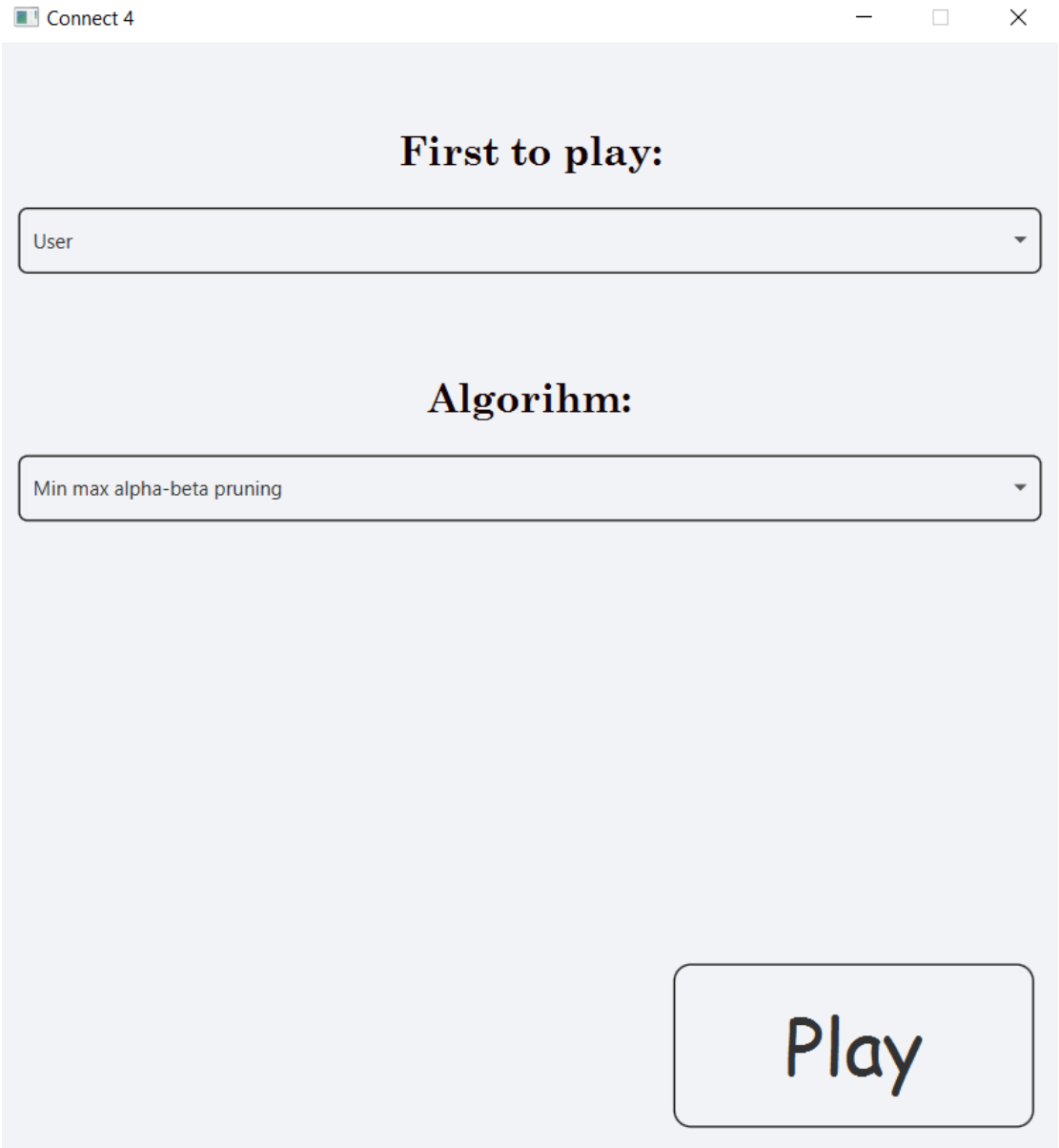
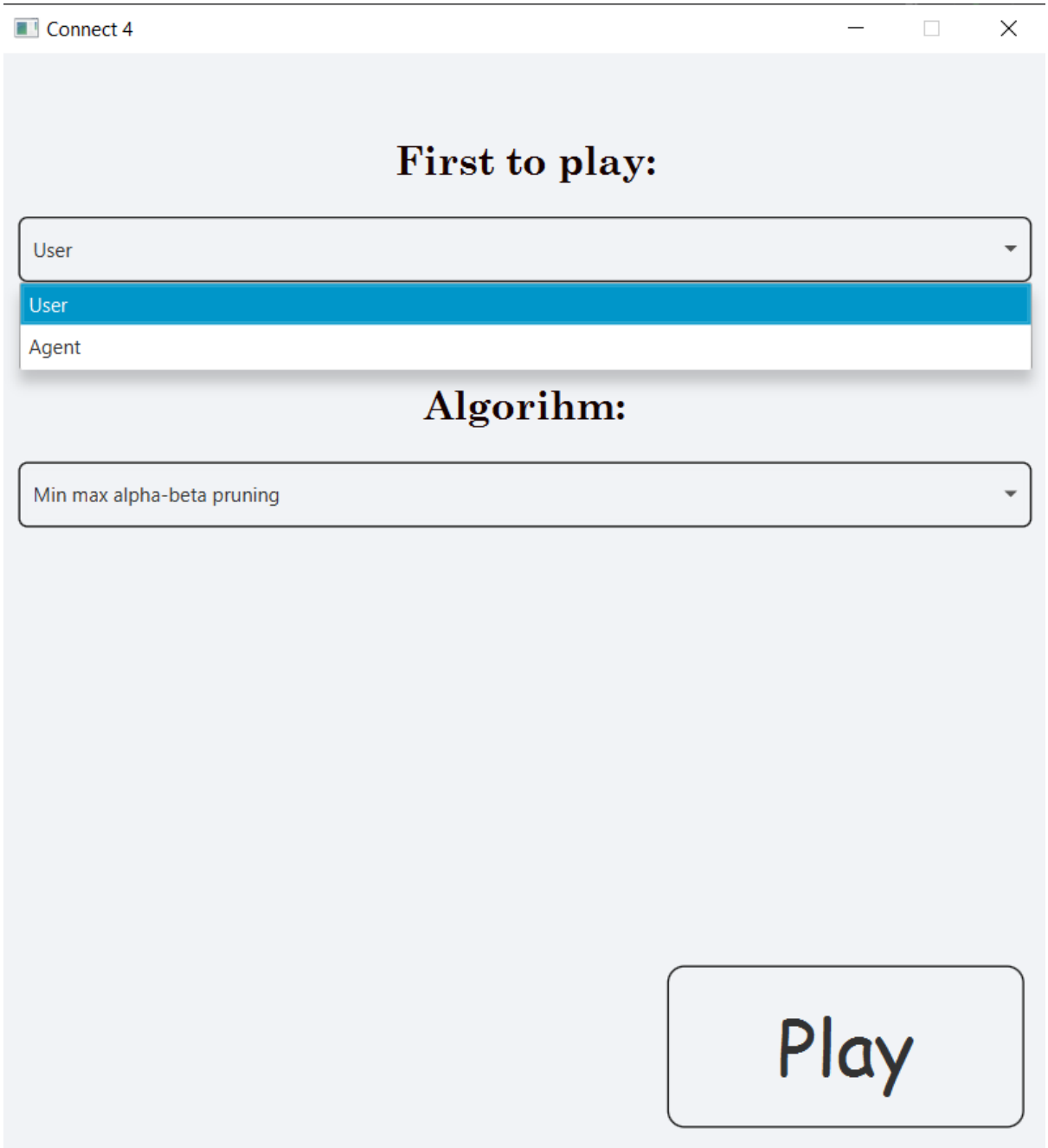
# Connect 4 game with AI

# using minimax algorithm

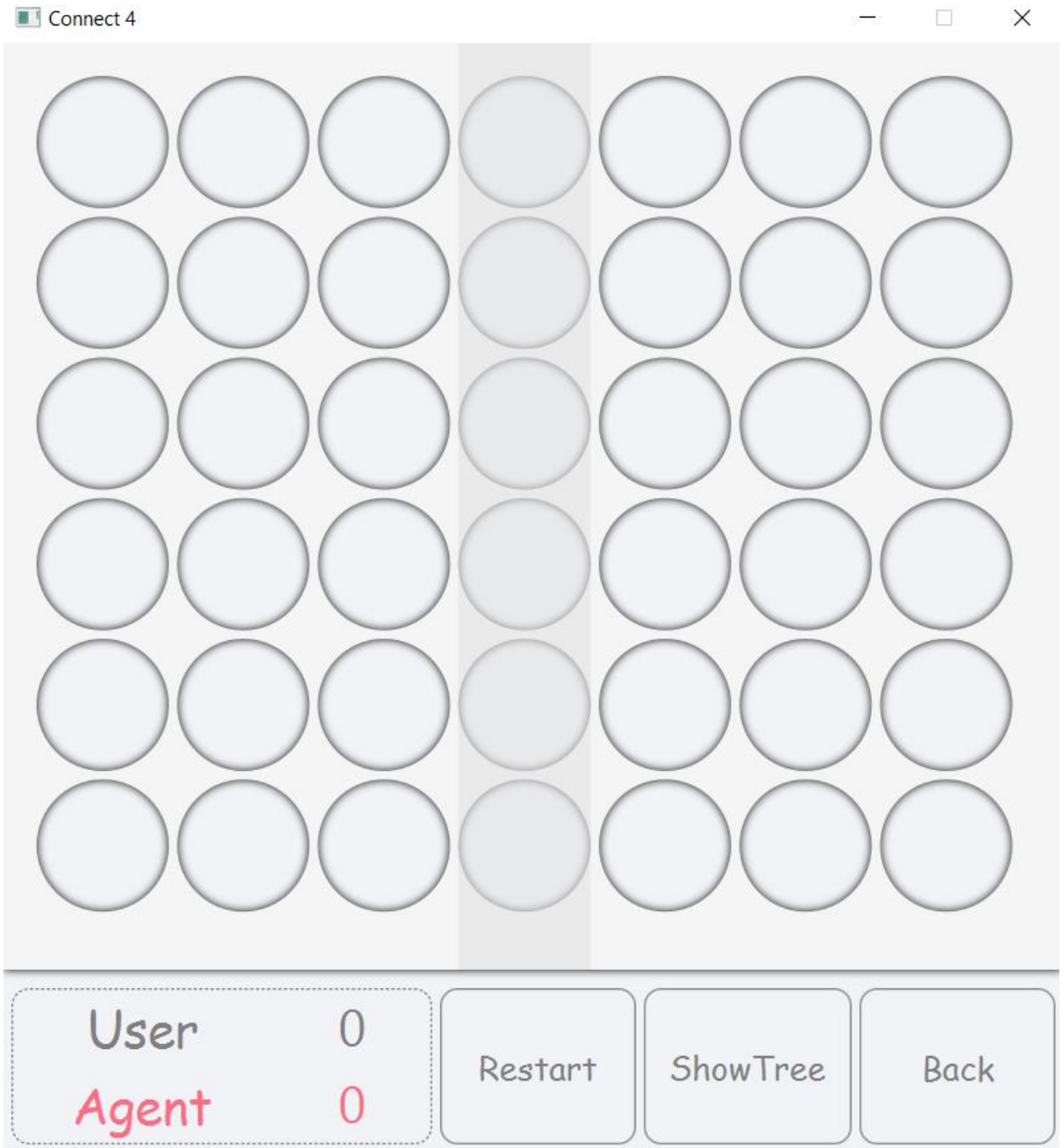

| Name | ID |
| --- | --- |
| Mazen medhat farid | 6160 |
| Ahmed Mohamed ahmed ali | 6338 |
| Mohamed zayton | 6670 |
| Mostafa Mohamed tawfik | 6672 |

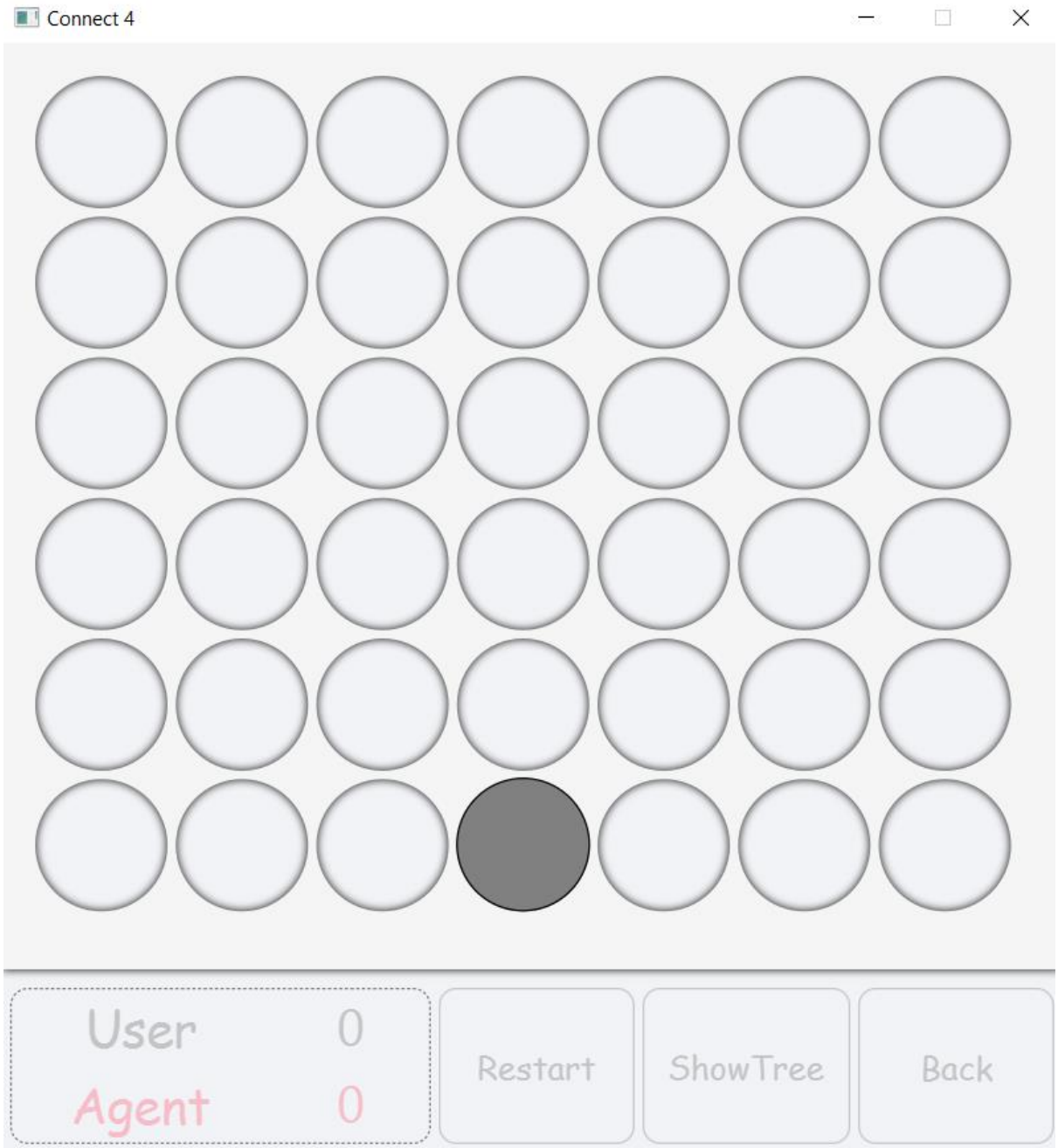
# Sample runs:

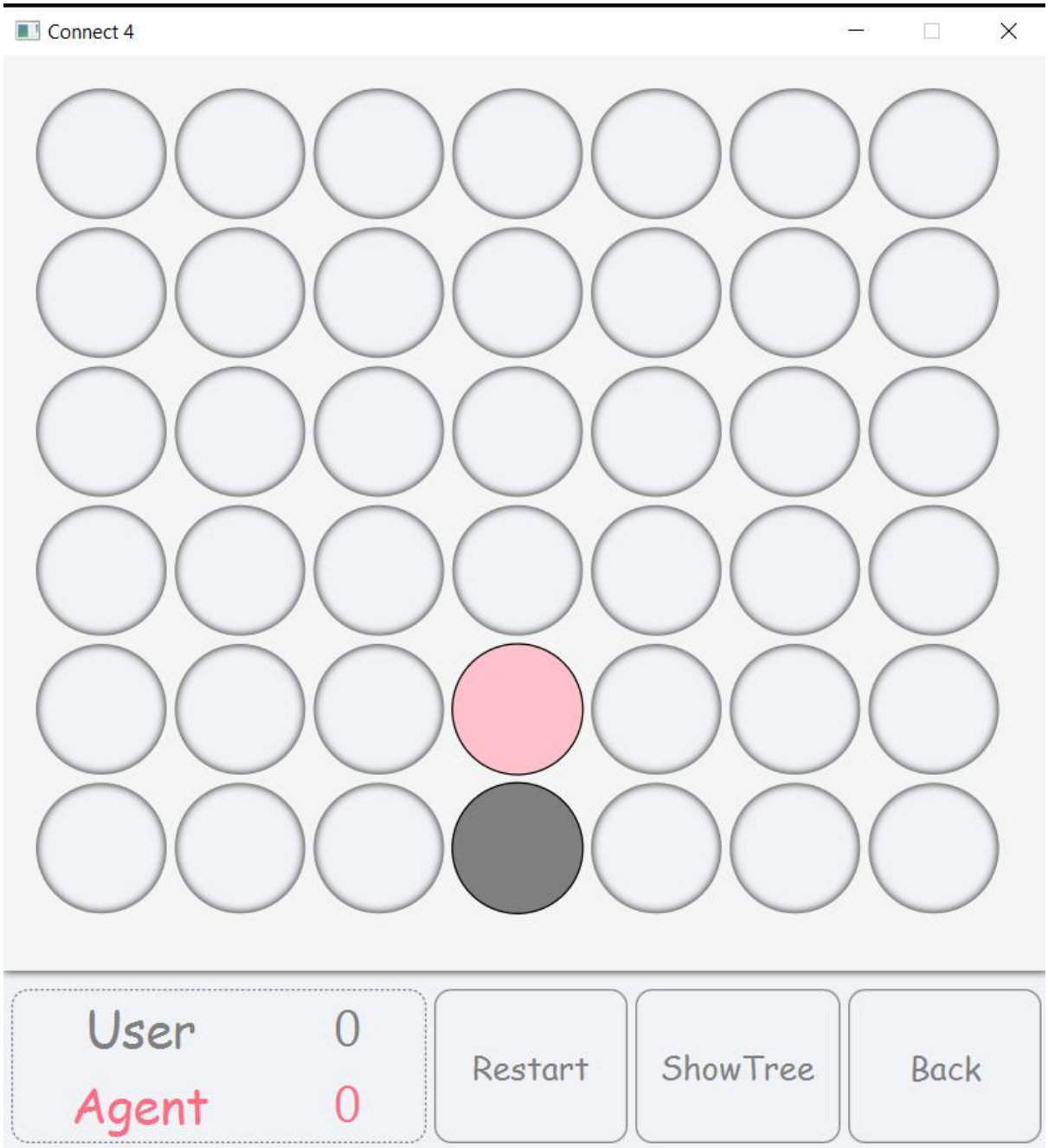

Connect 4
First to play:
User
Algorihm:
Min max alpha-beta pruning
Play

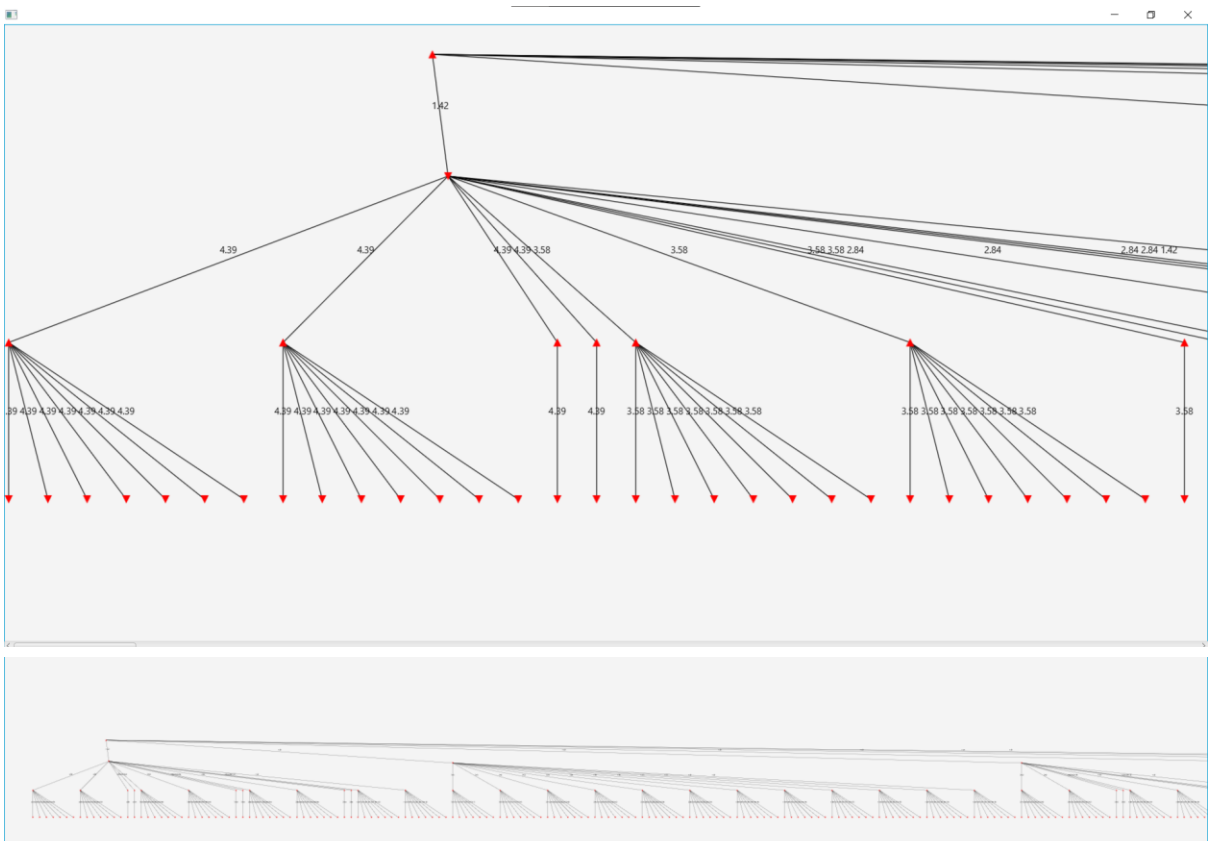

Connect 4
First to play:
User
User
Agent
Algorihm:
Min max alpha-beta pruning
Play

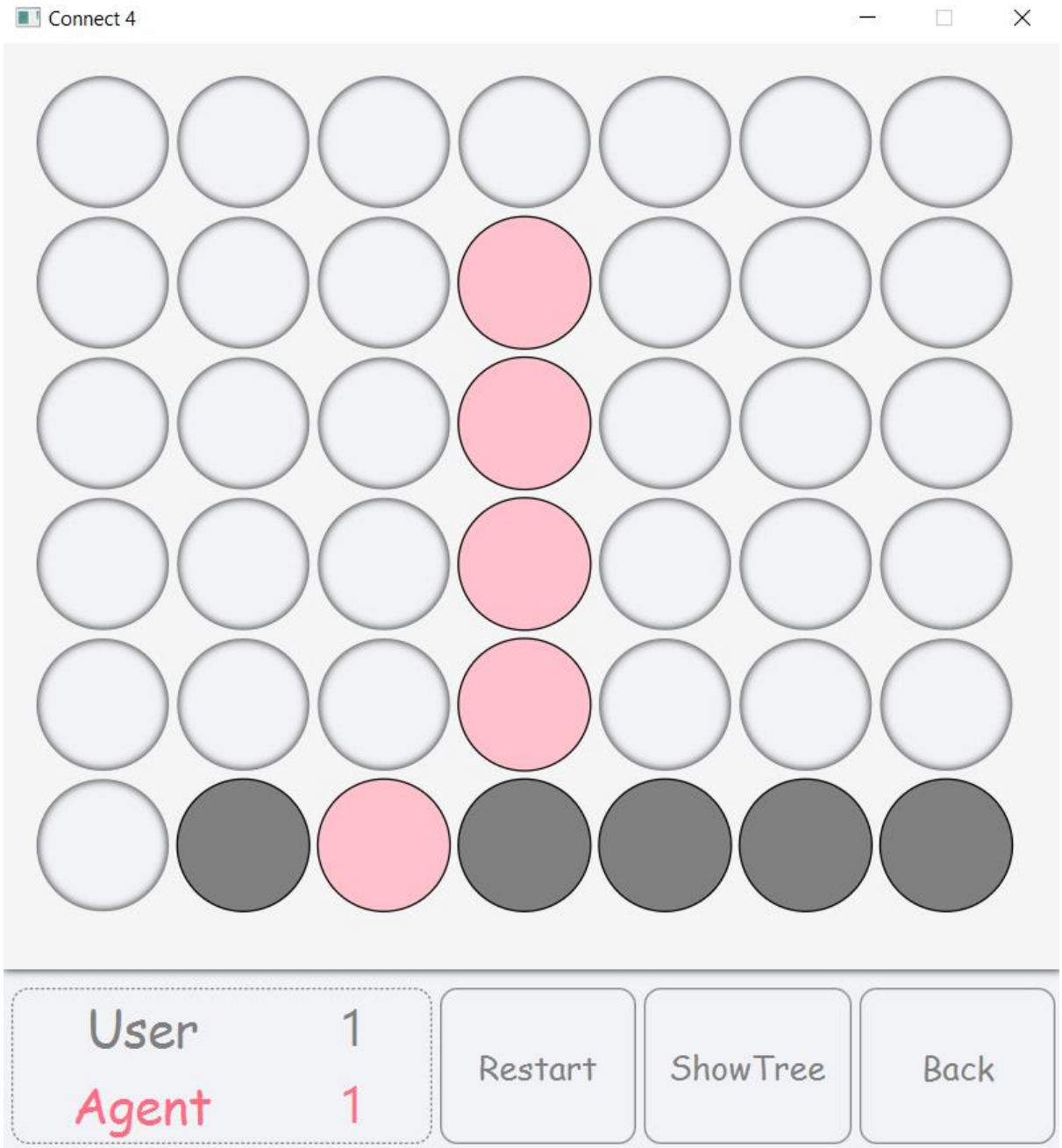

Connect 4
First to play:
User
Algorihm:
Min max alpha-beta pruning
Min max alpha-beta pruning
Min max
Play

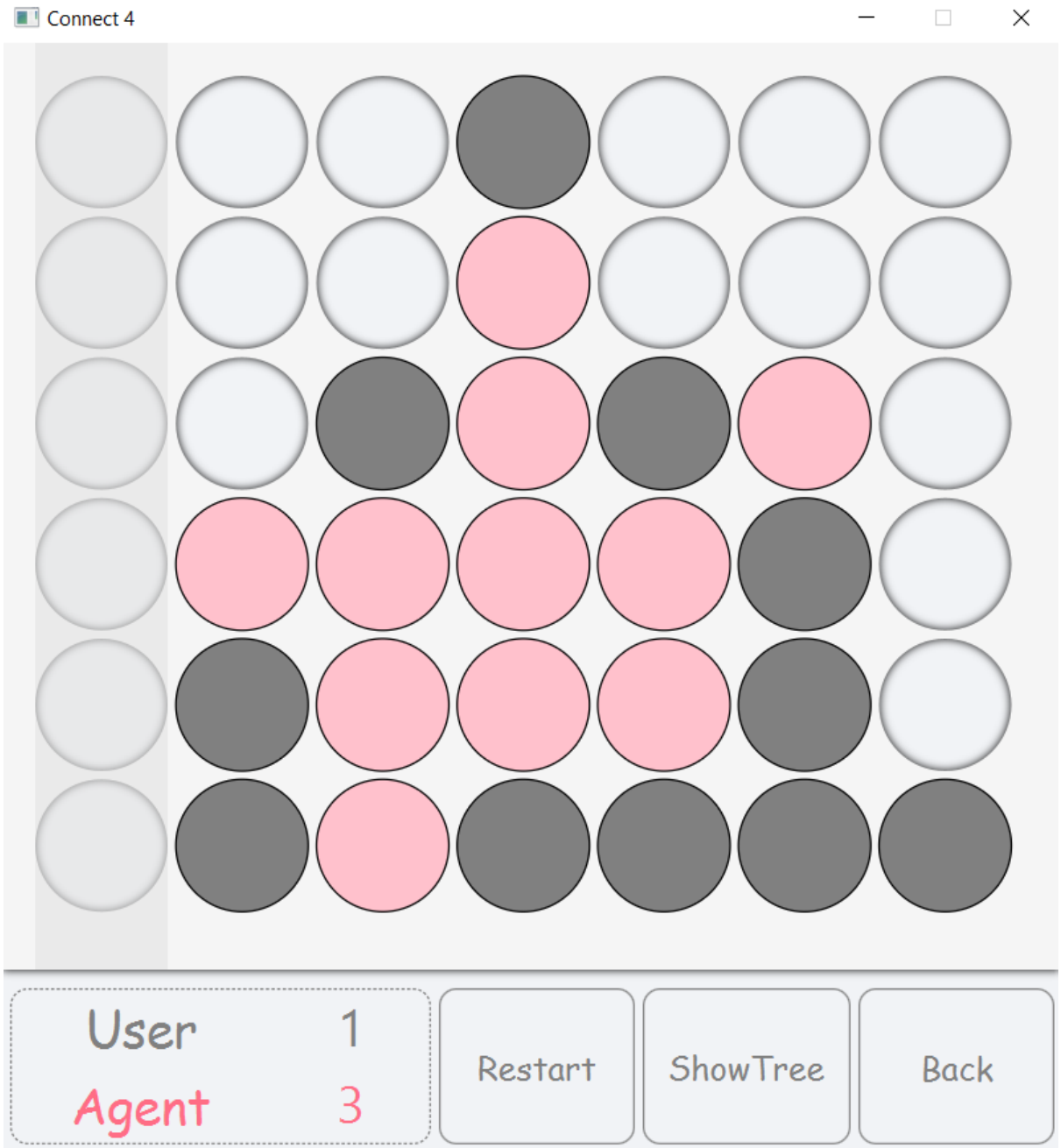

Connect 4
User 0
Agent 0
Restart
ShowTree
Back

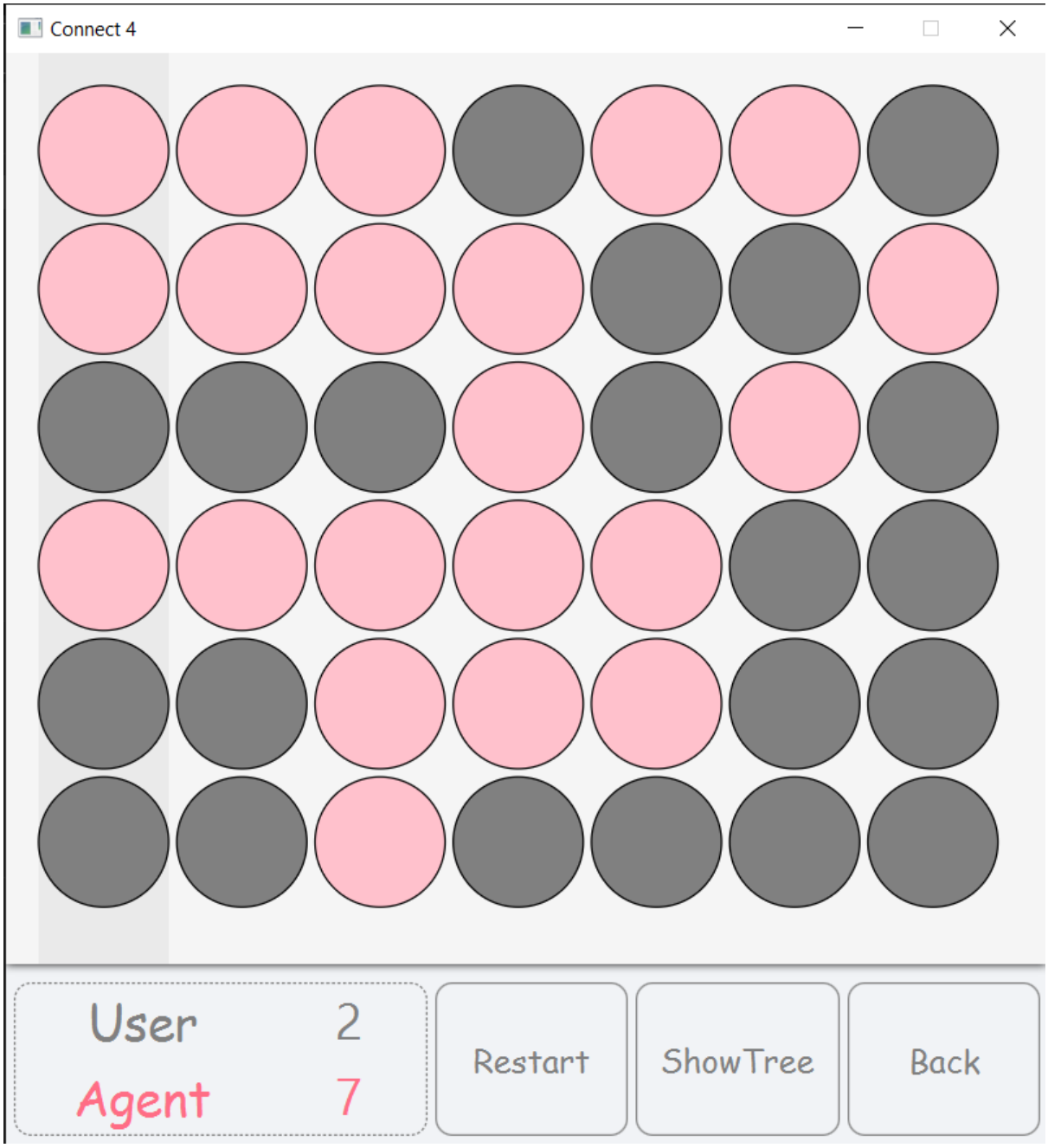

Connect 4
User 0
Agent 0
Restart
ShowTree
Back


Connect 4
User 0
Agent 0
Restart
ShowTree
Back

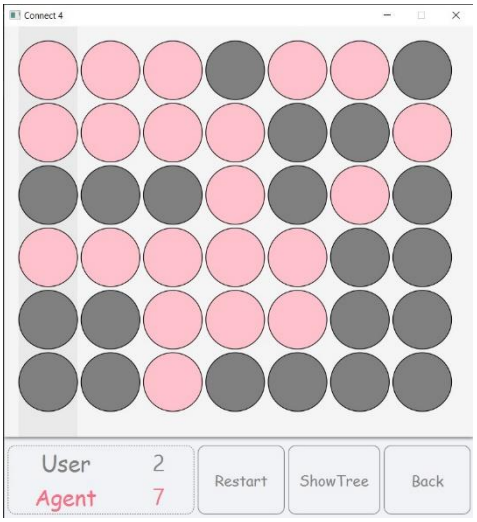

1.42
4.39
4.39
4.39 4.39 3.58
3.58
3.58 3.58 2.84
2.84
2.84 2.84 1.42
4.39 4.39 4.39 4.39 4.39 4.39 4.39
4.39 4.39 4.39 4.39 4.39 4.39 4.39
4.39
4.39
3.58 3.58 3.58 3.58 3.58 3.58 3.58
3.58 3.58 3.58 3.58 3.58 3.58 3.58
3.58

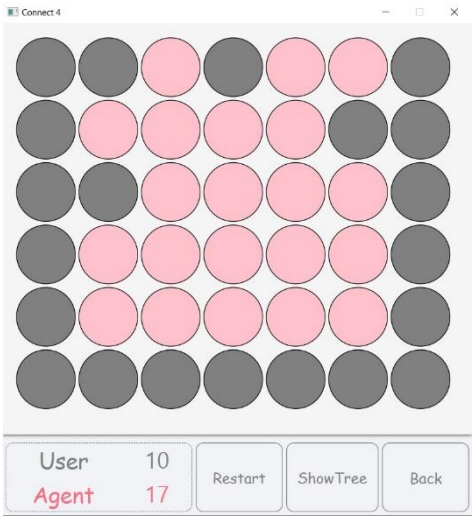

Connect 4
User 1
Agent 1
Restart
ShowTree
Back

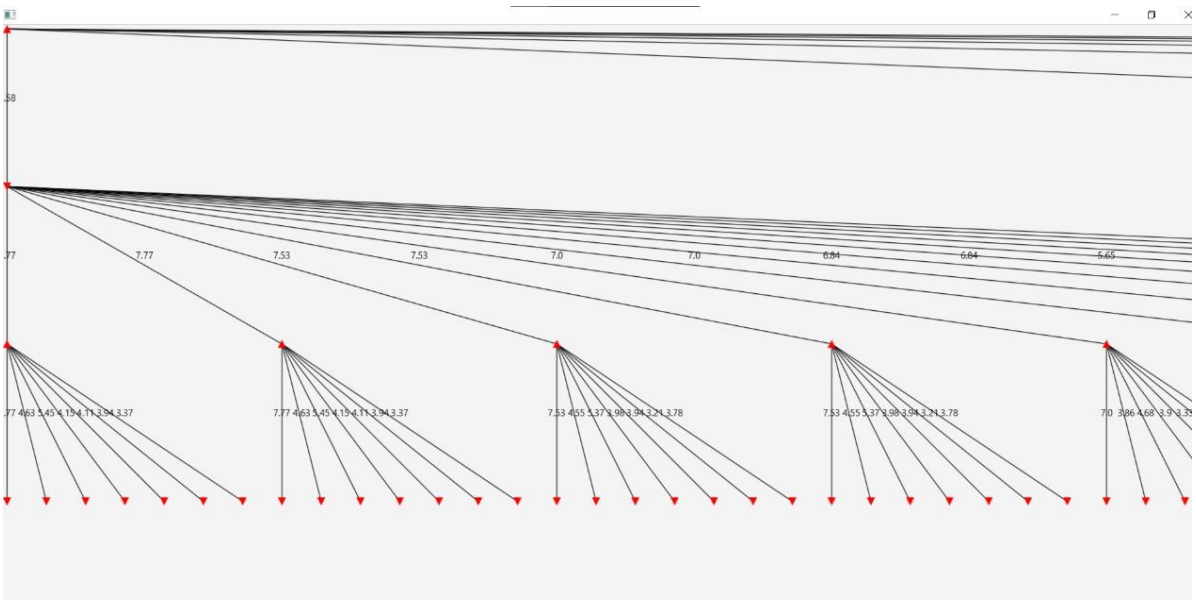

Connect 4
User 1
Agent 3
Restart
ShowTree
Back


Connect 4
User 2
Agent 7
Restart
ShowTree
Back

# Minimax without alpha beta:



# Comparison between 2 algorithms:

```
C:\Users\MohamedMedhatSaeedHa\.jdks\openjdk-18\bin\java.exe ...
At Depth 10
Time MinimaxAlphaBeta : 1.626
Expanded Nodes MinimaxAlphaBeta : 10157657
**********************************************************************
Exception in Application start method
OpenJDK 64-Bit Server VM warning: Exception java.lang.OutOfMemoryError occurred dispatching signal UNKNOWN to handler- the VM may need to be forcibly terminated
Exception in thread "JavaFX Application Thread"
Exception: java.lang.OutOfMemoryError thrown from the UncaughtExceptionHandler in thread "main"
```

```
C:\Users\MohamedMedhatSaeedHa\.jdks\openjdk-18\bin\java.exe ...
At Depth 8
Time MinimaxAlphaBeta : 0.266
Expanded Nodes MinimaxAlphaBeta : 611799
**********************************************************************
Time MiniMax : 4.653
Expanded Nodes Minimax : 17678113
```

```
C:\Users\MohamedMedhatSaeedHa\.jdks\openjdk-18\bin\java.exe ...
At Depth 7
Time MinimaxAlphaBeta : 0.115
Expanded Nodes MinimaxAlphaBeta : 112415
**********************************************************************
Time MiniMax : 1.001
Expanded Nodes Minimax : 1906049
```

# Data structures used:

- Pair
- Tree
- ArrayList

# State representation:

State is represented as long where each column in the 2d grid is represented as 9 bits 3 bits for number of occupied slots and 6 bits where each if the 6 bits denote who is playing user or agent.

# Pseudo Code:

# Minmax:

```
package algorithms;


import javafx.util.Pair;
import logic.Heuristic;
import logic.SlotState;
import logic.StateOperations;

public class MiniMax {


    public static TreeNode root;
    private static int maxDepth = 7;

    public static Pair<Long, TreeNode> decision(long state){
        root = new TreeNode(state, 0, true);
        var value = max(state, root, 0);
        root.val = value.getValue();
        Pair<Long, TreeNode> val = new Pair<>(value.getKey(), root);
        return val;
    }

    private static Pair<Long, Double> max(long state, TreeNode node, int depth) {

        if (StateOperations.getEmptySlotsCount(state) == 0 || depth >= maxDepth)
            return new Pair<Long, Double>(null , (double)
Heuristic.getStateScore(state));

        long maxChild = 0;
        double maxUtility = Double.NEGATIVE_INFINITY;

        for (var neighbour : StateOperations.getStateChildren(state,
SlotState.AGENT)) {
            var nodec = new TreeNode(neighbour,0, false);
            node.children.add(nodec);
            var value = min(neighbour,nodec ,depth + 1);
            var utility = value.getValue();
            nodec.val = utility;
           if (utility > maxUtility){
               maxChild = neighbour;
               maxUtility = utility;
           }

        }

        return new Pair<Long, Double>(maxChild, maxUtility);
    }

    private static Pair<Long, Double> min(long state, TreeNode node, int depth) {
        if (StateOperations.getEmptySlotsCount(state) == 0 || depth >= maxDepth)
            return new Pair<Long, Double>(null , (double)
Heuristic.getStateScore(state));
        long minChild = 0;
```

```
        double minUtility = Double.POSITIVE_INFINITY;

        for (long neighbour : StateOperations.getStateChildren(state,
SlotState.USER)) {
            var nodec = new TreeNode(neighbour,0, true);
            node.children.add(nodec);
            var value = max(neighbour, nodec,  depth+1);
            var utility = value.getValue();
            nodec.val = utility;
            node.children.add(nodec);
            if (utility < minUtility){
                minChild = neighbour;
                minUtility = utility;
            }

        }
        return new Pair<Long, Double>(minChild, minUtility);
    }


}
```

# Minimax with alpha-beta:

```
package algorithms;

import javafx.util.Pair;
import logic.Heuristic;
import logic.Node;
import logic.SlotState;
import logic.StateOperations;

import java.util.HashMap;
import java.util.Vector;

public class MinimaxAlphaBeta {

    static int maxDepth = 10;
    static TreeNode root = null;
    public static Pair<Long, TreeNode> decision(long state){
        root = new TreeNode(state, 0, true);
        var value = maximize(state, root, Double.NEGATIVE_INFINITY,
Double.POSITIVE_INFINITY, 0);
        root.val = value.getValue();
        Pair<Long, TreeNode> val = new Pair<>(value.getKey(), root);
        return val;
    }

    private static Pair<Long, Double> maximize(long state, TreeNode node, double
alpha, double beta, int depth) {
        if (StateOperations.getEmptySlotsCount(state) == 0 || depth >= maxDepth )
            return new Pair<Long, Double>(null , (double)
Heuristic.getStateScore(state));

        long maxChild = 0;
        double maxUtility = Double.NEGATIVE_INFINITY;
```

```java
        for (var c : StateOperations.getStateChildren(state, SlotState.AGENT)) {
            var nodec = new TreeNode(c,0, false);
            node.children.add(nodec);
            var value = minimize(c, nodec, alpha, beta, depth+1);
            var utility = value.getValue();
            nodec.val = utility;
            if (utility > maxUtility){
                maxChild = c;
                maxUtility = utility;
            }
            if (maxUtility >= beta)
                break;
            if (maxUtility > alpha)
                alpha = maxUtility;
        }

        return new Pair<Long, Double>(maxChild, maxUtility);


    }

    private static Pair<Long, Double> minimize(long state, TreeNode node, double
alpha, double beta, int depth) {
        if (StateOperations.getEmptySlotsCount(state) == 0 || depth >= maxDepth)
            return new Pair<Long, Double>(null , (double)
Heuristic.getStateScore(state));
        long minChild = 0;
        double minUtility = Double.POSITIVE_INFINITY;

        for (var c : StateOperations.getStateChildren(state, SlotState.USER)) {
            var nodec = new TreeNode(c,0, true);
            node.children.add(nodec);
            var value = maximize(c, nodec, alpha, beta, depth+1);
            var utility = value.getValue();
            nodec.val = utility;
            node.children.add(nodec);
            if (utility < minUtility){
                minChild = c;
                minUtility = utility;
            }
            if (minUtility <= alpha)
                break;
            if (minUtility < beta)
                beta = minUtility;
        }
        return new Pair<Long, Double>(minChild, minUtility);

    }
}
```