Logo

Description automatically generated

**AI project report**

**Team members:**

Peter Isaac Saad Iscander 1600407

Karim Salah Sadek Mahmoud 1601003

Mira Youhana Maxwell 1601518

Thomas Joseph Kamil Youssef 1600440

Bishoy Alber

Our implementation:

We implemented a terminal-based interaction of the mancala.

The start scene:

Text

Description automatically generated

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

Text

Description automatically generated

Changing the difficulty :

Text

Description automatically generated

Game start:

Text

Description automatically generated

Game GUI:

A picture containing text, orange

Description automatically generated

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 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
2. Different difficulties
3. Networking mode
4. Loading & saving of the game

**Detailed functions description:**

**eval\_func:**

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.

**Team members contribution:**

Peter Isaac Saad Iskander:

Karim Salah Sadek Mahmoud:

* eval\_func
* TreeNode
* generate\_search\_tree
* First working implementation of Player vs AI

Mira Youhana Maxwell:

Thomas Joseph Kamil Youssef:

* eval\_func
* TreeNode
* alpha\_beta

Bishoy Alber: