Course ID: CSE-434

Date : 11/02/2014

Report for Artificial Intelligence Project

Submitted to,

Abu Naser Tanu
Shahjalal University of Science and Technology

Submitted by,

Rajib Chandra Das (2009331008)
Nafis Ahmed (2009331042)

Project Name:

5 in a Row - Tic Tac Toe (10 x 10 Grid) with AI Strategy.

Project Requirement:

I. Language: Java.
II. Knowledge: Recursion, Min Max algorithm, Heuristic Function, Alpha Beta Pruning, Object Oriented Programming Concept.
III. Tools: Netbeans, Adobe Photoshop, Gliffy.

Game Description:

It is an abstract strategy board game. In this game, Human = 1st player and CPU (Computer) = 2nd Player. Program decides randomly who will give the first move. Human players are able to select only blank space to give move and on the contrary, Computer gives the optimal move from learned AI strategy. Making a 5 in a row in the board, a player can be winner. When the board is filled fully and there is no space to move for any player then this criteria will be considered as drawn.

In this project, we built the game with an intelligent computer player. We will explain each step more clearly in the following sections.

We have used utility functions and evaluation functions to guide the minimax search.

Moreover, our game has three level, Beginner, Intermediate and Advanced.

Flow Chart:

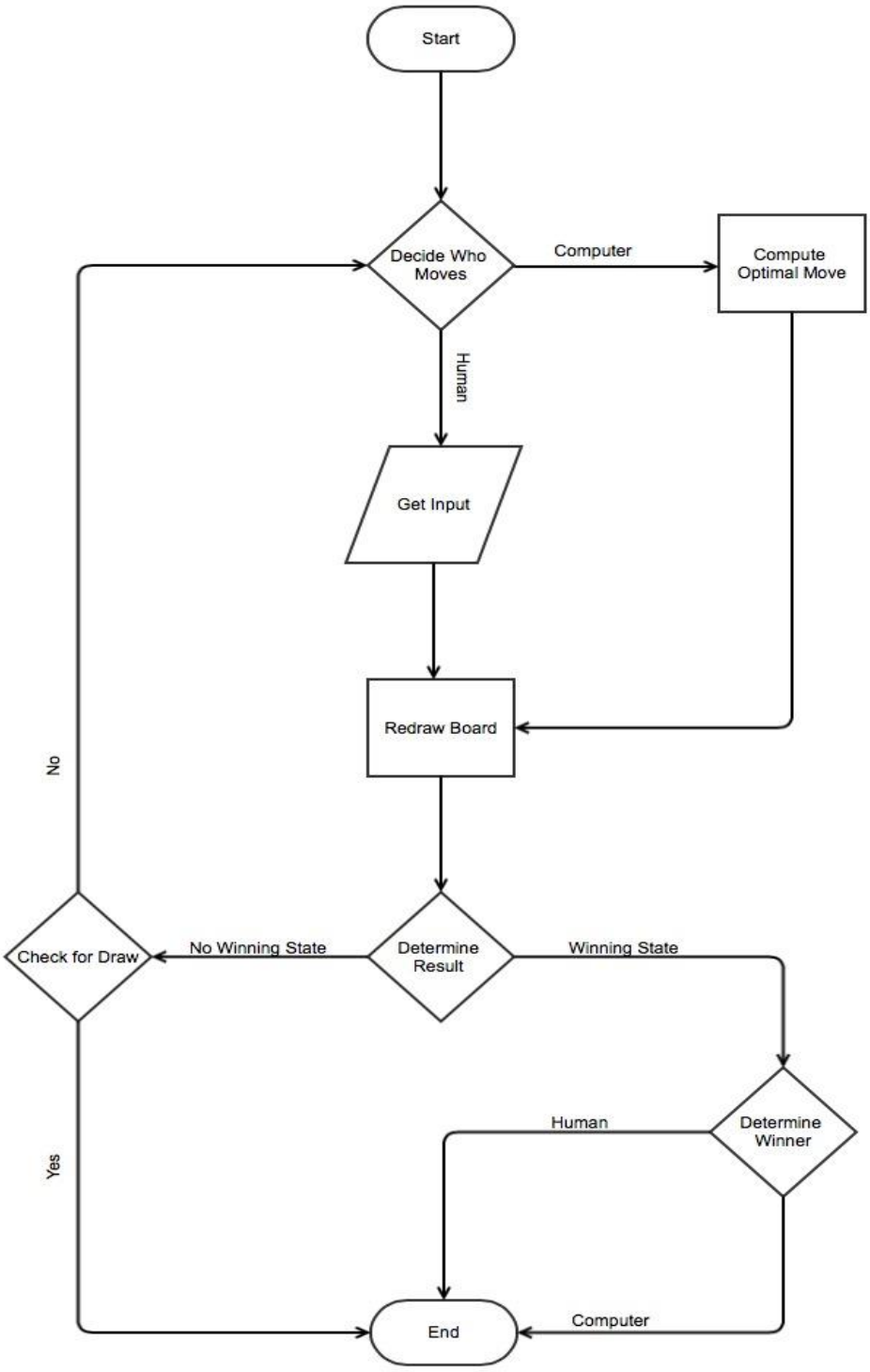


Figure : Flow Chart for 5 in a row Tic Tac Toe

Heuristic/ Utility Function:

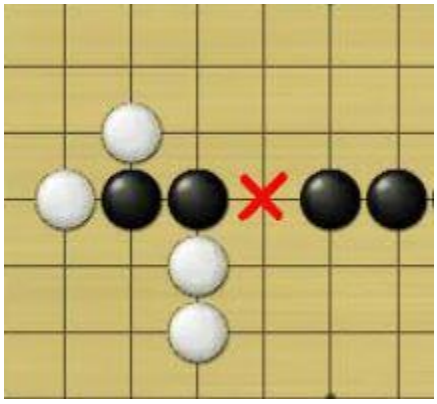
Here,

Computer - Black (O) = Max Player.

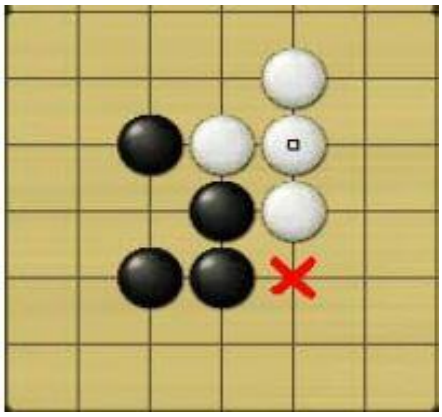
Human - White (X) = Min Player.

We need some knowledge about different criteria in this board game. Here are some moves for Black (computer):

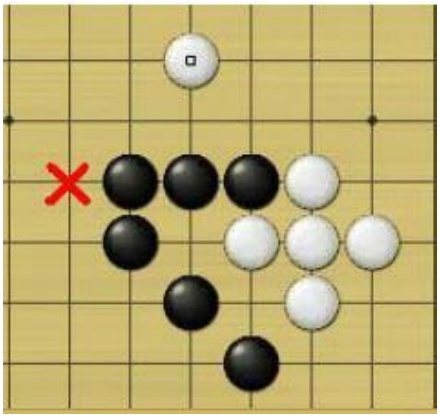
One Tricky Adjecent 5:



a) Two Live 3:



b) One sleep 4 & One Live 4:



Utility Score Distributions:

Criteria	Utility Score for Max Player	Utility Score for Min Player
Dead Five	500000	-500000
One Live 4 Two Sleep 4 Two Tricky Adjacent 5 One Tricky Adjacent 5 & One Sleep 4 One Tricky Adjacent 5 & Two Live 3 One Live 3 & One Sleep 4	100000	-100000
Two Live 3	40000	-40000
One Sleep 3 & One Live 3	10000	-10000
One Sleep 4 One Tricky Adjacent 5	4000	-4000
One Live 3	1000	-1000
Two Live 2	400	-400
One Sleep 3	100	-100
Two Sleep 2	40	-40
One Live 2	10	-10
One Sleep 2	4	-4
For Empty Board when at Middle Cell	1	-1

Beginner Level:

For Beginner Level we used only the heuristic function with some clever adjustments. We determined the utility score for every possible move according to previous human player's move. We also tried to simulate what the next move will be and for that how our score will change.

Used Function Names:

i) Point getOptimalMove(int[][] board, int n, int PlayerNum);
ii) Int Easy_Get_Utility_Score(int[][] Board, int PlayerNum, Point p);
iii) int Dead_Five(int[][] Board, int x, int y, int PlayerNum);
iv) int Tricky_Adjacent(int[][] Board, int x, int y, int PlayerNum, int num);
v) int Live_Adjacent(int[][] Board, int x, int y, int PlayerNum, int num);
vi) int Sleep_Adjacent(int[][] Board, int x, int y, int PlayerNum, int num);
vii) public Search_Location(int[][] Board, int x, int y, int n, int playerNum);

Algorithm for Beginner level:

```
int Easy_Get_Utility_Score(int[][] Board, int PlayerNum, Point p)
    {
        int X = p.x;
        int Y = p.y;
        int utility;
        if(PlayerNum==1) utility = -1;
        else utility = 1;

        /* For Complete Five In a row */
        if(Dead_Five(Board, X, Y, PlayerNum)>=1) return 500000*utility;
        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Dead_Five(Board, X, Y, 2)>=1) return 400000*utility;
            Board[X][Y]=1;
        }
        else
        {
            Board[X][Y]=1;
            if(Dead_Five(Board, X, Y, 1)>=1) return 400000*utility;
            Board[X][Y]=2;
        }
```

```
/*For Live 4 in a row, Two Sleep 4 in a row, One Live 3 in a row + One Sleep
4 in a row, Two Tricky Adjacent*/
        if(Live_Adjacent(Board, X, Y, PlayerNum, 4)>=1) return
100000*utility;
        if(Sleep_Adjacent(Board, X, Y, PlayerNum, 4)>=2) return
100000*utility;
        if(Tricky_Adjacent(Board, X, Y, PlayerNum, 5)>=2) return
100000*utility;
        if(Tricky_Adjacent(Board, X, Y, PlayerNum,
5)>=1&&Sleep_Adjacent(Board, X, Y, PlayerNum, 4)>=1) return 100000*utility;
        if(Tricky_Adjacent(Board, X, Y, PlayerNum,
5)>=1&&Live_Adjacent(Board, X, Y, PlayerNum, 3)>=2) return 100000*utility;
        if(Live_Adjacent(Board, X, Y, PlayerNum, 3)>=1 &&
Sleep_Adjacent(Board, X, Y, PlayerNum, 4)>=1) return 100000*utility;

        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Live_Adjacent(Board, X, Y, 2, 4)>=1) return 90000*utility;
            if(Sleep_Adjacent(Board, X, Y, 2, 4)>=2) return 90000*utility;
            if(Tricky_Adjacent(Board, X, Y, 2, 5)>=2) return 90000*utility;
            if(Tricky_Adjacent(Board, X, Y, 2, 5)>=1&&Sleep_Adjacent(Board,
X, Y, 2, 4)>=1) return 90000*utility;
            if(Tricky_Adjacent(Board, X, Y, 2, 5)>=1&&Live_Adjacent(Board,
X, Y, 2, 3)>=2) return 90000*utility;
            if(Live_Adjacent(Board, X, Y, 2, 3)>=1 && Sleep_Adjacent(Board,
X, Y, 2, 4)>=1) return 90000*utility;
            Board[X][Y]=1;
        }
        else
        {
            Board[X][Y]=1;
            if(Live_Adjacent(Board, X, Y, 1, 4)>=1) return 90000*utility;
            if(Sleep_Adjacent(Board, X, Y, 1, 4)>=2) return 90000*utility;
            if(Tricky_Adjacent(Board, X, Y, 1, 5)>=2) return 90000*utility;
            if(Tricky_Adjacent(Board, X, Y, 1, 5)>=1&&Sleep_Adjacent(Board,
X, Y, 1, 4)>=1) return 90000*utility;
            if(Tricky_Adjacent(Board, X, Y, 1, 5)>=1&&Live_Adjacent(Board,
X, Y, 1, 3)>=2) return 90000*utility;
            if(Live_Adjacent(Board, X, Y, 1, 3)>=1 && Sleep_Adjacent(Board,
X, Y, 1, 4)>=1) return 90000*utility;
            Board[X][Y]=2;
        }

        /* For Two Live 3 in a row */
        if(Live_Adjacent(Board, X, Y, PlayerNum, 3)>=2) return
40000*utility;
        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Live_Adjacent(Board, X, Y, 2, 3)>=2) return 30000*utility;
            Board[X][Y]=1;
        }
        else
        {
            Board[X][Y]=1;
```

```
            if(Live_Adjacent(Board, X, Y, 1, 3)>=2) return 30000*utility;
            Board[X][Y]=2;
        }


/* For one sleep 3 in a row + one live 3 in a row */
        if(Sleep_Adjacent(Board, X, Y, PlayerNum, 3)>=1 &&
Live_Adjacent(Board, X, Y, PlayerNum, 3)>=1) return 10000*utility;
        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Sleep_Adjacent(Board, X, Y, 2, 3)>=1 && Live_Adjacent(Board,
X, Y, 2, 3)>=1) return 9000*utility;
            Board[X][Y]=1;
        }
        else
        {
            Board[X][Y]=1;
            if(Sleep_Adjacent(Board, X, Y, 1, 3)>=1 && Live_Adjacent(Board,
X, Y, 1, 3)>=1) return 9000*utility;
            Board[X][Y]=2;
        }

         /*One Sleep 4 in a Row Or One Tricky Adjecent*/
        if(Sleep_Adjacent(Board, X, Y, PlayerNum, 4)>=1) return
4000*utility;
        if(Tricky_Adjacent(Board, X, Y, PlayerNum, 5)>=1) return
3999*utility;
        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Sleep_Adjacent(Board, X, Y, 2, 4)>=1) return 3000*utility;
            if(Tricky_Adjacent(Board, X, Y, 2, 5)>=1) return 2999*utility;
            Board[X][Y]=1;
        }
        else
        {
            Board[X][Y]=1;
            if(Sleep_Adjacent(Board, X, Y, 1, 4)>=1) return 3000*utility;
            if(Tricky_Adjacent(Board, X, Y, 2, 5)>=1) return 2999*utility;
            Board[X][Y]=2;
        }


        /* For One Live 3 in a Row*/
        if(Live_Adjacent(Board, X, Y, PlayerNum, 3)>=1) return 1000*utility;
        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Live_Adjacent(Board, X, Y, 2, 3)>=1) return 900*utility;
            Board[X][Y]=1;
        }
        else
        {
            Board[X][Y]=1;
            if(Live_Adjacent(Board, X, Y, 1, 3)>=1) return 900*utility;
            Board[X][Y]=2;
```

```
        }



        /* For Two Live 2 in a row*/
        if(Live_Adjacent(Board, X, Y, PlayerNum, 2)>=2) return 400*utility;
        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Live_Adjacent(Board, X, Y, 2, 2)>=2) return 300*utility;
            Board[X][Y]=1;
        }
        else
        {
            Board[X][Y]=1;
            if(Live_Adjacent(Board, X, Y, 1, 2)>=2) return 300*utility;
            Board[X][Y]=2;
        }

         /* For One Sleep 3 in a row*/
        if(Sleep_Adjacent(Board, X, Y, PlayerNum, 3)>=1) return 100*utility;
        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Sleep_Adjacent(Board, X, Y, 2, 3)>=1) return 90*utility;
            Board[X][Y]=1;
        }
        else
        {
            Board[X][Y]=1;
            if(Sleep_Adjacent(Board, X, Y, 1, 3)>=1) return 90*utility;
            Board[X][Y]=2;
        }

        /*For Two Sleep 2 in row*/
        if(Sleep_Adjacent(Board, X, Y, PlayerNum, 2)>=2) return 40*utility;
        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Sleep_Adjacent(Board, X, Y, 2, 2)>=2) return 30*utility;
            Board[X][Y]=1;
        }
        else
        {
            Board[X][Y]=1;
            if(Sleep_Adjacent(Board, X, Y, 1, 2)>=2) return 30*utility;
            Board[X][Y]=2;
        }

        /*For one Live 2 in a row*/
        if(Live_Adjacent(Board, X, Y, PlayerNum, 2)>=1) return 10*utility;
        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Live_Adjacent(Board, X, Y, 2, 2)>=1) return 9*utility;
            Board[X][Y]=1;
```

```
        }
        else
        {
            Board[X][Y]=1;
            if(Live_Adjacent(Board, X, Y, 1, 2)>=1) return 9*utility;
            Board[X][Y]=2;
        }

        /*For One sleep 2 in a row*/
        if(Sleep_Adjacent(Board, X, Y, PlayerNum, 2)>=1) return 4*utility;
        if(PlayerNum==1)
        {
            Board[X][Y]=2;
            if(Sleep_Adjacent(Board, X, Y, 2, 2)>=1) return 3*utility;
            Board[X][Y]=1;
        }
        else
        {
            Board[X][Y]=1;
            if(Sleep_Adjacent(Board, X, Y, 1, 2)>=1) return 3*utility;
            Board[X][Y]=2;
        }

        if(X==SZ/2&&Y==SZ/2) return 1;
        return 0;
    }
```

Intermediate & Advanced Level:

For Intermediate and Advanced level we have used MiniMax Search Algorithm with heuristic function which searches until 2 depths for intermediate level and 4 depths for Advanced level.

Used Function Names:

i) Point getOptimalMove(int[][] board, int n, int PlayerNum);
ii) int MiniMax(int[][] Board, int PlayerNum, Point p, int depth);
iii) Int Get_Utility_Score(int[][] Board, int PlayerNum, Point p);
iv) int Dead_Five(int[][] Board, int x, int y, int PlayerNum);
v) int Tricky_Adjacent(int[][] Board, int x, int y, int PlayerNum, int num);
vi) int Live_Adjacent(int[][] Board, int x, int y, int PlayerNum, int num);
vii) int Sleep_Adjacent(int[][] Board, int x, int y, int PlayerNum, int num);
viii) public Search_Location(int[][] Board, int x, int y, int n, int playerNum);

Algorithm for Intermediate and Advanced level:

```
int MiniMax(int[][] Board, int PlayerNum, Point p, int depth)
    {
        if(depth==0) return Get_Utility_Score(Board, PlayerNum, p);
        if(Dead_Five(Board, p.x, p.y, PlayerNum)>=1) return
Get_Utility_Score(Board, PlayerNum, p);
        int bestValue, val, utl;

        utl = Get_Utility_Score(Board, PlayerNum, p);
        if (PlayerNum == 1)
        {
            bestValue = -(1<<30);
            for (int i = 0; i < SZ; i++)
            {
                for (int j = 0; j < SZ; j++)
                {
                    if (Board[i][j] == 0)
                    {
                        Board[i][j] = 2;
                        val = MiniMax(Board, 2, new Point(i, j), depth-1);
                        if (bestValue < val)
                           bestValue = val;
                        Board[i][j] = 0;
                    }
                }
            }
        }
        else
        {
            bestValue = (1<<30);
            for (int i = 0; i < SZ; i++)
            {
                for (int j = 0; j < SZ; j++)
                {
                    if (Board[i][j] == 0)
                    {
                        Board[i][j] = 1;
                        val = MiniMax(Board, 1, new Point(i, j), depth-1);
                        if (bestValue > val)
                           bestValue = val;
                        Board[i][j] = 0;
                    }
                }
            }
        }
        return utl+bestValue;
    }
```

Conclusion:

We have studied various thesis papers and project overviews to implement and create a utility score function/heuristics function which gives sufficiently good results for our purpose. We have finally made our **Advanced Level Unbeatable,** though we have to make a tradeoff and reduce the grid size for searching through more depths.

References:

- Stuart Russell, Peter Norvig, Artificial Intelligence A modern approach 2nd edition.
- L. V. Allis, H. J. van den Herik, Huntjens, Go-Moke and Threat-Space Search.
- http://en.wikipedia.org/wiki/Gomoku
- Further discussion with our professor Afra Zomorodian.