Name: Nada Taher Elwazane ID: 6876

Name: Mariem Mostafa Mahmoud ID: 6873

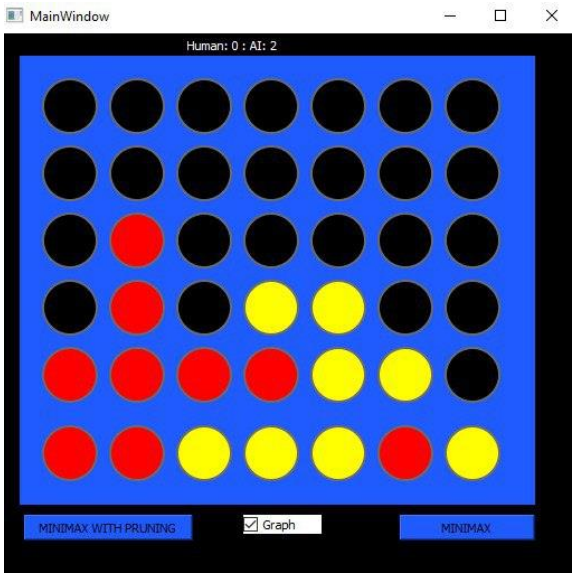
Name: Haguar Tarek Dessouky ID: 6878

Name: Manar Amgad ID: 7113

Artificial Intelligence

Project 2

Connect 4 AI Agent



Connect Four is a two-player strategy game played on a 7-column by 6-row board. Each player has a color and drops successively a disc of his color in one column, the disc falls down to the lowest empty cell of the column. When a player makes an alignment of four discs of his color (the alignment can be either vertical, horizontal, on the left diagonal or on the right diagonal), his score is incremented by 1, and at the end of the game, the player who made the highest number of alignments is the winner. If the board is filled without alignment it's a draw game.

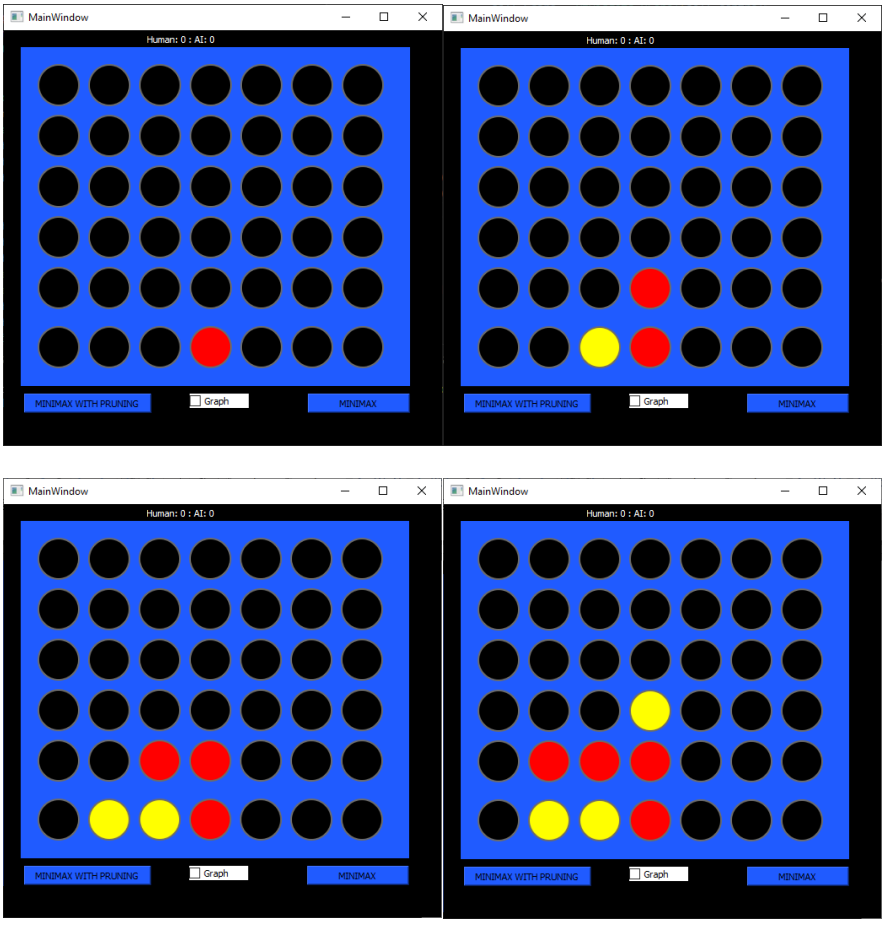
The main purpose of the code is to develop a nearly perfect playing Connect-4 agent. We started by the **minmax** algorithm which was efficient to a certain level and then, we tried to optimize it with the **alpha-beta** pruning algorithm.

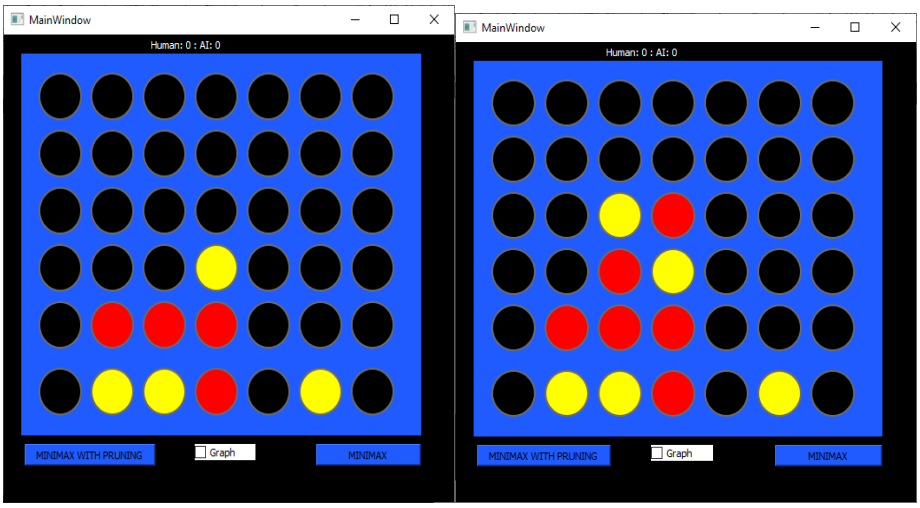
We give the option to the human player to either chose the **minmax** algorithm or the **alpha-beta pruning** algorithm in the beginning in the GUI and the efficiency of each algorithm will be already detected when playing especially by measuring the time taken for the agent to make a move. The score of each player is updated on the top and there is an option to print the tree of the algorithm chose at each move illustrating the best move coming.

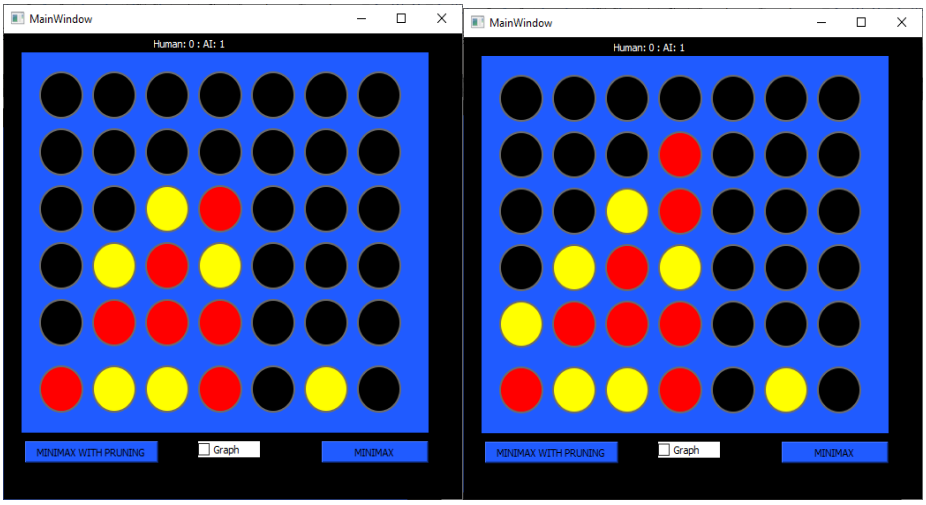
Sample Runs:

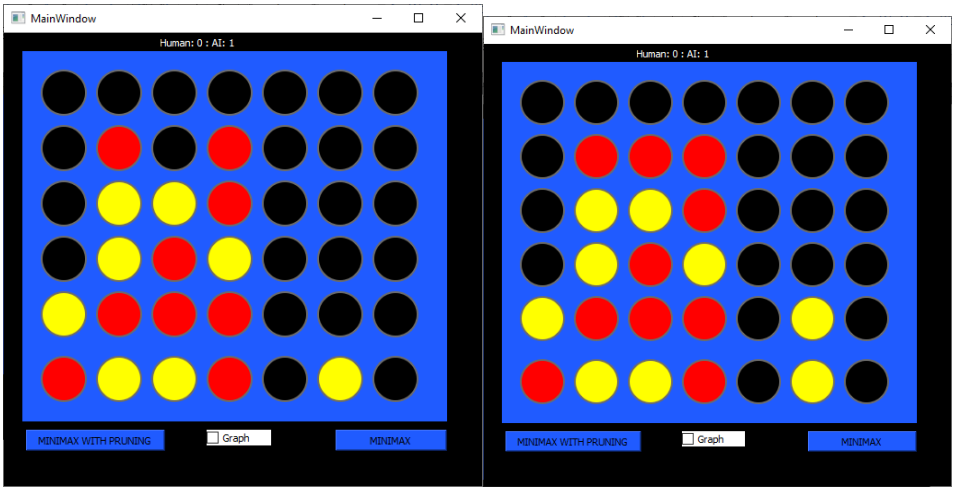
Minmax Algorithm:

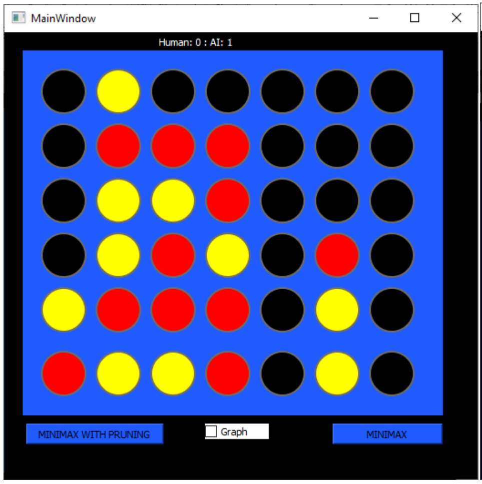
Depth=6

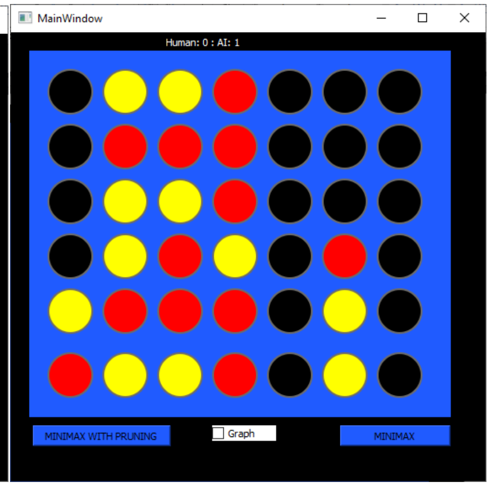


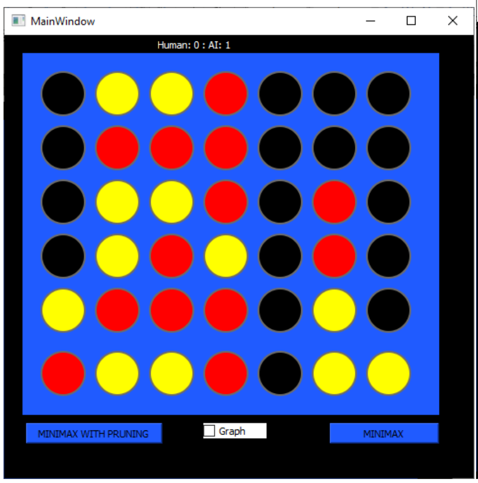

MainWindow
Human: 0 : AI: 0
MINIMAX WITH PRUNING
Graph
MINIMAX
MainWindow
Human: 0 : AI: 0
MINIMAX WITH PRUNING
Graph
MINIMAX

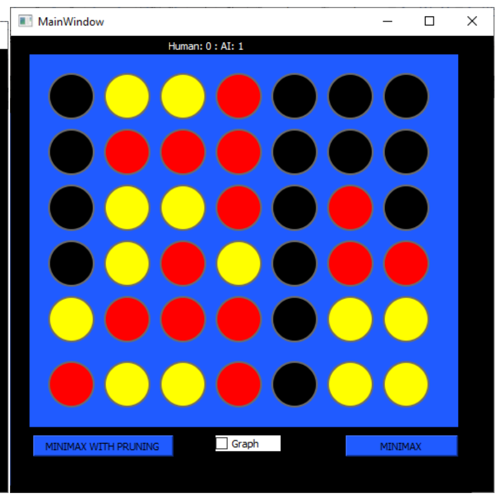

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX
MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

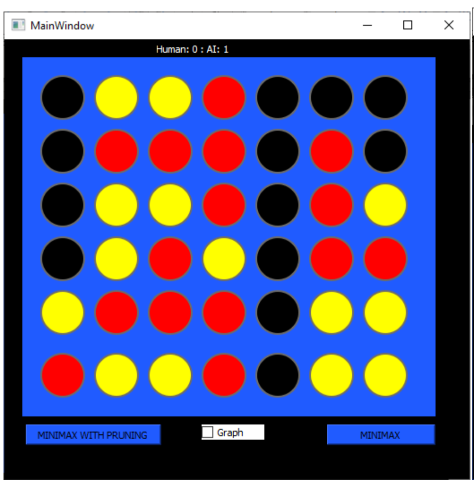

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX
MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

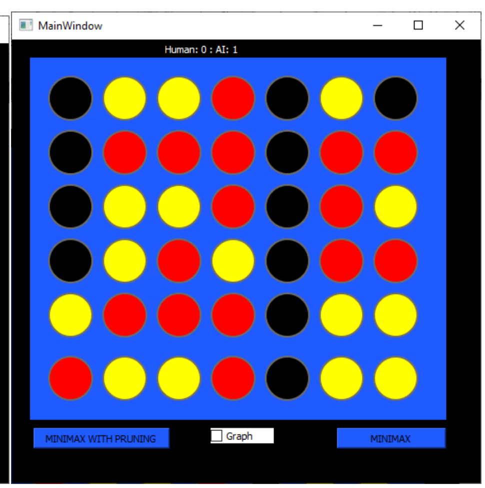

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

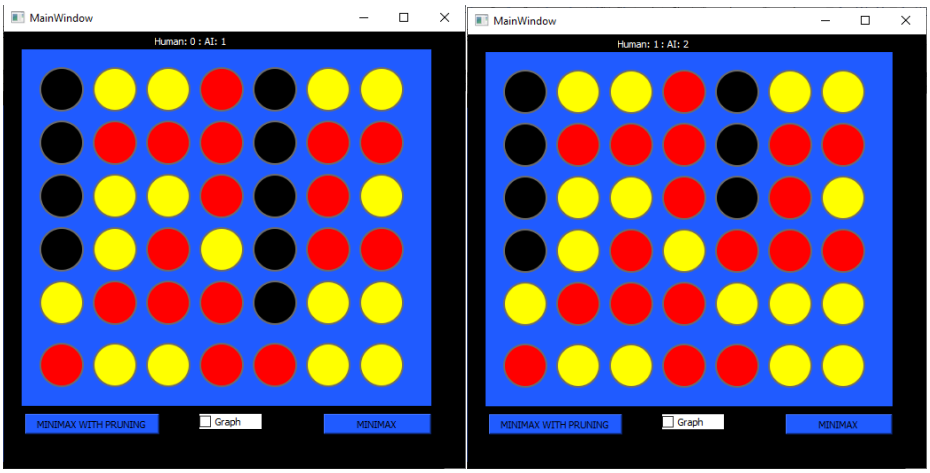

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

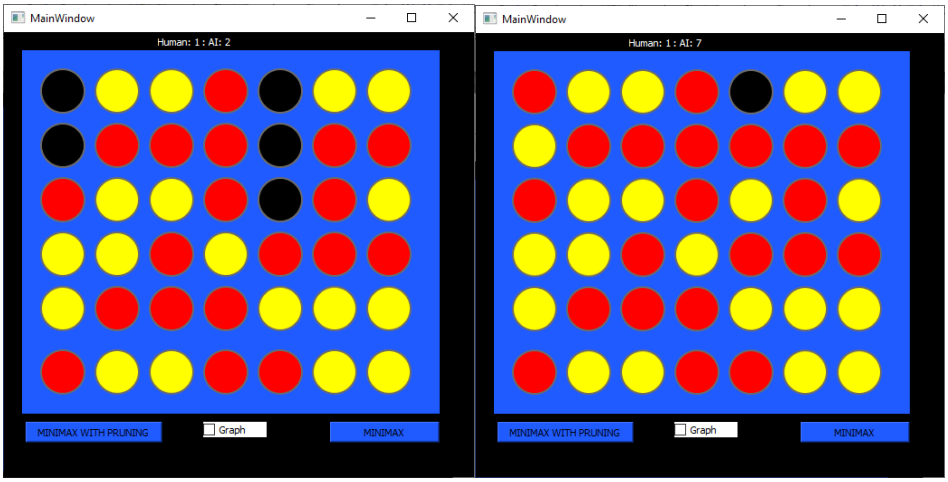

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

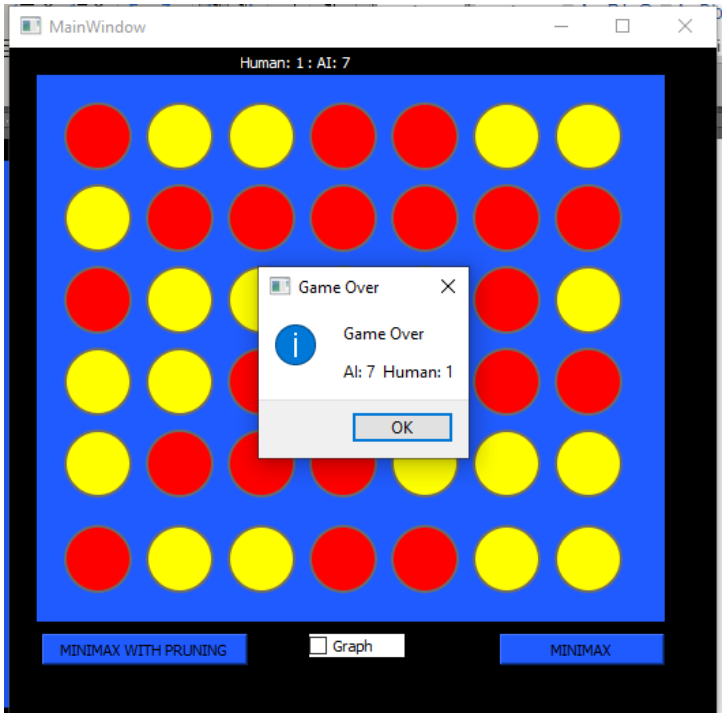

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX


MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

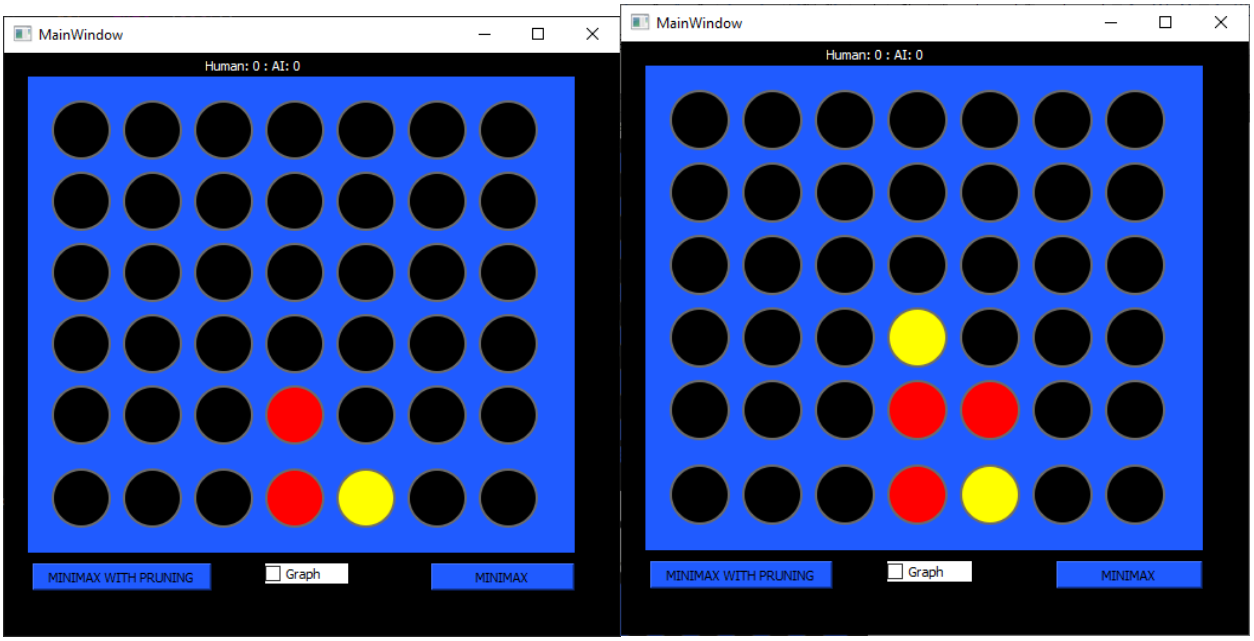

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

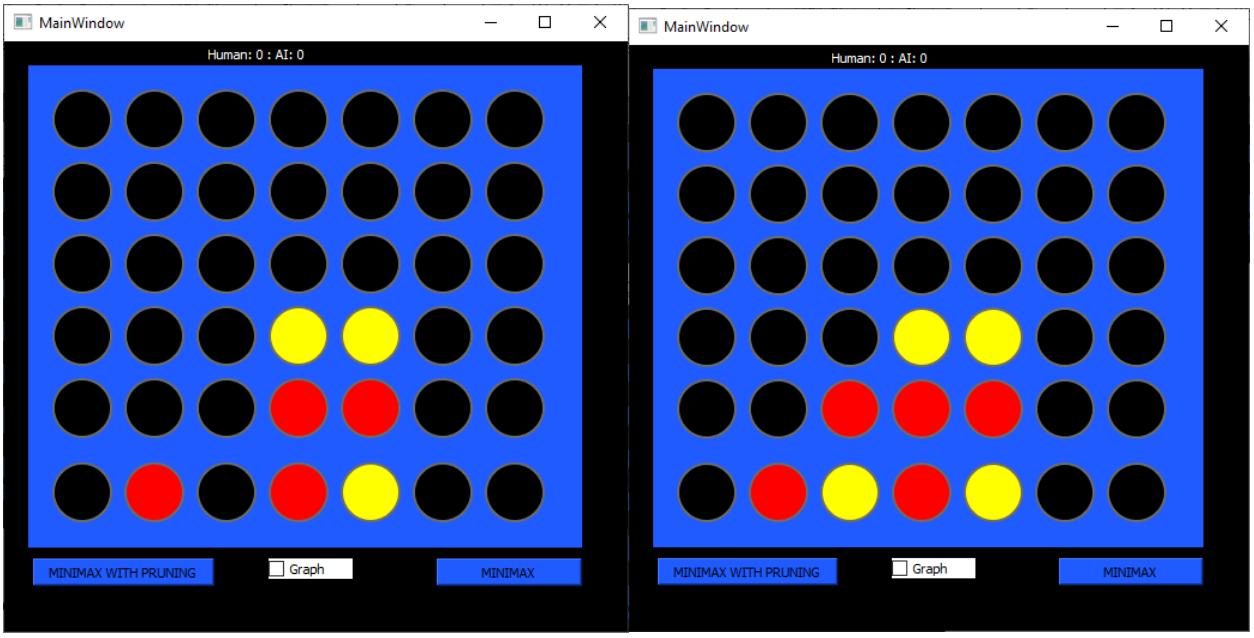

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX
MainWindow
Human: 1 : AI: 2
MINIMAX WITH PRUNING
Graph
MINIMAX

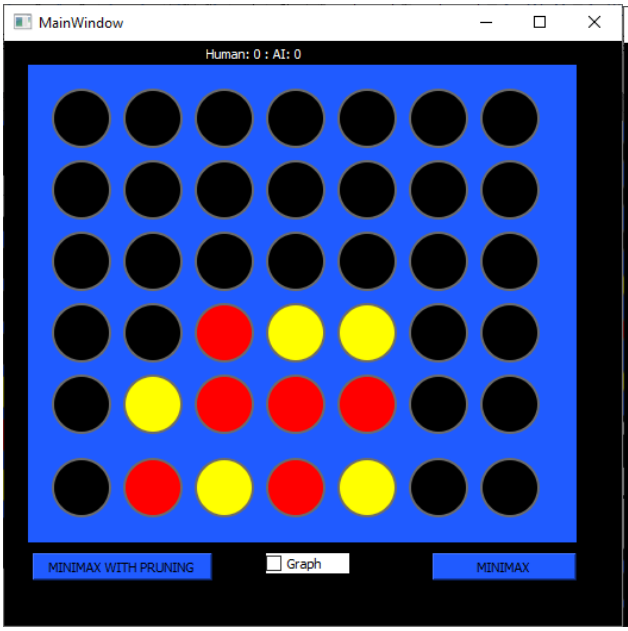

MainWindow
Human: 1 : AI: 2
MINIMAX WITH PRUNING
Graph
MINIMAX
MainWindow
Human: 1 : AI: 7
MINIMAX WITH PRUNING
Graph
MINIMAX

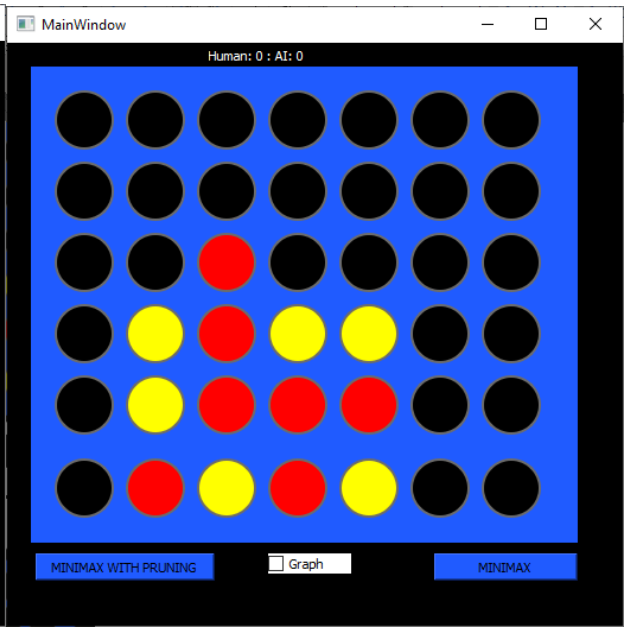

MainWindow
Human: 1 : AI: 7
Game Over
Game Over
AI: 7 Human: 1
OK
MINIMAX WITH PRUNING
Graph
MINIMAX

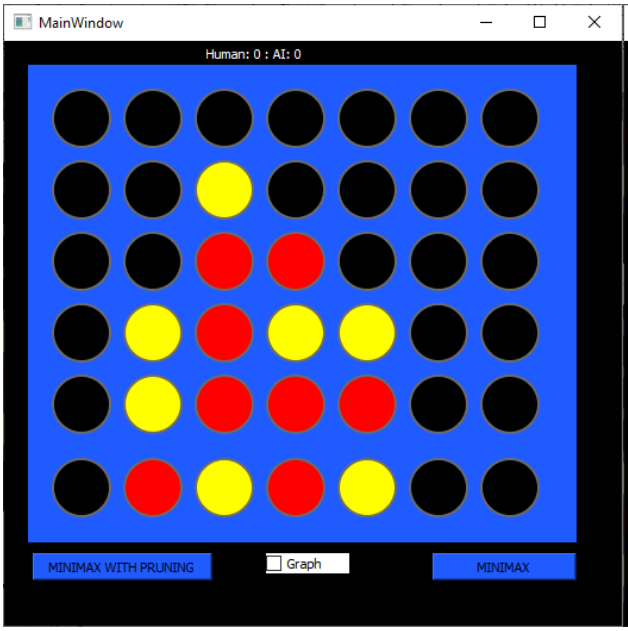
Minimax with alpha beta pruning:

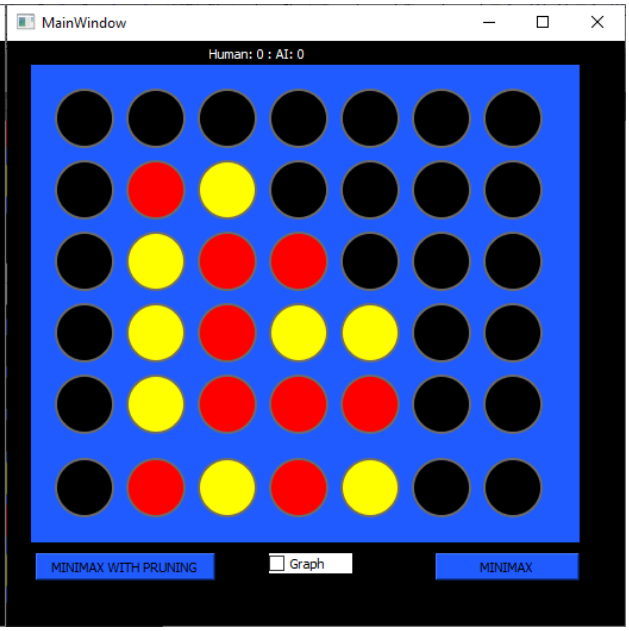


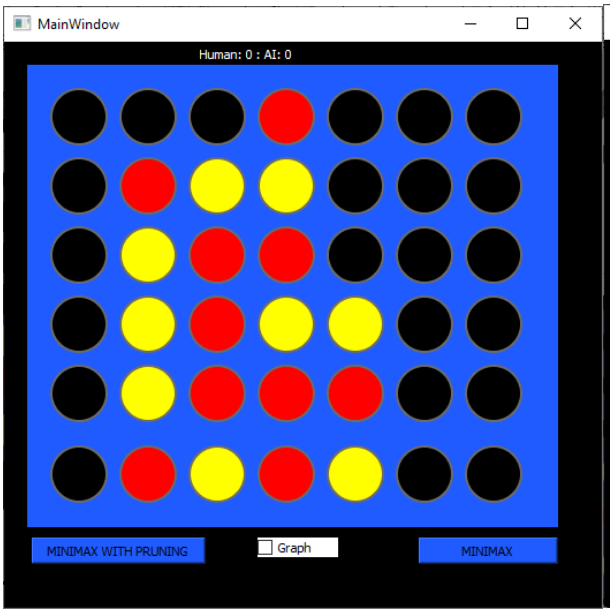


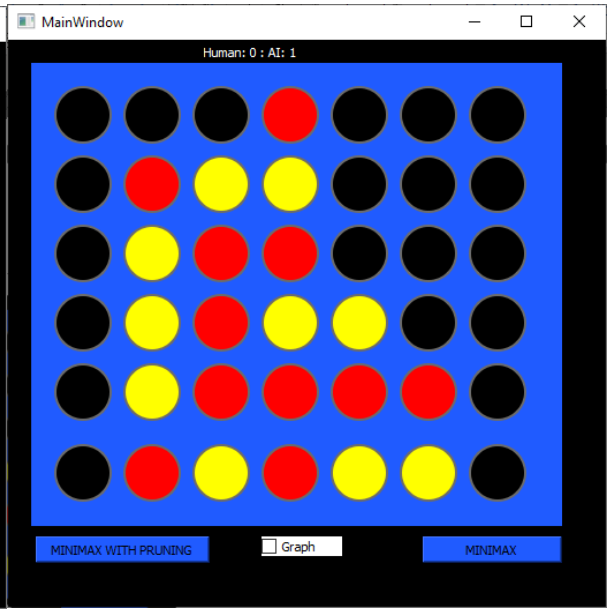

MainWindow
Human: 0 : AI: 0
MINIMAX WITH PRUNING
Graph
MINIMAX

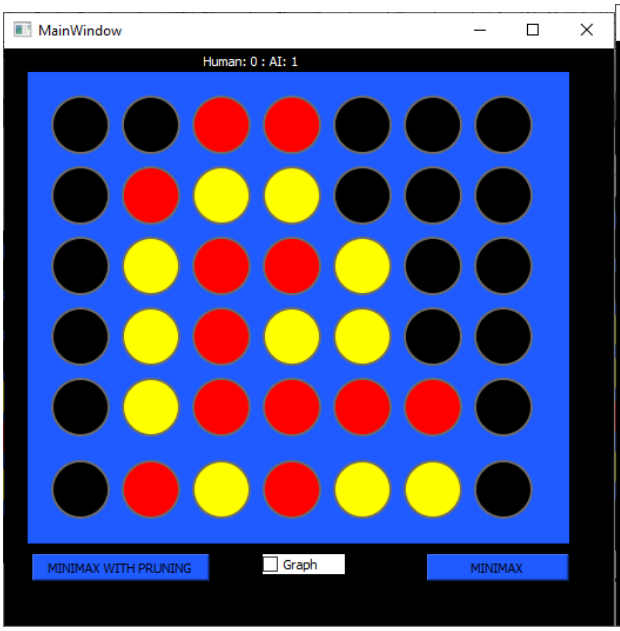

MainWindow
Human: 0 : AI: 0
MINIMAX WITH PRUNING
Graph
MINIMAX

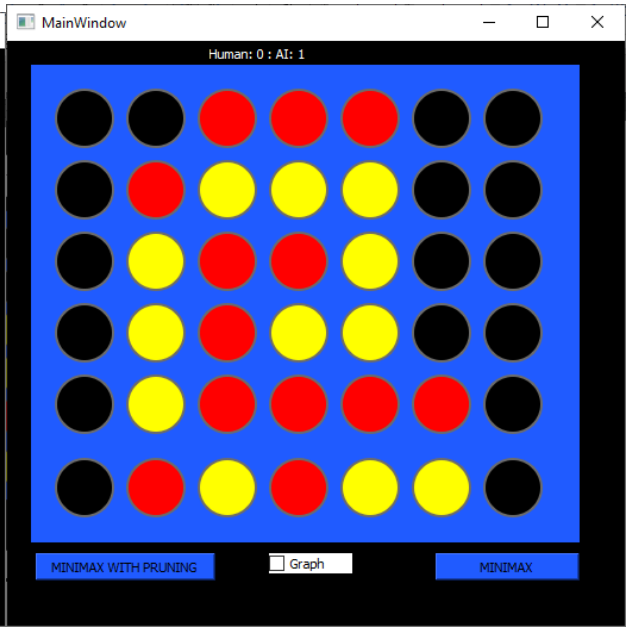

MainWindow
Human: 0 : AI: 0
MINIMAX WITH PRUNING
Graph
MINIMAX

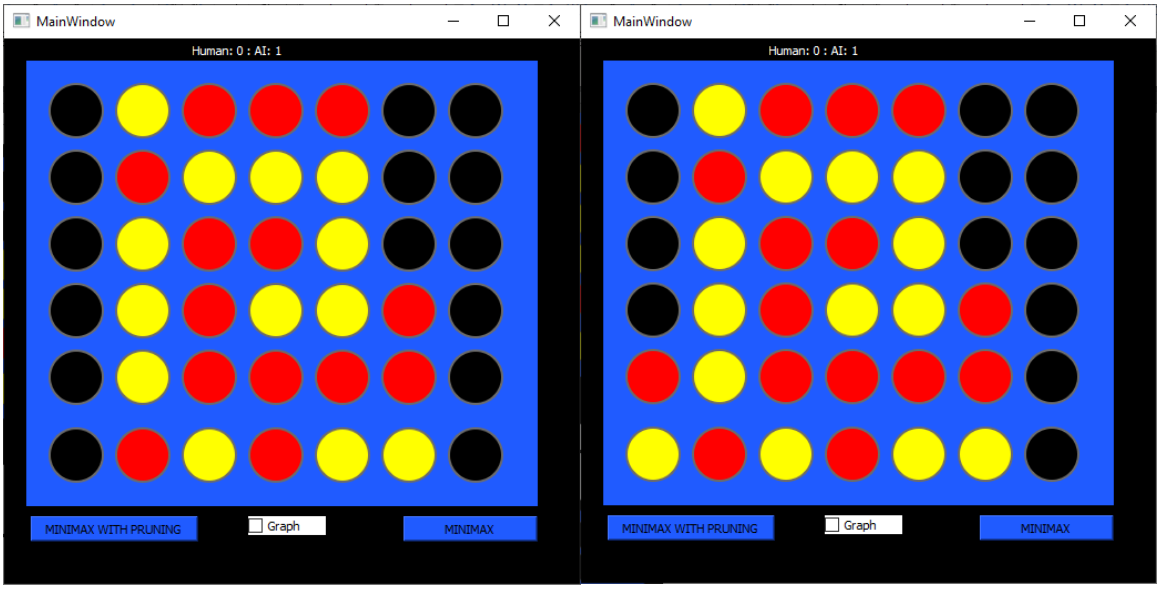

MainWindow
Human: 0 : AI: 0
MINIMAX WITH PRUNING
Graph
MINIMAX

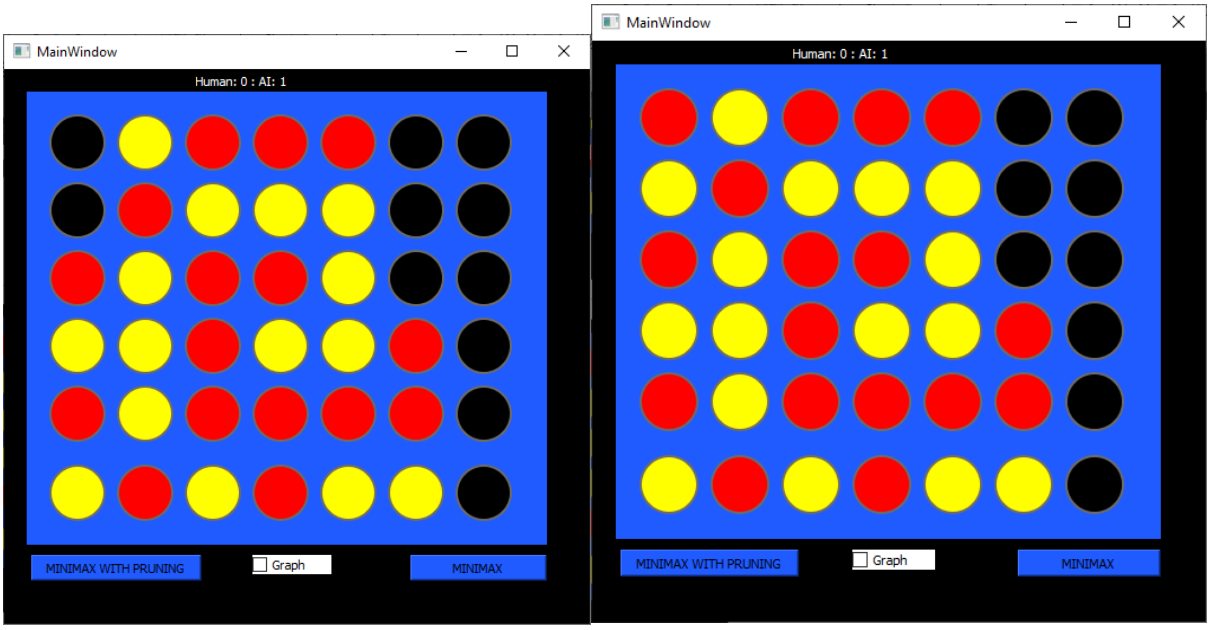

MainWindow
Human: 0 : AI: 0
MINIMAX WITH PRUNING
Graph
MINIMAX

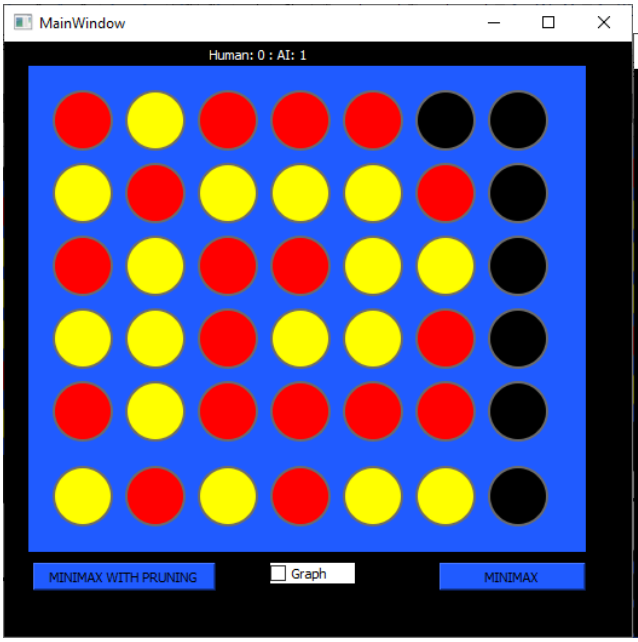

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

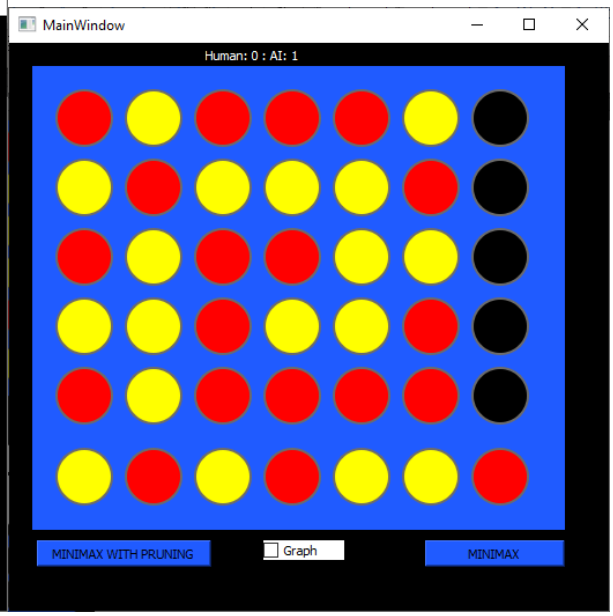

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

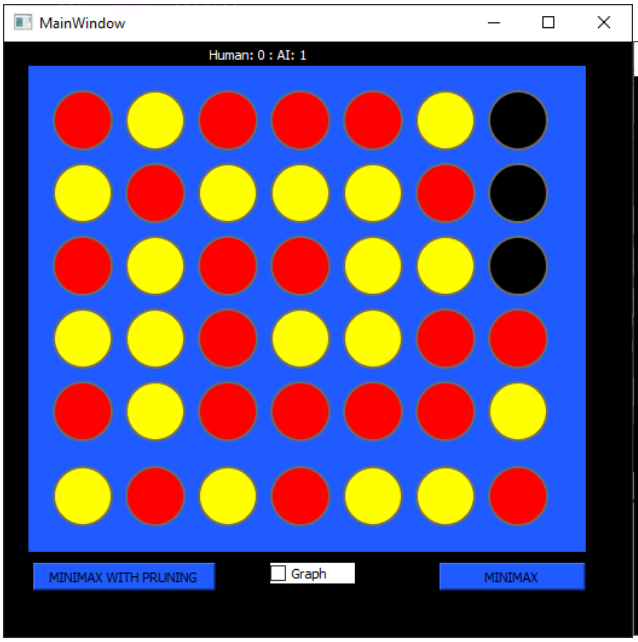

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

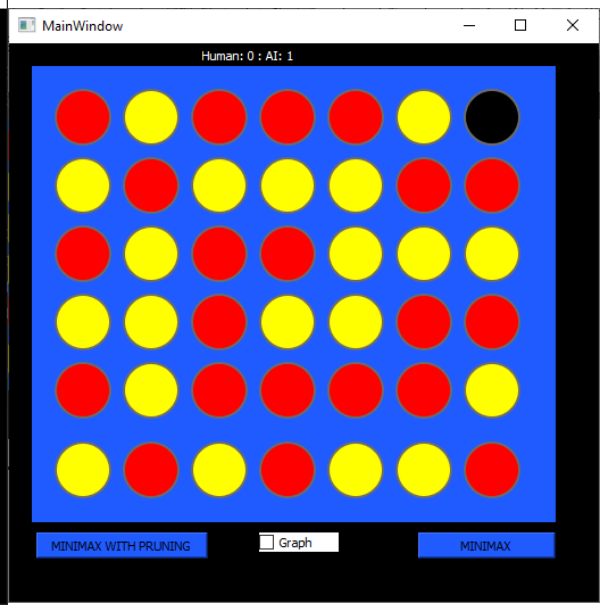

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX
MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

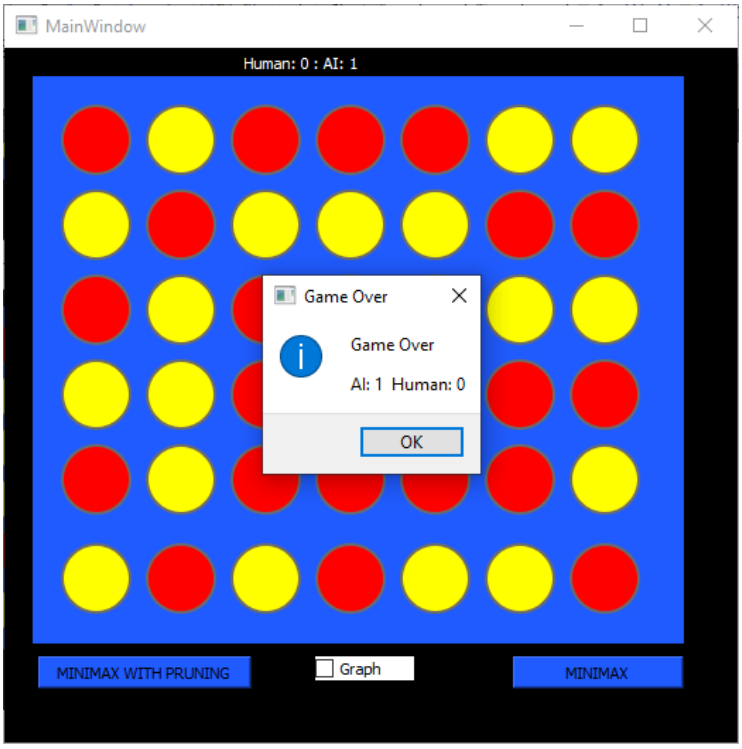

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX
MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX


MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

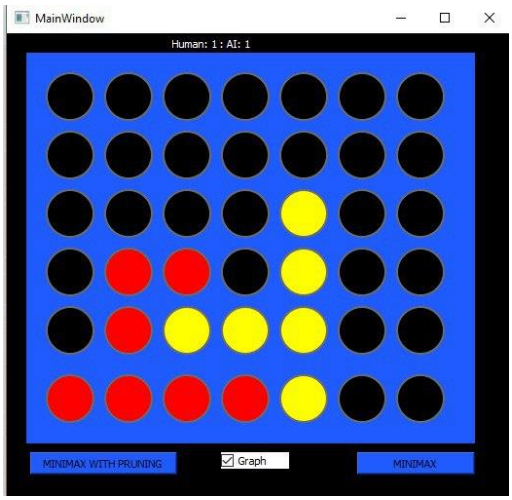

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX


MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX

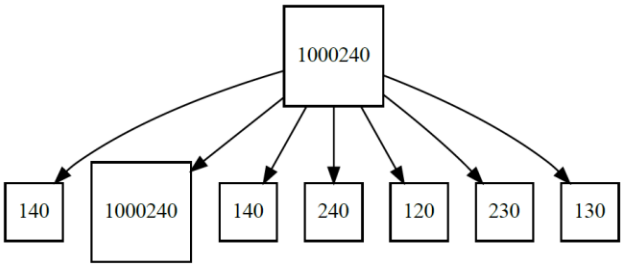

MainWindow
Human: 0 : AI: 1
MINIMAX WITH PRUNING
Graph
MINIMAX



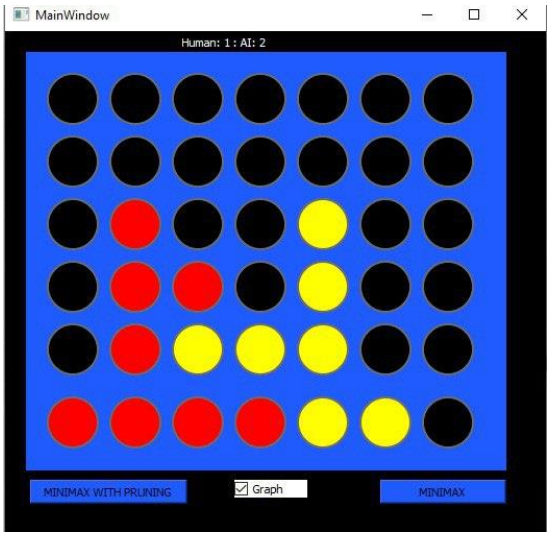
Sample Run of the board and its corresponding minmax tree:



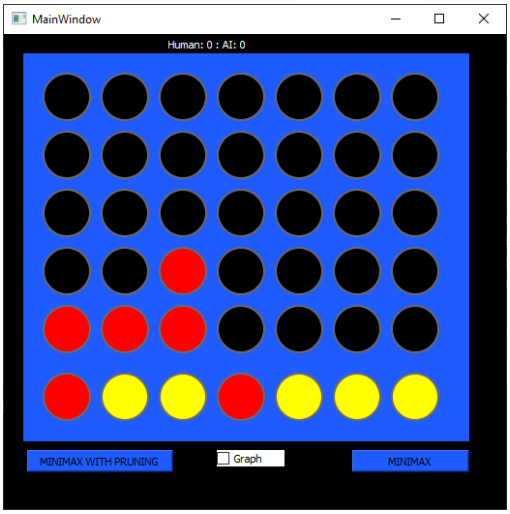
Depth =1:

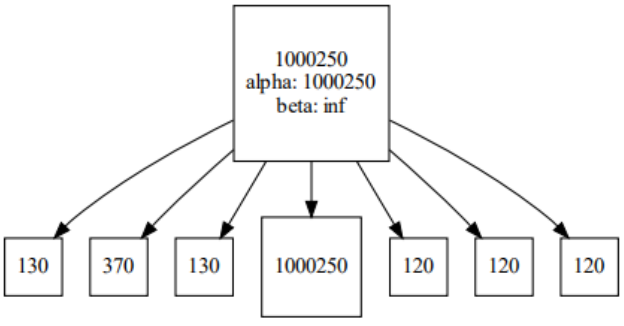


The corresponding child board:

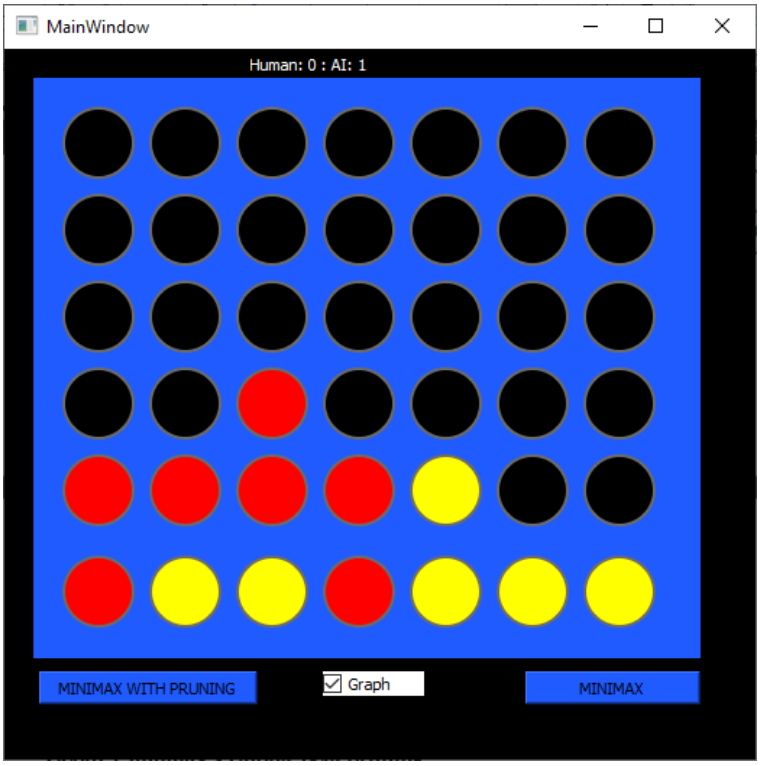


Sample Run of the board and its corresponding minmax tree using alpha-beta pruning



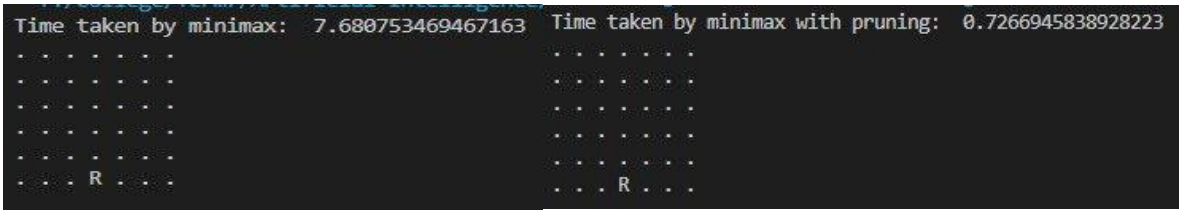


The corresponding child board:



Depth 7 minmax vs alpha-beta pruning :

Depth 6 minmax vs alpha-beta pruning:



Note: the difference in time taken by each algorithm cannot be ignored, these samples will be used in the compariosn between the 2 algorithms a

The Data structures used:

The main data structure used to implement the Board Game is the **bit array.** On a 7x6 board game there are 42 empty spots to play. We represent our board with the bit position of each cell in the board. We use extra 7 bits on the top of the board that helps in the detection of the connected-tows, connected-threes and connected fours. We also use extra 14 bits to represent the last filled cell for each column and the color. That makes a total of **63** bits to represent the full board.

In order to represent the board with all the filled cells we use a **mask** bit array (same size of the board) which helps to get the opponent's board -by applying an XOR operation with the Board when switching the turn or detect the winner at each move , as well as a **position** bit array set to 1 if there is a yellow/red token in the cell which is or-ed later with the **mask** to update it when a player makes a move in the corresponding position.

The Code Structure:

We have 5 code files assigned:

Main.py: This is the first main implemented before building the GUI, it prints the structure of the board in the console (in a simple manner) in order to track the moves. It is also used to print the Minmax Tree in the beginning.

Connect4helpers.py: This file contains all the basic object definitions of the board.

The Gameboard object has the following attributes:

- The bit array of the board
- The bit array of the mask
- The corresponding alpha and beta used in alpha-beta pruning algorithm
- The position which is set to one in the bit that represents the new move
- The bool attribute pruned which checks if there is a pruning or not
- The utility computed by the heuristic to be used later in the algorithms
- The current scores of the human and the agent
- The depth currently used
- The current turn
- The twos heuristic
- The threes heuristic

- The set-initial-utility which sets the utility in the beginning to infinity or -infinity depending on the turn

Another definition **child** is used to generate the new child with the new move and add it to the array **children** used in the minmax and the alpha-beta pruning algorithms.

We are implementing the **get_position()** and the **make_move()** definitions.

Get_position() gets the column where the user/agent wants to add into it to translate it in terms of bit array and adding it to the mask.

Make_move() checks if the column is full and do the OR the operation between the position and the mask and dependently update the board if it is the current turn and then it flips the turn.

Win_detect() , connected_twos() , connected_threes() functions detect the connected-fours, connected-twos and connect-threes in order to add it to the score by logical and arithmetic operations or update the heuristic.

And finally the function **heuristic()** which computes the heuristic of the board by the following equation:

Utility= 1000000*agent_connected_four-1000000 * human_connected_four + 100 * agent_connected_threes-100*human_connected_threes+10*agent_connectes_twos-10*human_connected_twos

Minimax.py: it contains the algorithms of the minmax and the alpha-beta pruning algorithms.

Minmax Algorithm:

Max_value Function:

It starts with a recursive call that checks if the depth equals to zero or not, if it is equal to zero it sets the utility computed from the connect4helpers.py file. If not, it sets the max utility as negative infinity and then loops over the utilities of the seven children that can be generate by each move by calling the Min_value() function (as the children is in the minimizer level) and returns the child that has the maximum utility besides its utility to be set later as its own utility.

Min_value Function:

It starts with a recursive call that checks if the depth equals to zero or not, if it is equal to zero it sets the utility computed from the connect4helpers.py file. If not, it sets the min utility as positive infinity and then, it loops over the utilities of the seven children that can be generate by each move by calling the Max_value() function (as the children is in the maximizer level)

and returns the child that has the minimum utility besides its utility to be set later as its own utility.

Minimax_decision Function:

It is the final function that calls the **max_value()** function to set the current board utility and then it plots the tree of the minimax algorithm taking the depth as an input to draw the tree accordingly. The function returns the child board with its corresponding utility computed in the function.

Minmax with alpha-beta pruning Algorithm:

Max_value_pruning Function:

The function takes the board object, the depth required, alpha and beta values. It starts with a recursive call that checks if the depth equals to zero or not, if it is equal to zero it sets the utility computed from the connect4helpers.py file. If not, it sets the max utility as negative infinity and then loops over the utilities of the seven children that can be generated by each move by calling the Min_value() function (as the children is in the minimizer level), setting the alpha and beta of the child as the board alpha and beta values. If the child utility recently updated is greater than the maximum utility, the child set the max utility as it sown utility updating the value of alpha if the it has a value smaller than the max utility. And then comes the main check for the pruning operation, if the alpha/max utility is greater than or equal to the beta, we prune the remaining children and break from the loop that iterates over the seven children. The function returns the child that has the maximum utility besides its utility to be set later as its own utility.

Min_value_pruning Function:

The function takes the board object, the depth required, alpha and beta values. It starts with a recursive call that checks if the depth equals to zero or not, if it is equal to zero it sets the utility computed from the connect4helpers.py file. If not, it sets the min utility as positive infinity and then loops over the utilities of the seven children that can be generated by each move by calling the Max_value() function (as the children is in the maximizer level), setting the alpha and beta of the child as the board alpha and beta values. If the child utility recently updated is smaller than the minimum utility, the child set the min utility as it sown utility updating the value of beta if the it has a value greater than the min utility. And then comes the main check for the pruning operation, if the beta/minimum utility is smaller than or equal to the alpha we prune the remaining children and break from the loop that iterates over the seven children.

The function returns the child that has the minimum utility besides its utility to be set later as its own utility.

Minimax_decision_pruning Function:

It is the final function that calls the **max_value_pruning()** function to set the current board utility and then it plots the tree of the minimax algorithm taking the depth as an input to draw the tree accordingly. The function returns the child board with its corresponding utility computed in the function.

gui.py: It is the generated code of QT5 designs in order to be used and linked to the other files.

Connect4.py: This file contains the main from which we can run the code associated with the GUI. It calls all the relevant functions listed in the minmax file and the connect4helpers file to be executed in a GUI-way to the user.

Comparison of the 2 Algorithms used

At the end of the project implementation and after trying multiple game samples using both algorithms, it is palpably clear that the alpha-beta pruning algorithm is better which is already derived from the sample run section when comparing the time taken by each algorithm.

Both algorithms give the same answer in the end. However, their main difference is that the alpha-beta pruning does not expand all paths, like minmax does, but prunes those that are guaranteed to be an optimal state for the current player, that is maximizer or minimizer. We can say that the alpha-beta is a optimized version of the minmax.