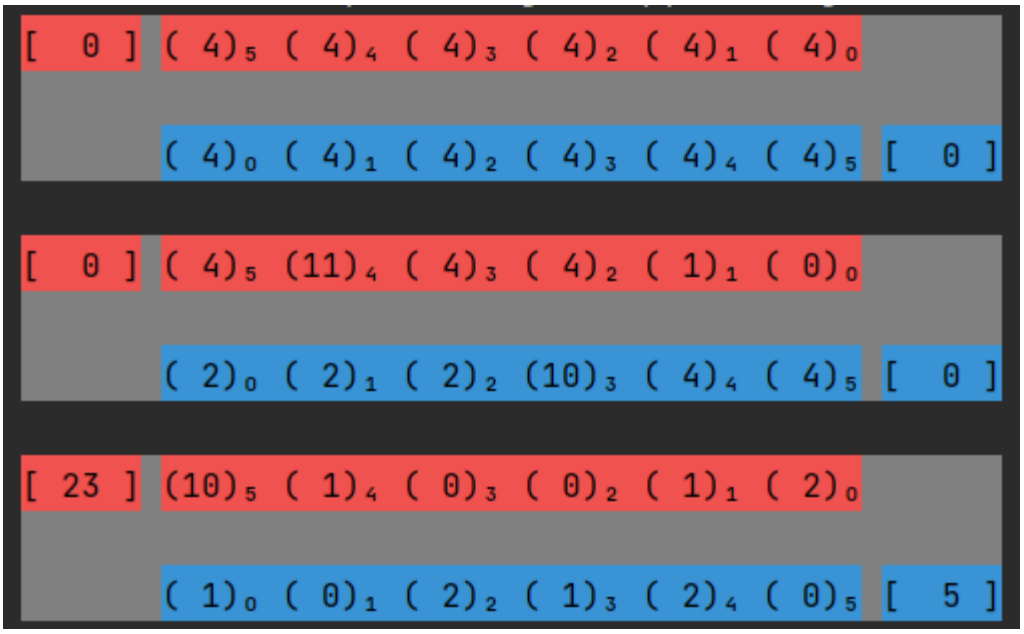
```
1 - Not even playing
2 - Stupid
3 - Very Easy
4 - Easy
5 - Normal
6 - Above Average
7 - Hard
8 - Very Hard
9 - Smarter than most
10 - Next step in the evolution (1 turn may take over a minute)
```

Game start:

```
Choose who starts!
1 - Me
2 - AI
```

Game GUI:

examples:



The GUI is terminal based and the blue slots indicate the player's slots and the red ones indicate the opponent's.

Each slot has two numbers: $(x)_y$, the x represents the number of beads in one slot and the y represents the number of the slot which the player can choose to play.

Players take turns where each player inputs the number of the slot he wants to move the beads from until one side is empty and the game ends, the player having the most number of beans in their mancala + beads on their side wins.

Bonus features:

1) Verbose mode:

   The game supports verbose mode which can be turned on or off , in verbose mode , AI information is shown briefly in console window and written in more details in a txt file in verbose directory

2) Different difficulties:

   Our game supports multiple levels of difficulties starting from **1 to 10** as follow("Not even playing", "Stupid", "Very Easy", "Easy", "Normal", "Above Average", "Hard", "Very Hard", "Smarter than most" and "Next step in the evolution (1 turn may take over a minute) ")

3) Networking mode:

   It supports **AI vs AI mode** on the same device as well as **player vs player mode**. It is implemented using client and server technique. Furthermore, The **TCP/IP** socket can support playing on different machines from different networks if port forward is used in router configuration.

4) Loading & saving of the game:

   The game offers saving and loading option, where saving is to store the latest state of the game including the difficulty level, modes activated and number of turns played in a file using **pickle.dump()**, while loading is to read all these information from the file to be able complete the game starting from the last saved move.

Detailed functions description:

eval_func:

It is the score evaluation function that takes as parameters the current state and the state where I was to calculate the score, it tries to do two things; maximize your score and minimize your opponent's.

A state's score depends on two things; the number of beads in your mancala and the number of beads in the row in front of you, the combination of both is dependent on a factor called mankla_to_front_fact and the combination of how much you want to maximize your score and minimize your opponent's is dependent on a factor called lamda, various difficulties changes these factors.

TreeNode:

Class TreeNode is the building block of our tree, it contains important information like whether the node is a maximizer or not, or whether a cutoff started at this node or not ..etc.

generate_search_tree:

This recursive function takes the current state and builds a tree which contains all the possible state changes up to a specified max_depth, it assigns utility values to all leaf nodes using eval_func and returns its root in the end.

alpha_beta:

Alpha-beta optimization function , take as input the root tree node, each node is initialized with an alpha of a very high negative value and a beta of a very high positive value, we first loop over that node's children , if a child is a leaf check if that child is a maximizer or a minimizer and update the node's alpha or beta correspondingly also check if the node's alpha is bigger or equal to it's beta if so indicate a cut off and prune accordingly.

If the child is not a leaf, pass down the alpha and the beta of the node to it and recurse the function on the child.

UI:

This function represents the interface between the user and the game code itself. It prints in the terminal the board in an illustrative way, where player 1 is colored in blue, player 2 is colored in red, the background of the board is colored in grey to determine its edges, and each pocket is determined by its index subscripted beside it. The board's shape is consistent for clarity. The function's only parameter is the state vector *state_vec*, and according to it, it prints the board in the terminal with corrects pockets and mancalas values.

take_action:

This function is the implementation of the game itself and how it is played.

Parameters:

current_state_vec is a 14-element list that represent the 6 pockets and mancala of each player making them 14 elements, starting indexing from pocket 0 at player 1 (blue side) and ending at the mancala of player 2 (red side) in a counter-clockwise direction.

is_stealing is a boolean variable that if 0 it indicates that stealing mode is off, and if 1 then stealing mode is on.

action is the index of the pocket from which the player selects to play on his side, this indexing follows the subscripting as shown in the terminal UI.

player_side is a Boolean variable that if 0 indicates the turn of player one (blue), and if 1 indicates the turn of player 2 (red).

Return values:

current_state_vec is returned after it is manipulated according to the current turn and action.

next_turn is a boolean variable that indicates the new player's turn after modifications occurred to **current_state_vec**. If 0, it is the turn of player 1 (blue). If 1 it is the turn of player 2 (red).

reset_state:

This function initializes the game board to its first state, and it takes no arguments.

take_next_input:

This function is responsible for handling the input of the next turn in case user enters an invalid input.

traverse:

This function takes the generated alpha-beta optimized tree as input and traverses through it depth-first style until it visits all the nodes, it extracts from the nodes all the information needed for verbose mode.

print_verbose:

This function is responsible for providing information about leaf node values, cut off nodes, maximum depth reached. It prints a brief description about these information in the console and then, it saves the full description in verbose directory in a text note. It takes as arguments: verbose mode, tree, time difference between different moves, number of turns played and file name.

Team Members Contribution:

Peter Isaac Saad Iskander:

- Networking mode:
  - AI vs AI
  - Player vs Player
  - TCP/IP socket initialization (server side)
- Verbose Mode
- Saving the game parameters needed to initiate the game
- Project Video

Karim Salah Sadek Mahmoud:

- eval_func
- TreeNode
- generate_search_tree
- First working implementation of Player vs AI

Mira Yohanna Maxwell:

- Networking mode:
  - AI vs AI
  - Player vs Player
  - TCP/IP socket initialization (client side)
- Verbose mode
- Handling loading game mode and setting the loaded parameters

Thomas Joseph Kamil Youssef:

- eval_func
- TreeNode
- alpha_beta

Beshoy Alber Fahmy Abdel-Malek:

- take_action
- UI