Game Playing**

TIC TAC TOE

Team Members: Venkatesh Bonageri(800964302), Vikas Deshpande(800964852)

ITCS 6150- Intelligent Systems

Introduction:

Games are helpful in improving the physical and mental health of human. Games provide a real source of enjoyment in daily life. Apart from daily life physical games, people also play computer games. These games are different than those of physical games in a sense that they do not involve much physical activity rather mental and emotional activities. Getting games to react back to the user of a game has always been long hard question for game programmers. In this project, we have implemented Tic Tac Toe game using minimax algorithm with alpha beta pruning.

Problem description:

To implement tic tac toe using Minimax algorithm with alpha beta pruning.

This game is very popular and is fairly simple by itself. It is actually a player vs computer game. In this game, there is a board with n x n squares. The goal of Tic-Tac-Toe is to be one of the players to get three same symbols in a row - horizontally, vertically or diagonally - on a n x n grid.

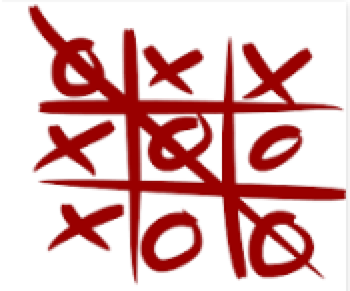


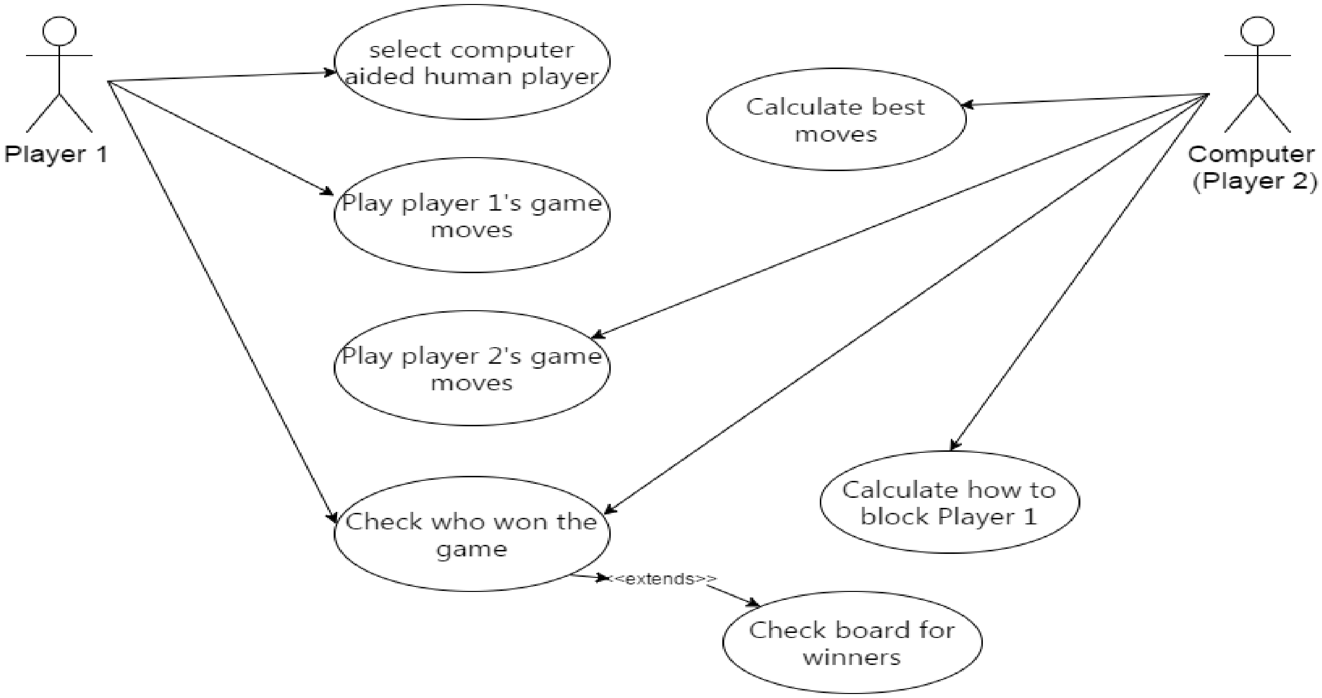
Fig 1: End state of Tic-tac-toe

Objective of the game:

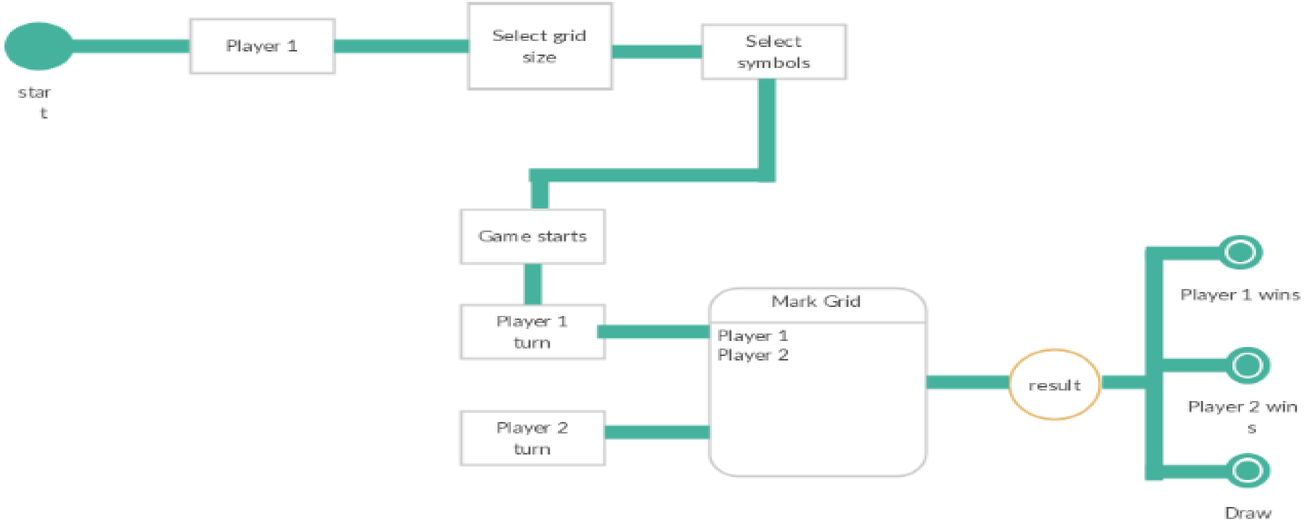
The objective of this game is to be the first to win ‘n’ tic-tac-toe markers lying in a row, column, or diagonal within the greater-square.

Tic-Tac-Toe game is played by two players where the square block (n x n) can be filled with a cross (X) or a circle (O). The game will toggle between the players by giving the chance for each player to mark their move. When one of the players make a combination of n same markers in a horizontal, vertical or diagonal line the program will display which player has won, whether X or O. In this project, we implement a nxn tic-tac-toe game in android. The game is designed so that a player and a computer can play tic-tac-toe using their android device.

Use Case diagram for Tic Tac Toe:


Player 1
select computer aided human player
Play player 1's game moves
Play player 2's game moves
Check who won the game
<<extends>>
Check board for winners
Calculate best moves
Calculate how to block Player 1
Computer (Player 2)

State Diagram for Tic Tac Toe:



Approach used

Our approach to implement Tic Tac Toe is to use minimax algorithm with alpha beta pruning.

Explanation:

- Minimax is recursive algorithm which flips back and forth between the players until a final score is found.
- The player during his turn desires to pick the move with the maximum score. In turn, the scores for each of the available moves are determined by the opposing player deciding which of its available moves has the minimum score. And the scores for the opposing player moves are again determined by the turn-taking player trying to maximize its score and so on all the way down the move tree to an end state.

Formulation:

- States: Any arrangement of "X's" and "O's" on the board of size n*n.
- Initial State: For the chosen n, there will be an empty n*n board.
- Transition State: The state which changes the number of positions available for the players to play i.e decreases the number of positions available to play by '1' (For ex: When Player 'X' plays, he will occupy one position on the board, similarly 'O'). From set of all possible next states, computer (Might be 'X' or 'O') selects the node with the best heuristics.
- Action: Placing "X" and "O" alternatively until the goal state is met.

- Heuristics: An evaluation function for Tic-Tac-Toe is as below

1. For Computer: +(10x) points for each 'x'-in-a-line for computer.

   For Example:

   - +1000 for EACH 3-in-a-line for computer.
   - +10 for EACH 2-in-a-line (with a empty cell) for computer.

2. For Opponent: -(10x) points for each 'x'-in-a-line for computer.

   For Example:

   - Negative scores for opponent, i.e., -1000, -100, -10 for EACH opponent's 3-in-a-line, 2-in-a-line and 1-in-a-line.

3. 0 otherwise i.e., when both X and O in a line
4. 1 for each empty cell

- Goal State: Any possible state where there is n "X's" or "O's" in a line (vertically, horizontally or diagonally) on the board.

Testing Samples (Output)

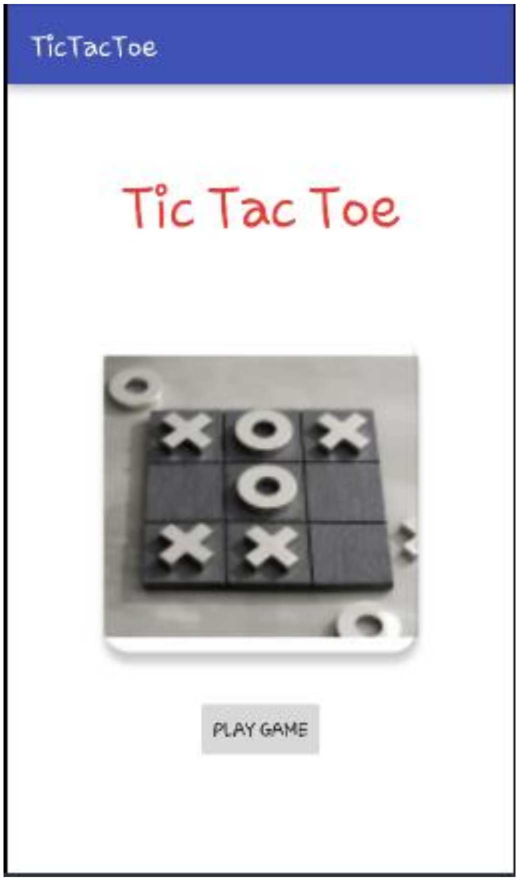


Fig t1: Game’s Welcome Page

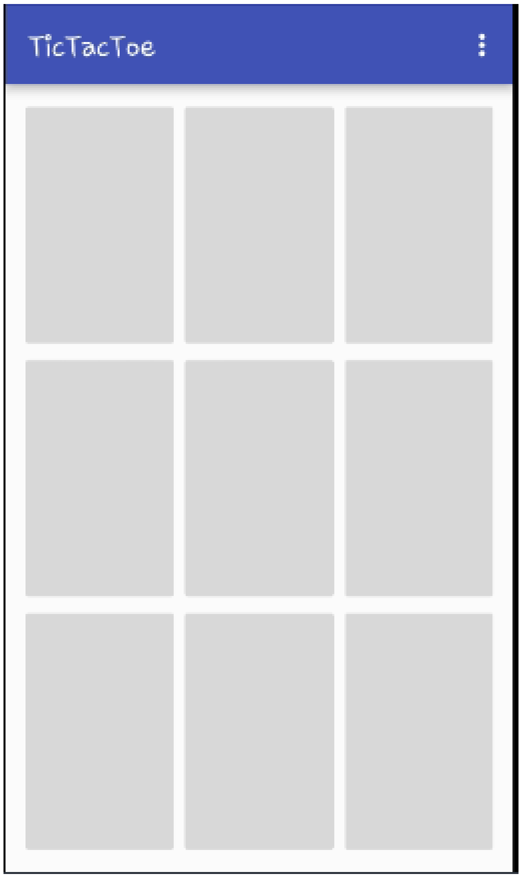


Fig t2: Game’s Main Page (By default the game will be in 3X3 mode)

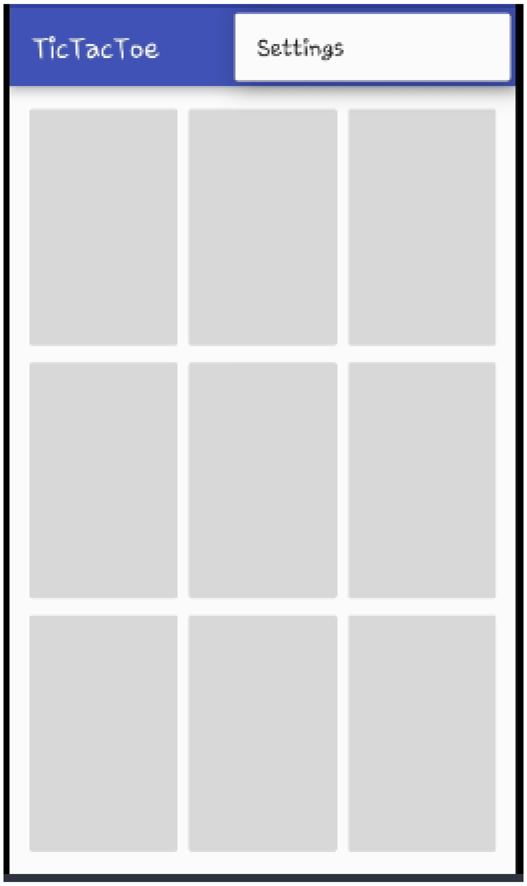
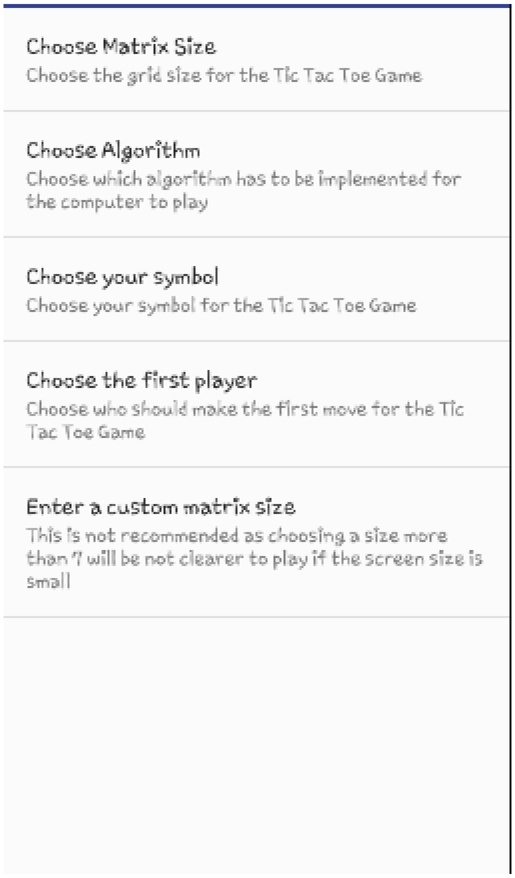


Fig t3: Game’s Main Page (An option to change algorithm used and board size)



Choose Matrix Size

Choose the grid size for the Tic Tac Toe Game

Choose Algorithm

Choose which algorithm has to be implemented for the computer to play

Choose your symbol

Choose your symbol for the Tic Tac Toe Game

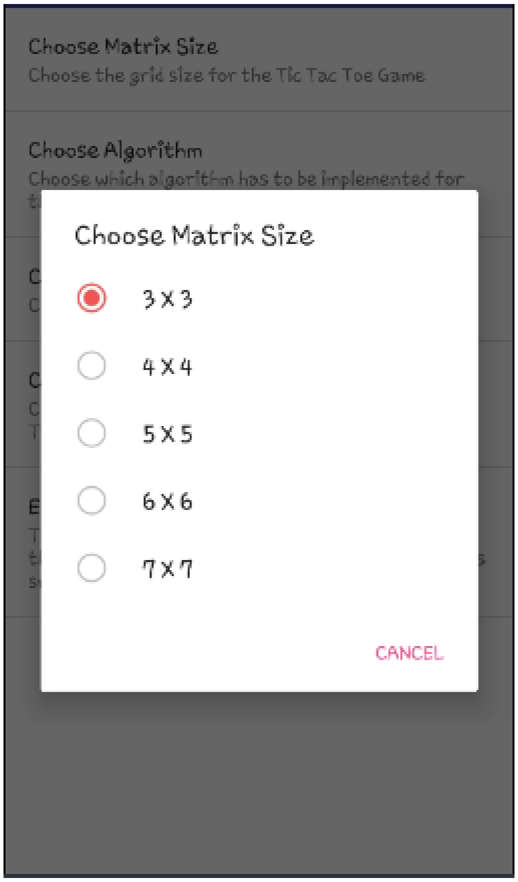
Choose the first player

Choose who should make the first move for the Tic Tac Toe Game

Enter a custom matrix size

This is not recommended as choosing a size more than 7 will be not clearer to play if the screen size is small

Fig t4: Game’s Settings Page (An activity where board size, algorithm chooser, symbol, first player, custom matrix size options are provided)



Choose Matrix Size

Choose the grid size for the Tic Tac Toe Game

Choose Algorithm

Choose which algorithm has to be implemented for

Choose Matrix Size

● 3 X 3

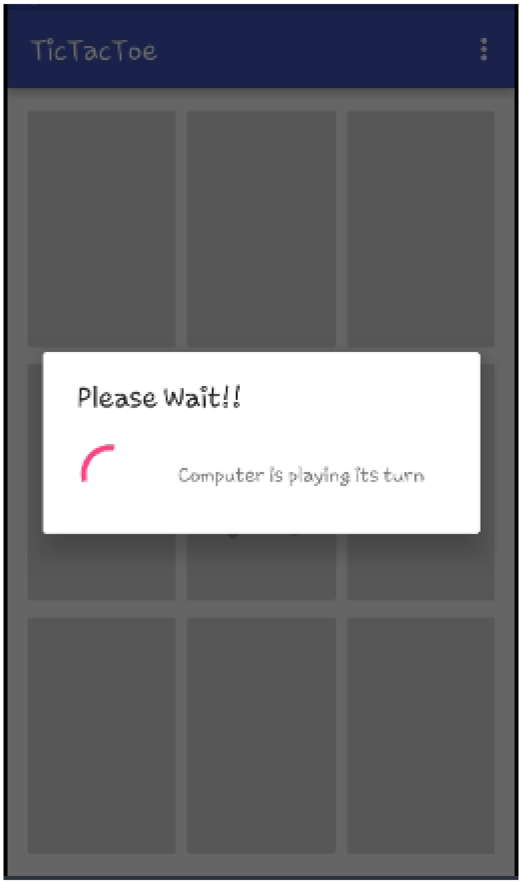
○ 4 X 4

○ 5 X 5

○ 6 X 6

○ 7 X 7

Fig t5: Game’s Board Size Page (Choosing board size)



Please Wait!!

Computer is playing its turn

Fig t6: Computer is playing its turn

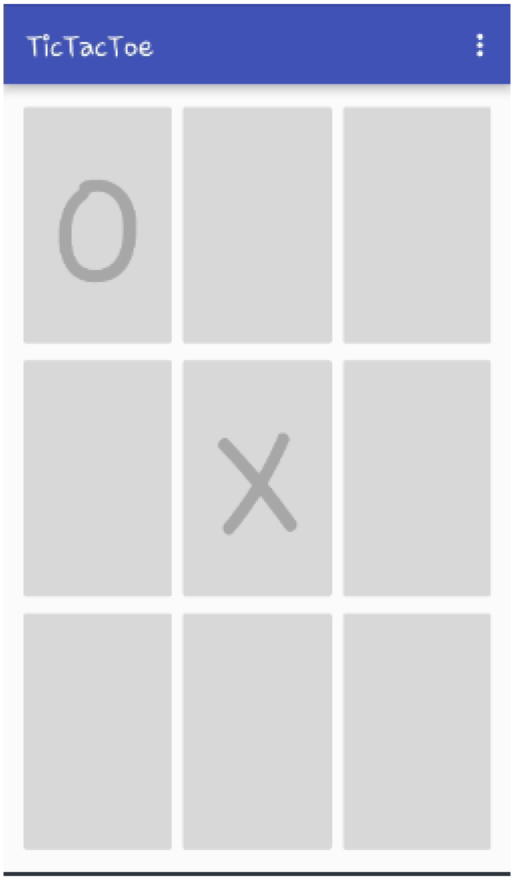
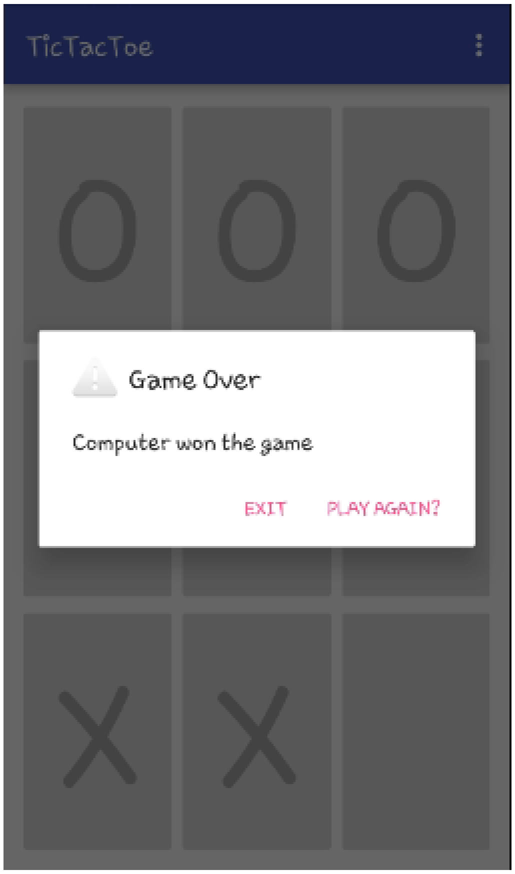


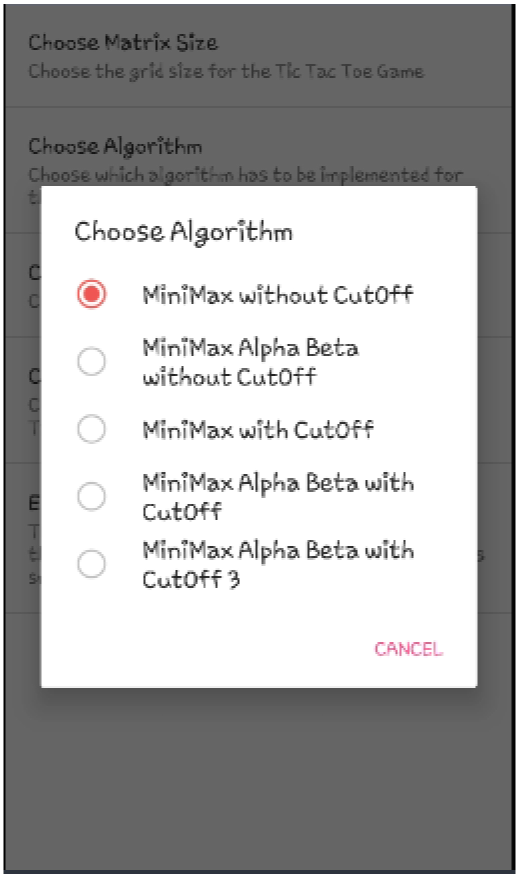
Fig t7: The board looks like



Game Over

Computer won the game

Fig t8: Game’s Board when the goal state is reached



Choose Matrix Size

Choose the grid size for the Tic Tac Toe Game

Choose Algorithm

Choose which algorithm has to be implemented for

Choose Algorithm

● MiniMax without CutOff

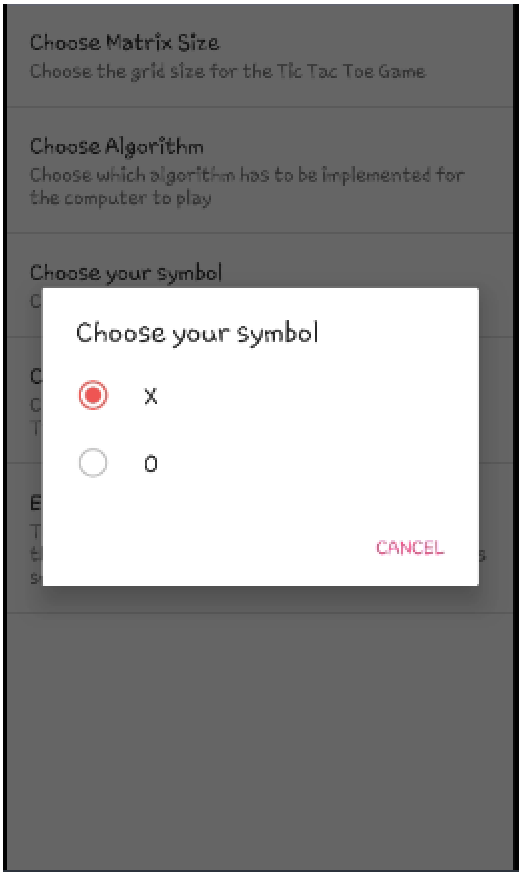
○ MiniMax Alpha Beta without CutOff

○ MiniMax with CutOff

○ MiniMax Alpha Beta with CutOff

○ MiniMax Alpha Beta with CutOff 3

Fig t9: Game's Algorithm Page (Selecting algorithm activity)



Choose Matrix Size

Choose the grid size for the Tic Tac Toe Game

Choose Algorithm

Choose which algorithm has to be implemented for the computer to play

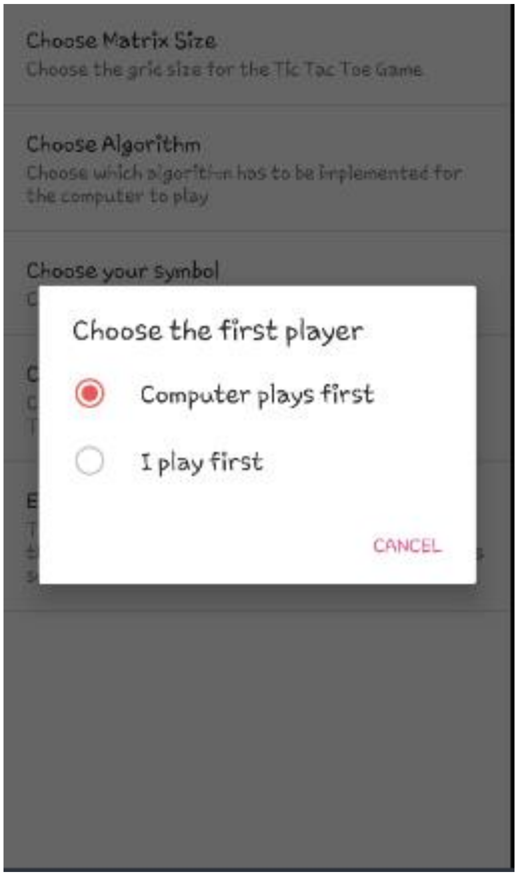
Choose your symbol

Choose your symbol

● X

○ O

Fig t10: Game’s Symbol Chooser Page (Choosing symbol to play)



Choose Matrix Size

Choose the grid size for the Tic Tac Toe Game

Choose Algorithm

Choose which algorithm has to be implemented for the computer to play

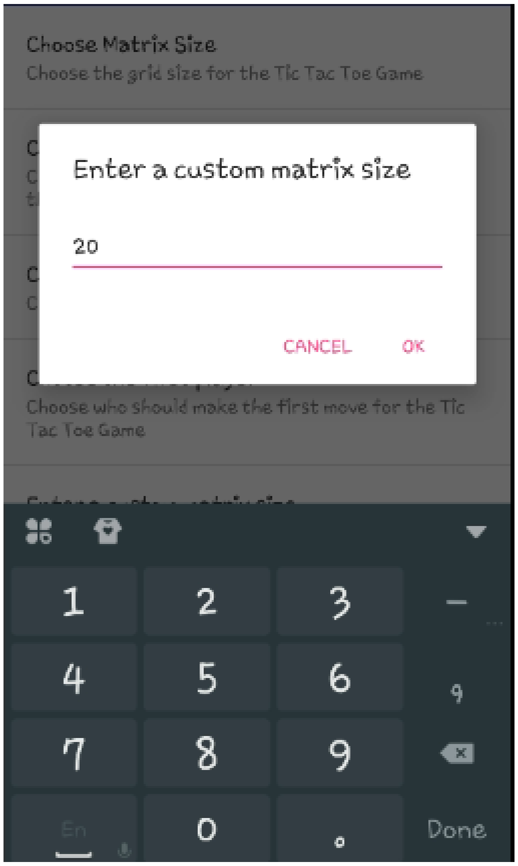
Choose your Symbol

Choose the first player

● Computer plays first

○ I play first

Fig t11: Game’s Options Page (Choosing the first player to play the game)



Choose Matrix Size

Choose the grid size for the Tic Tac Toe Game

Enter a custom matrix size

20

Choose who should make the first move for the Tic Tac Toe Game

Fig t12: Game’s entering custom size Page (Entering custom matrix size page)

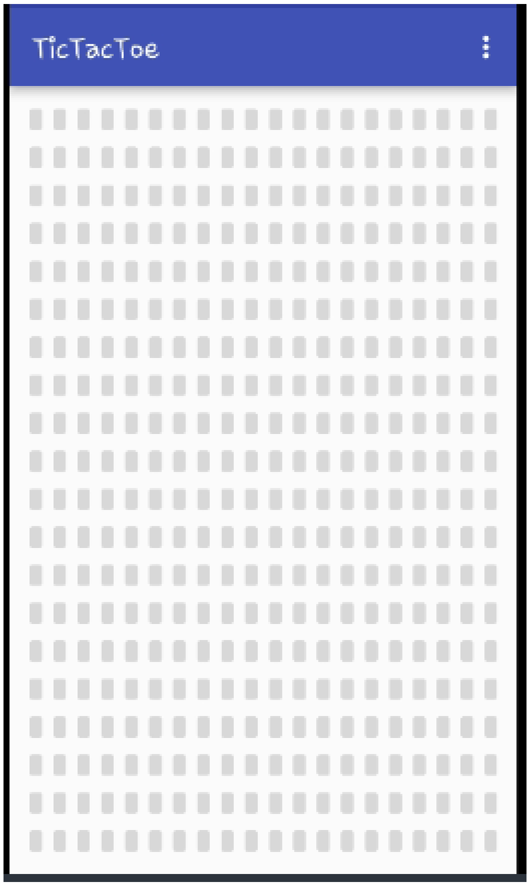


Fig t13: Game’s Board when the size entered is 20

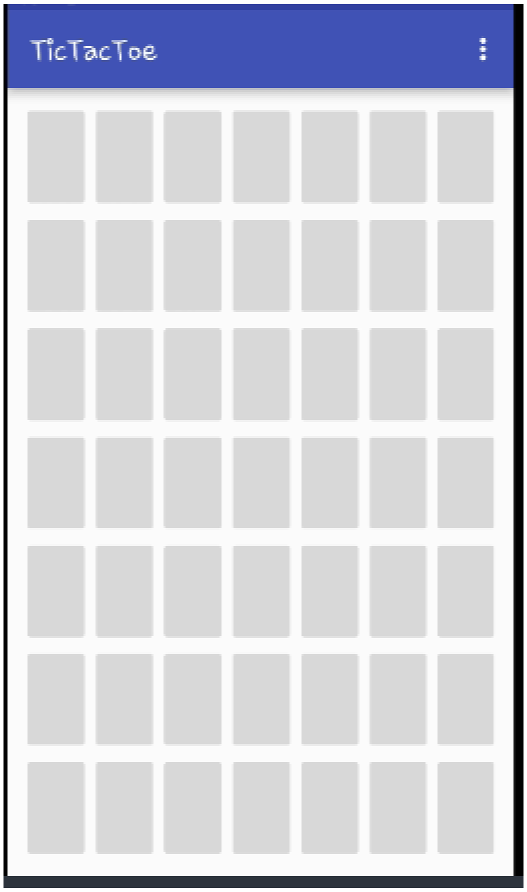


Fig t14: Game’s Board when the size entered is 07

Application Requirements

Android platform with the minimum SDK 17 (android 4.2) and a maximum SDK 23 (android 6.0)

How to run our program?

Since, this is an android application a system with android studio or a mobile phone with android operating system is a must.

In case of android studio, Download and import the project to android studio and click on 'Run' which launches the emulator which in turn opens the application.

In case of having a cell phone with Android operating system, install the apk file and open.

Possible extensions for the project:

The calculations increase if the board size increases, so the future scope of the project would be to implement minimax algorithm in a best possible way, because it is an expensive task to find the best possible move. Boards are valued exponentially.

Outline of the work performed

Time Range	Work by Venkatesh	Work by Vikas
3rd Nov to 10th Nov	Preparation of pseudo code for tic tac toe game using the Min Max Alpha Beta Pruning algorithm for 3x3 board	Designing the UI for the 3X3 board on Android Platform
11th Nov to 20th Nov	Designing the UI for NXN board on Android platform	Implementing the pseudo code developed for the UI designed for 3X3 board
21th Nov to 23th Nov	Testing the 3X3 tic-tac-toe game developed	Preparing the pseudo code for NxN board by extending the pseudo code prepared for 3x3 board to NxN board
24th Nov to 28th Nov	Rectifying the issues identified in the previous testing of 3x3 board tic tac toe game. Implementing the pseudo code developed for the UI designed for NxN board	Rectifying the issues identified in the previous testing of 3x3 board tic tac toe game. Implementing the pseudo code developed for the UI designed for NxN board
29th Nov to 2nd Dec	Testing the NXN tic-tac-toe game developed. Rectifying the issues identified in the previous testing of NxN board tic tac toe game.	Testing the NXN tic-tac-toe game developed. Rectifying the issues identified in the previous testing of NxN board tic tac toe game.

Comparison of results with different algorithms and board sizes

Note: The game is started by ‘Computer’

The table shows the number of nodes generated by the computer to find best possible position for its first three moves.

	Minimax without cutoff	Minimax Alpha beta without cutoff	Minimax with cutoff	Minimax Alpha beta with cutoff	Minimax Alpha Beta with cutoff = 3
3X3	549945-> 7583->173	549945-> 7583->173	3609->259->141	3609->979->141	3609->979->141
4X4	Crossed time bounds	Crossed time bounds	140441->62873->38257	134876->67204->34533	9898->6513->4253
5X5	Crossed time bounds	Crossed time bounds	6693625->4261555->2593941	6693625->4261555->2593941	14425->11155->8421
6X6	Crossed time bounds	Crossed time bounds	281132250->187421500->8519159	281132250->187421500->8519159	93849->75265->58520
7X7	Crossed time bounds	Crossed time bounds	1181554500->9371075000->34076636	1181554500->9371075000->34076636	229587->195866->172156

Program Structure:

The program has the below features

- The user can choose the matrix or grid size. The program works for N*N matrix
- The user can choose who should play first (either computer or the user)
- The user can choose the symbol for his game (either X or O)
- The user can choose the algorithm that the computer should use to get its move

The program is written for a n*n Tic Tac Toe game using the below algorithms

- MiniMax without CutOff
- MiniMax with CutOff N (N is the size of the matrix chosen)
- MiniMax with Alpha Beta Pruning without CutOff
- MiniMax with Alpha Beta Pruning with CutOff N (N is the size of the matrix chosen)
- MiniMax with Alpha Beta Pruning with CutOff 3

Why five algorithms?

Initially, we started off with implementing Minimax algorithm without cutoff and Minimax with alpha beta pruning without cutoff. Until we implemented 3X3 everything was going in a smooth manner. But the tricky part was to implement nXn for which both Minimax and Minimax with alpha beta pruning would seem like you are waiting forever for the computer to make a move. So, we went on to implement both the algorithms considering cutoff into picture. Though the game ended up being a little bit faster than what it was before, the computer was still taking lot of time to make a move. So, we decided to implement Minimax with alpha beta pruning with cutoff set to 3 (This algorithm has its own disadvantages as well. For example, in a board of size 4X4 and when the depth is 3 the computer might end up losing sometime which is not factually correct according to the algorithm.).

The important files of the Program are as below

- MainActivity.java
- NodeMiniMaxNoDepth.java
- NodeMiniMaxWithDepth.java
- NodeAlphaBetaNoDepth.java
- NodeAlphaBetaWithDepth.java
- NodeAlphaBetaWithDepth3.java
- SettingsPreference.java
- WelcomePage.java
- activity_main.xml
- activity_welcome_page.xml
- menu.xml
- preferences.xml

Global Variables:

- **public static** Seed *myseed*, *oppseed*; //variables to differentiate between computer and the user as two players
- **public static int** *size*=3; //Size of the matrix choosen. By default it will be 3

Functions

- Mainactivity.java

This file plots the grid in a separate thread as per the user's selection of the grid size and also handles the user's input and calls the appropriate functions to determine the computer's move and places the symbol accordingly and also calls the appropriate functions to check if it has reached the end

- ❑ initializewinmatrix()--> assigns the symbols to the user and the computer
- ❑ onClick() --> Places the user symbol on the location of the grid chosen by the user
- ❑ playCompTurn → Calls the appropriate functions based on the user's algorithm selection to determine the computer's next move

- NodeMiniMaxNoDepth.java

This file handles the MiniMax algorithm with Goal state as the leaf node and generates the computer's next move and returns the value back to the mainActivity.java

- ❏ Constructor() → creates matrix of size of N*N where n is the size specified by the user. Assigns the '\0' as the value to the matrix
- ❏ place() → places X in the matrix to identify as the computer's move and places O in the matrix to identify as the player's move
- ❏ getCompNextMove() → Checks if the solution state has been reached else calls the minimax function to get the computer's next move
- ❏ minimax() → it's a recursive function which implements the minimax algorithm
- ❏ generateMoves() → generates the next set of available moves that the computer can play.
- ❏ evaluate() → It is called by the minimax function. Evaluates each move or the state of the matrix and returns the evaluated value back to the minimax function.
- ❏ evaluateeachdiag1() → This function is called by the evaluate() function. This function evaluates the primary diagonal of the matrix and returns the value back to the evaluate function.
- ❏ evaluateeachdiag2() → This function is called by the evaluate() function. This function evaluates the secondary diagonal of the matrix and returns the value back to the evaluate function.
- ❏ evaluateeachrow() → This function is called by the evaluate() function. This function evaluates the each row of the matrix and returns the value back to the evaluate function.
- ❏ evaluateeachcol() → This function is called by the evaluate() function. This function evaluates the each column of the matrix and returns the value back to the evaluate function.
- ❏ checkend() → This function checks the matrix to see if the goal state has been reached. Like Computer Won or Player Won or Game Drawn or still the game on.

- NodeMiniMaxWithDepth.java

This file handles the MiniMax algorithm with cutoff of N steps as the leaf node where N is the size of the matrix and generates the computer's next move and returns the value back to the mainActivity.java

- ❏ Constructor() → creates matrix of size of N*N where n is the size specified by the user. Assigns the '\0' as the value to the matrix
- ❏ place() → places X in the matrix to identify as the computer's move and places O in the matrix to identify as the player's move
- ❏ getCompNextMove() → Checks if the solution state has been reached else calls the minimax function to get the computer's next move
- ❏ minimax() → it's a recursive function which implements the minimax algorithm
- ❏ generateMoves() → generates the next set of available moves that the computer can play.
- ❏ evaluate() → It is called by the minimax function. Evaluates each move or the state of the matrix and returns the evaluated value back to the minimax function.

- ❑ evaluateeachdiag1() → This function is called by the evaluate() function. This function evaluates the primary diagonal of the matrix and returns the value back to the evaluate function.
- ❑ evaluateeachdiag2() → This function is called by the evaluate() function. This function evaluates the secondary diagonal of the matrix and returns the value back to the evaluate function.
- ❑ evaluateeachrow() → This function is called by the evaluate() function. This function evaluates the each row of the matrix and returns the value back to the evaluate function.
- ❑ evaluateeachcol() → This function is called by the evaluate() function. This function evaluates the each column of the matrix and returns the value back to the evaluate function.
- ❑ checkend() → This function checks the matrix to see if the goal state has been reached. Like Computer Won or Player Won or Game Drawn or still the game on.

- NodeAlphaBetaNoDepth.java

This file handles the MiniMax algorithm with AlphaBeta pruning with Goal state as the leaf node and generates the computer's next move and returns the value back to the mainActivity.java

- ❑ Constructor() → creates matrix of size of N*N where n is the size specified by the user. Assigns the '\0' as the value to the matrix
- ❑ place() → places X in the matrix to identify as the computer's move and places O in the matrix to identify as the player's move
- ❑ getCompNextMove() → Checks if the solution state has been reached else calls the minimax function to get the computer's next move
- ❑ minimax() → it's a recursive function which implements the minimax algorithm with alpha beta pruning
- ❑ generateMoves() → generates the next set of available moves that the computer can play.
- ❑ evaluate() → It is called by the minimax function. Evaluates each move or the state of the matrix and returns the evaluated value back to the minimax function.
- ❑ evaluateeachdiag1() → This function is called by the evaluate() function. This function evaluates the primary diagonal of the matrix and returns the value back to the evaluate function.
- ❑ evaluateeachdiag2() → This function is called by the evaluate() function. This function evaluates the secondary diagonal of the matrix and returns the value back to the evaluate function.
- ❑ evaluateeachrow() → This function is called by the evaluate() function. This function evaluates the each row of the matrix and returns the value back to the evaluate function.
- ❑ evaluateeachcol() → This function is called by the evaluate() function. This function evaluates the each column of the matrix and returns the value back to the evaluate function.
- ❑ checkend() → This function checks the matrix to see if the goal state has been reached. Like Computer Won or Player Won or Game Drawn or still the game on.

- NodeAlphaBetaWithDepth.java

This file handles the MiniMax algorithm with AlphaBeta pruning with cutoff of N steps as the leaf node where N is the size of the matrix and generates the computer's next move and returns the value back to the mainActivity.java

- ❑ Constructor() → creates matrix of size of N*N where n is the size specified by the user. Assigns the '\0' as the value to the matrix
- ❑ place() → places X in the matrix to identify as the computer's move and places O in the matrix to identify as the player's move
- ❑ getCompNextMove() → Checks if the solution state has been reached else calls the minimax function to get the computer's next move
- ❑ minimax() → it's a recursive function which implements the minimax algorithm with alpha beta pruning
- ❑ generateMoves() → generates the next set of available moves that the computer can play.
- ❑ evaluate() → It is called by the minimax function. Evaluates each move or the state of the matrix and returns the evaluated value back to the minimax function.
- ❑ evaluateeachdiag1() → This function is called by the evaluate() function. This function evaluates the primary diagonal of the matrix and returns the value back to the evaluate function.
- ❑ evaluateeachdiag2() → This function is called by the evaluate() function. This function evaluates the secondary diagonal of the matrix and returns the value back to the evaluate function.
- ❑ evaluateeachrow() → This function is called by the evaluate() function. This function evaluates the each row of the matrix and returns the value back to the evaluate function.
- ❑ evaluateeachcol() → This function is called by the evaluate() function. This function evaluates the each column of the matrix and returns the value back to the evaluate function.
- ❑ checkend() → This function checks the matrix to see if the goal state has been reached. Like Computer Won or Player Won or Game Drawn or still the game on.

- NodeAlphaBetaWithDepth3.java

This file handles the MiniMax algorithm with AlphaBeta pruning with cutoff of 3 steps as the leaf node and generates the computer's next move and returns the value back to the mainActivity.java

- ❑ Constructor() → creates matrix of size of N*N where n is the size specified by the user. Assigns the '\0' as the value to the matrix
- ❑ place() → places X in the matrix to identify as the computer's move and places O in the matrix to identify as the player's move
- ❑ getCompNextMove() → Checks if the solution state has been reached else calls the minimax function to get the computer's next move

- minimax() → it’s a recursive function which implements the minimax algorithm with alpha beta pruning
- generateMoves() → generates the next set of available moves that the computer can play.
- evaluate() → It is called by the minimax function. Evaluates each move or the state of the matrix and returns the evaluated value back to the minimax function.
- evaluateeachdiag1() → This function is called by the evaluate() function. This function evaluates the primary diagonal of the matrix and returns the value back to the evaluate function.
- evaluateeachdiag2() → This function is called by the evaluate() function. This function evaluates the secondary diagonal of the matrix and returns the value back to the evaluate function.
- evaluateeachrow() → This function is called by the evaluate() function. This function evaluates the each row of the matrix and returns the value back to the evaluate function.
- evaluateeachcol() → This function is called by the evaluate() function. This function evaluates the each column of the matrix and returns the value back to the evaluate function.
- checkend() → This function checks the matrix to see if the goal state has been reached. Like Computer Won or Player Won or Game Drawn or still the game on.

- SettingsPreference.java

This page handles the user’s selection like the user’s symbol, algorithm, player to play first and the sizeof the matrix

- WelcomePage.java
  Initial welcome page which can start the Mainactivity.java to start the game
- activity_main.xml
  UI page for the mainacitvity.java.
- activity_welcome_page.xml
  UI page for the Welcome page
- preferences.xml
  UI for the preferences.

The Source code of the program is as below

- **MainActivity.java**

```
package com.example.venkatesh.istictactoe;
import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Handler;
import android.os.Message;
import android.preference.PreferenceManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.util.TypedValue;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
public class MainActivity extends AppCompatActivity implements View.OnClickListener, Runnable{
  int idcounter=12589;
  public static int size=3;
  String playersymbol = "O";
  String compsymbol = "X";
  int algochooser=1;
  ProgressDialog progress;
  Handler handler;
  static int countnodes=0;
  NodeAlphaBetaWithDepth mygame1;
  NodeAlphaBetaNoDepth mygame2;
  NodeMiniMaxWithDepth mygame3;
```

```
NodeMiniMaxNoDepth mygame4;
NodeAlphaBetaWithDepth3 mygame5;
static Context maincontext;
Button btn[][];
public static enum Seed {Cross, Round, Empty}
public static Seed myseed, oppseed;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    maincontext = getApplicationContext();
    progress = new ProgressDialog(this);
    progress.setTitle("Please Wait!!");
    progress.setMessage("Computer is playing its turn");
    progress.setCancelable(true);
    progress.setProgressStyle(ProgressDialog.STYLE_SPINNER);
    SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences(this);
    String algoType = pref.getString("algoPreference","");
    String matrixSize = pref.getString("matrixPreference","");
    String symboltype = pref.getString("symbolPreference","");
    int custommatrxsize = 0;
    try{
        custommatrxsize = Integer.parseInt(pref.getString("custommatrixsize",""));
        if(custommatrxsize<3)
            custommatrxsize=3;
    }catch (Exception e){
        custommatrxsize = 3;
    }
    size = custommatrxsize;
    if(symboltype.equals("O"))
    {
        playersymbol="O";
        compsymbol="X";
    }else if (symboltype.equals("X"))
```

```
{
    playersymbol="X";
    compsymbol="O";
}
LinearLayout root = (LinearLayout) findViewById(R.id.root);
root.setWeightSum(size);
if(algoType.equals("AD"))
{
    algochooser=1;
    mygame1 = new NodeAlphaBetaWithDepth();
}
else if(algoType.equals("AN"))
{
    algochooser=2;
    mygame2 = new NodeAlphaBetaNoDepth();
}
else if(algoType.equals("MD"))
{
    algochooser=3;
    mygame3 = new NodeMiniMaxWithDepth();
}
else if(algoType.equals("MN"))
{
    algochooser=4;
    mygame4 = new NodeMiniMaxNoDepth();
}
else if(algoType.equals("ADD"))
{
    algochooser=5;
    mygame5 = new NodeAlphaBetaWithDepth3();
}
int duplicateidkeeper=idcounter;
for(int i=0;i<size;i++)
{
        LinearLayout eachrow = new LinearLayout(this);
        eachrow.setOrientation(LinearLayout.HORIZONTAL);
        LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.MATCH_PARENT,1.0f);
        eachrow.setLayoutParams(params);
        eachrow.setWeightSum(size);

        for(int j=0;j<size;j++)
        {
            Button eachButton = new Button(this);
            LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.MATCH_PARENT,1.0f);
            eachButton.setTextSize(TypedValue.COMPLEX_UNIT_SP, 990/(size*size));
            eachButton.setId(duplicateidkeeper++);
            eachButton.setOnClickListener(this);
            eachButton.setTag((i*10)+j);
            eachrow.addView(eachButton,lp);
        }
        root.addView(eachrow);
    }

    initializewinmatrix();
    btn = new Button[MainActivity.size][MainActivity.size];
    duplicateidkeeper=idcounter;
    for (int i=0;i<size;i++)
    {
        for (int j=0;j<size;j++)
        {
            btn[i][j] = (Button) findViewById(duplicateidkeeper++);
        }
    }
    final String playfirst = pref.getString("playfirstPreference","");
    handler = new Handler(new Handler.Callback() {
        @Override
    public boolean handleMessage(Message msg) {
        switch(msg.what) {
            case 1:
                btn[msg.getData().getInt("row")][msg.getData().getInt("col")].setText(compsymbol);
                Log.d("computernodes ",MainActivity.countnodes+"");
                btn[msg.getData().getInt("row")][msg.getData().getInt("col")].setEnabled(false);
                progress.dismiss();
                break;

            case 2:progress.dismiss();
                new AlertDialog.Builder(MainActivity.this)
                        .setTitle("Game Over")
                        .setMessage(msg.getData().getString("tempwinner"))
                        .setCancelable(false)
                        .setIcon(android.R.drawable.ic_dialog_alert)
                        .setPositiveButton("Play Again?", new DialogInterface.OnClickListener() {

                            public void onClick(DialogInterface dialog, int whichButton) {
                                Intent i = new Intent(getApplicationContext(),MainActivity.class);
                                finish();
                                startActivity(i);

                            }})
                        .setNegativeButton("Exit", new DialogInterface.OnClickListener(){
                            public void onClick(DialogInterface dialog, int whichButton) {
                                finish();
                            }}).show();
                break;
        }
        return false;
    }
});

if(playfirst.equals("computer"))
```

```
    {
        progress.show();
        Thread t1=new Thread(this);
        t1.start();
    }
}

//assigns the symbols to the user and the computer
public void initializewinmatrix()
{
    myseed=Seed.Cross;
    oppseed=Seed.Round;
}

//Places the user symbol on the location of the grid chosen by the user
@Override
public void onClick(View v) {
    Button temp = (Button) findViewById(v.getId());
    temp.setText(playersymbol);
    temp.setEnabled(false);
    int temprowcol = (Integer) v.getTag();
    int row = temprowcol/10;
    int col = temprowcol%10;
    if(algochooser==1)
    {
        mygame1.place(row,col,oppseed);
    }else if(algochooser==2)
    {
        mygame2.place(row,col,oppseed);

    }else if(algochooser==3)
    {
        mygame3.place(row,col,oppseed);
    }else if(algochooser==4)
    {
        mygame4.place(row,col,oppseed);

    }else if(algochooser==5)
    {
        mygame5.place(row,col,oppseed);

    }



    progress.show();
    Thread t1=new Thread(this);
    t1.start();
}

//Calls the appropriate functions based on the user's algorithm selection to determine the computer's next move
void playCompTurn()
{
    MainActivity.countnodes=0;
    int rowcol[]=new int[3];
    if(algochooser==1)
    {
        rowcol = mygame1.getCompNextMove();
    }else if(algochooser==2)
    {
        rowcol = mygame2.getCompNextMove();

    }else if(algochooser==3)
    {
        rowcol = mygame3.getCompNextMove();

    }else if(algochooser==4)
    {
```

```
    rowcol = mygame4.getCompNextMove();
}else if(algochooser==5)
{
    rowcol = mygame5.getCompNextMove();
}



if(rowcol[0]==rowcol[1] && rowcol[0]==-1)
{
    String tempwinner;
    if(rowcol[2]==1)
        tempwinner="Computer won the game";
    else if(rowcol[2]==2)
        tempwinner="You won the game";
    else
        tempwinner="Game Drawn";
    Message msg = new Message();
    msg.what = 2;
    Bundle data = new Bundle();
    data.putString("tempwinner",tempwinner);
    msg.setData(data);
    handler.sendMessage(msg);
}
else {
    int tempresult=0;
    if(algochooser==1)
    {
        mygame1.place(rowcol[0],rowcol[1],myseed);
        tempresult = mygame1.checkend();
    }else if(algochooser==2)
    {
        mygame2.place(rowcol[0],rowcol[1],myseed);
        tempresult = mygame2.checkend();
```

```
}else if(algochooser==3)
{
    mygame3.place(rowcol[0],rowcol[1],myseed);
    tempresult = mygame3.checkend();

}else if(algochooser==4)
{
    mygame4.place(rowcol[0],rowcol[1],myseed);
    tempresult = mygame4.checkend();
}else if(algochooser==5)
{
    mygame5.place(rowcol[0],rowcol[1],myseed);
    tempresult = mygame5.checkend();
}
Message msg = new Message();
msg.what = 1;
Bundle data = new Bundle();
data.putInt("row",rowcol[0]);
data.putInt("col",rowcol[1]);
msg.setData(data);
handler.sendMessage(msg);
if(tempresult>0)
{
    String tempwinner;
    if(tempresult==1)
        tempwinner="Computer won the game";
    else if(tempresult==2)
        tempwinner="You won the game";
    else
        tempwinner="Game Drawn";
    Message msg1 = new Message();
    msg1.what = 2;
    Bundle data1 = new Bundle();
    data1.putString("tempwinner",tempwinner);
```

```
                msg1.setData(data1);
                handler.sendMessage(msg1);
            }
        }
    }
    @Override
    public void run() {
        playCompTurn();
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu, menu);
        return true;
    }
    @Override
    public MenuInflater getMenuInflater() {
        return super.getMenuInflater();
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        try{
            int id = item.getItemId();
            if(id == R.id.settings){
                Intent i = new Intent(this, SettingsPreference.class);
                finish();
                startActivity(i);
            }
        }catch (Exception e)
        {
            Log.d("debug2",e.getMessage());
        }
        return super.onOptionsItemSelected(item);
    }
}
```

- **NodeMiniMaxNoDepth.java**

```
package com.example.venkatesh.istictactoe;
import java.util.ArrayList;
import java.util.List;
/**
* Created by Venkatesh on 11/12/2016.
*/
public class NodeMiniMaxNoDepth {
    char [][] matrix;
    //creates matrix of size of N*N where n is the size specified by the user. Assigns the '\0' as the value to the matrix
    public NodeMiniMaxNoDepth() {
        this.matrix = new char[MainActivity.size][MainActivity.size];
        for(int i=0;i<MainActivity.size;i++)
        {
            for (int j=0;j<MainActivity.size;j++)
            {
                matrix[i][j]='\0';
            }
        }
    }
    //places X in the matrix to identify as the computer's move and places O in the matrix to identify as the player's move
    public void place(int row, int col,MainActivity.Seed x)
    {
        if(x==MainActivity.myseed)
            matrix[row][col]='X';
        else
            matrix[row][col]='O';
    }

    //Checks if the solution state has been reached else calls the minimax function to get the computer's next move
    int[] getCompNextMove()
    {
        int ifend = checkend();
        if(ifend>0)
    {
        return new int[] {-1,-1,ifend};
    }
    else
    {
        int[] temp = new int[2];
        temp = minimax(MainActivity.myseed);
        return new int[] {temp[1],temp[2],0};
    }
}
//it's a recursive function which implements the minimax algorithm
private int[] minimax(MainActivity.Seed player) {
    // Generate possible next moves in a List of int[2] of {row, col}.
    List<int[]> nextMoves = generateMoves();
    // mySeed is maximizing; while oppSeed is minimizing
    int bestScore = (player == MainActivity.myseed) ? Integer.MIN_VALUE : Integer.MAX_VALUE;
    int currentScore;
    int bestRow = -1;
    int bestCol = -1;
    if (nextMoves.isEmpty()) {
        // Gameover , evaluate score
        bestScore = evaluate();
    } else {
        for (int[] move : nextMoves) {
            // Try this move for the current "player"
            MainActivity.countnodes++;
            if(player==MainActivity.Seed.Cross)
                matrix[move[0]][move[1]]='X';
            else
                matrix[move[0]][move[1]]='O';
            if (player == MainActivity.myseed) {  // mySeed (computer) is maximizing player
                currentScore = minimax(MainActivity.oppseed)[0];
                if (currentScore > bestScore) {
                    bestScore = currentScore;
```

```java
                    bestRow = move[0];
                    bestCol = move[1];
                }
            } else {  // oppSeed is minimizing player
                currentScore = minimax(MainActivity.myseed)[0];
                if (currentScore < bestScore) {
                    bestScore = currentScore;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            }
            // Undo move
            matrix[move[0]][move[1]] = '\0';
        }
    }
    return new int[] {bestScore, bestRow, bestCol};
}
//generates the next set of available moves that the computer can play.
private List<int[]> generateMoves() {
    List<int[]> nextMoves = new ArrayList<int[]>(); // allocate List
    // If gameover, i.e., no next move
    if (checkend()>0) {
        return nextMoves;   // return empty list
    }
    // Search for empty cells and add to the List
    for (int row = 0; row < MainActivity.size; ++row) {
        for (int col = 0; col < MainActivity.size; ++col) {
            if (matrix[row][col] == '\0') {
                nextMoves.add(new int[] {row, col});
            }
        }
    }
    return nextMoves;
}
```

```
//It is called by the minimax function. Evaluates each move or the state of the matrix and returns the evaluated value back to the minimax function.

private int evaluate() {

    int score = 0;

    for(int i=0;i<MainActivity.size;i++)

    {

        score=score+evaluateeachrow(i);

    }

    for(int i=0;i<MainActivity.size;i++)

    {

        score=score+evaluateeachcol(i);

    }

    score = score + evaluateeachdiag1();

    score =  score + evaluateeachdiag2();

    return score;

}

//This function is called by the evaluate() function. This function evaluates the primary diagonal of the matrix and returns the value back to the evaluate function.

public int evaluateeachdiag1()

{

    int score=0;

    for (int i=0;i<MainActivity.size;i++)

    {

        for (int j=0;j<MainActivity.size;j++)

        {

            if(i==j)

            {

                if(matrix[i][j]=='X')

                {

                    if(score>0)

                    {

                        score=score*10;

                    }

                    else if(score<0)
```

```
                        {
                            return 0;
                        }
                        else
                        {
                            score=10;
                        }
                    }
                    else if(matrix[i][j]=='O')
                    {
                        if(score<0)
                        {
                            score=score*10;
                        }
                        else if(score>0)
                        {
                            return 0;
                        }
                        else
                        {
                            score=-10;
                        }
                    }
                }
            }
        }
        return score;
    }
    //This function is called by the evaluate() function. This function evaluates the secondary diagonal of the matrix and returns the value back to the evaluate function.
    public int evaluateeachdiag2()
    {
        int score=0;
        for (int i=0;i<MainActivity.size;i++)
```

```
{
    for (int j=0;j<MainActivity.size;j++)
    {
        if(i+j==MainActivity.size-1)
        {
            if(matrix[i][j]=='X')
            {
                if(score>0)
                {
                    score=score*10;
                }
                else if(score<0)
                {
                    return 0;
                }
                else
                {
                    score=10;
                }
            }
            else if(matrix[i][j]=='O')
            {
                if(score<0)
                {
                    score=score*10;
                }
                else if(score>0)
                {
                    return 0;
                }
                else
                {
                    score=-10;
                }
                }
            }
        }
    }
    return score;
}
//This function is called by the evaluate() function. This function evaluates the each row of the matrix and returns the value back to the evaluate function.
public int evaluateeachrow(int i)
{
    int score=0;
    int emptycells=0;
    for(int j=0;j<MainActivity.size;j++)
    {
        if(matrix[i][j]=='X'){
            if(score>0)
            {
                    score=score*10;
            }
            else if(score<0)
            {
                return 0;
            }
            else {
                score = 10;
            }
        }
        else if(matrix[i][j]=='O')
        {
            if(score<0)
            {
                score=score*10;
            }
            else if(score>0)
```

```
                {
                    return 0;
                }
                else {
                    score=-10;
                }
            }
            else if(matrix[i][j]=='\0')
            {
                emptycells++;
            }
        }
        return score+emptycells;
    }
    //This function is called by the evaluate() function. This function evaluates the each column of the matrix and returns the value back to the evaluate function.
    public int evaluateeachcol(int j)
    {
        int score=0;
        int emptycells=0;
        for(int i=0;i<MainActivity.size;i++)
        {
            if(matrix[i][j]=='X'){
                if(score>0)
                {
                    score=score*10;
                }
                else if(score<0)
                {
                    return 0;
                }
                else {
                    score = 10;
                }
            }
            else if(matrix[i][j]=='O')
            {
                if(score<0)
                {
                    score=score*10;
                }
                else if(score>0)
                {
                    return 0;
                }
                else {
                    score=-10;
                }
            }
            else if(matrix[i][j]=='\0')
            {
                emptycells++;
            }
        }
        return score;
    }
    //This function checks the matrix to see if the goal state has been reached. Like Computer Won or Player Won or Game Drawn or still the game on.
    public int checkend()
    {
        int count=0;

        int [] winrow= new int[MainActivity.size];
        int [] wincol= new int[MainActivity.size];
        int  windiag1=0;
        int  windiag2=0;
        int foundwin=0;
        int checkdraw=0;
```

```
for(int i=0;i<MainActivity.size;i++)
{
    for(int j=0;j<MainActivity.size;j++)
    {
        if(matrix[i][j]=='X')
        {
            checkdraw++;
            winrow[i]++;
            wincol[j]++;
            if(i==j)
                windiag1++;
            if((i+j)==MainActivity.size-1)
                windiag2++;
        }
        else if(matrix[i][j]=='O')
        {
            checkdraw++;
            winrow[i]--;
            wincol[j]--;
            if(i==j)
                windiag1--;
            if((i+j)==MainActivity.size-1)
                windiag2--;
        }
    }
}
for(int i=0; i<MainActivity.size && (foundwin!=1 || foundwin!=2);i++ )
{
    if(winrow[i]==MainActivity.size)
    {
        foundwin = 1;
        break;
    }
    else if(winrow[i]==(-1*MainActivity.size))
```

```
            {
                foundwin = 2;
                break;
            }
            else if(wincol[i]==MainActivity.size)
            {
                foundwin = 1;
                break;
            }
            else if(wincol[i]==(-1*MainActivity.size))
            {
                foundwin = 2;
                break
            }
        }
        if(windiag1==MainActivity.size || windiag2==MainActivity.size)
        {
            foundwin = 1;
        }
        else if(windiag1==(-1*MainActivity.size) || windiag2==(-1*MainActivity.size))
        {
            foundwin = 2;
        }
        if((foundwin!=1 || foundwin!=2)&& checkdraw==(MainActivity.size*MainActivity.size))
            return 3;
        return foundwin;
    }
}
```

- **NodeMiniMaxWithDepth.java**

```
package com.example.venkatesh.istictactoe;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
* Created by Venkatesh on 11/12/2016.
*/
public class NodeMiniMaxWithDepth {
  char [][] matrix;
  //creates matrix of size of N*N where n is the size specified by the user. Assigns the ‘\0’ as the value to the matrix
  public NodeMiniMaxWithDepth() {
     this.matrix = new char[MainActivity.size][MainActivity.size];
     for(int i=0;i<MainActivity.size;i++)
     {
        for (int j=0;j<MainActivity.size;j++)
        {
           matrix[i][j]='\0';
        }
     }
  }
  //places X in the matrix to identify as the computer’s move and places O in the matrix to identify as the player’s move
  public void place(int row, int col,MainActivity.Seed x)
  {
     if(x==MainActivity.myseed)
        matrix[row][col]='X';
     else
        matrix[row][col]='O';
  }
  //Checks if the solution state has been reached else calls the minimax function to get the computer’s next move
  int[] getCompNextMove()
  {
     int ifend = checkend();
```

```
    if(ifend>0)

    {

        return new int[] {-1,-1,ifend};

    }

    else

    {

        int[] temp = new int[2];

        if(MainActivity.size==3)

            temp = minimax(MainActivity.size+1,MainActivity.myseed);

        else

            temp = minimax(MainActivity.size,MainActivity.myseed);

        return new int[] {temp[1],temp[2],0};

    }

}

//minimax() → it's a recursive function which implements the minimax algorithm

private int[] minimax(int depth,MainActivity.Seed player) {

    // Generate possible next moves in a List of int[2] of {row, col}.

    List<int[]> nextMoves = generateMoves();

    // mySeed is maximizing; while oppSeed is minimizing

    int bestScore = (player == MainActivity.myseed) ? Integer.MIN_VALUE : Integer.MAX_VALUE;

    int currentScore;

    int bestRow = -1;

    int bestCol = -1;

    if (nextMoves.isEmpty() || depth==0) {

        // Gameover or depth reached, evaluate score

        bestScore = evaluate();

    } else {

        for (int[] move : nextMoves) {

            // Try this move for the current "player"

            MainActivity.countnodes++;

            if(player==MainActivity.Seed.Cross)

                matrix[move[0]][move[1]]='X';

            else

                matrix[move[0]][move[1]]='O';
            if (player == MainActivity.myseed) {  // mySeed (computer) is maximizing player
                currentScore = minimax(depth-1,MainActivity.oppseed)[0];
                if (currentScore > bestScore) {
                    bestScore = currentScore;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            } else {  // oppSeed is minimizing player
                currentScore = minimax(depth-1,MainActivity.myseed)[0];
                if (currentScore < bestScore) {
                    bestScore = currentScore;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            }
            // Undo move
            matrix[move[0]][move[1]] = '\0';
        }
    }
    return new int[] {bestScore, bestRow, bestCol};
}

//generateMoves() → generates the next set of available moves that the computer can play.
private List<int[]> generateMoves() {
    List<int[]> nextMoves = new ArrayList<int[]>(); // allocate List
    // If gameover, i.e., no next move
    if (checkend()>0) {
        return nextMoves;   // return empty list
    }
    // Search for empty cells and add to the List
    for (int row = 0; row < MainActivity.size; ++row) {
        for (int col = 0; col < MainActivity.size; ++col) {
            if (matrix[row][col] == '\0') {
                nextMoves.add(new int[] {row, col});
        }
      }
    }
    return nextMoves;
  }
  //evaluate() → It is called by the minimax function. Evaluates each move or the state of the matrix and returns the evaluated value
back to the minimax function.
  private int evaluate() {
    int score = 0;
    for(int i=0;i<MainActivity.size;i++)
    {
      score=score+evaluateeachrow(i);
    }
    for(int i=0;i<MainActivity.size;i++)
    {
      score=score+evaluateeachcol(i);
    }
    score = score + evaluateeachdiag1();
    score =  score + evaluateeachdiag2();
    return score;
  }
  //evaluateeachdiag1() → This function is called by the evaluate() function. This function evaluates the primary diagonal of the matrix
and returns the value back to the evaluate function.
  public int evaluateeachdiag1()
  {
    int score=0;
    for (int i=0;i<MainActivity.size;i++)
    {
      for (int j=0;j<MainActivity.size;j++)
      {
        if(i==j)
        {
          if(matrix[i][j]=='X')
          {
```

}

```
                    if(score>0)
                    {
                        score=score*10;
                    }
                    else if(score<0)
                    {
                        return 0;
                    }
                    else
                    {
                        score=10;
                    }
                }
                else if(matrix[i][j]=='O')
                {
                    if(score<0)
                    {
                        score=score*10;
                    }
                    else if(score>0)
                    {
                        return 0;
                    }
                    else
                    {
                        score=-10;
                    }
                }
            }
        }
    }
    return score;
}
```

//evaluateeachdiag2() → This function is called by the evaluate() function. This function evaluates the secondary diagonal of the matrix and returns the value back to the evaluate function.

```
public int evaluateeachdiag2()
{
    int score=0;
    for (int i=0;i<MainActivity.size;i++)
    {
        for (int j=0;j<MainActivity.size;j++)
        {
            if(i+j==MainActivity.size-1)
            {
                if(matrix[i][j]=='X')
                {
                    if(score>0)
                    {
                        score=score*10;
                    }
                    else if(score<0)
                    {
                        return 0;
                    }
                    else
                    {
                        score=10;
                    }
                }
                else if(matrix[i][j]=='O')
                {
                    if(score<0)
                    {
                        score=score*10;
                    }
                    else if(score>0)
                    {
                        return 0;
                    }
                    else
                    {
                        score=-10;
                    }
                }
            }
        }
    }
    return score;
}
//evaluateeachrow() → This function is called by the evaluate() function. This function evaluates the each row of the matrix and returns the value back to the evaluate function.
public int evaluateeachrow(int i)
{
    int score=0;
    int emptycells=0;
    for(int j=0;j<MainActivity.size;j++)
    {
        if(matrix[i][j]=='X'){
            if(score>0)
            {
                    score=score*10;
            }
            else if(score<0)
            {
                return 0;
            }
            else {
                score = 10;
            }
        }
        else if(matrix[i][j]=='O')
        {
            if(score<0)
            {
                score=score*10;
            }
            else if(score>0)
            {
                return 0;
            }
            else {
                score=-10;
            }
        }
        else if(matrix[i][j]=='\0')
        {
            emptycells++;
        }
    }
    return score+emptycells;
}
//evaluateeachcol() → This function is called by the evaluate() function. This function evaluates the each column of the matrix and returns the value back to the evaluate function.
public int evaluateeachcol(int j)
{
    int score=0;
    int emptycells=0;
    for(int i=0;i<MainActivity.size;i++)
    {
        if(matrix[i][j]=='X'){
            if(score>0)
            {
                score=score*10;
            }
            else if(score<0)
            {
                return 0;
            }
            else {
                score = 10;
            }
        }
        else if(matrix[i][j]=='O')
        {
            if(score<0)
            {
                score=score*10;
            }
            else if(score>0)
            {
                return 0;
            }
            else {
                score=-10;
            }
        }
        else if(matrix[i][j]=='\0')
        {
            emptycells++;
        }
    }
    return score;
}
//checkend() → This function checks the matrix to see if the goal state has been reached. Like Computer Won or Player Won or Game
Drawn or still the game on.
public int checkend()
{
    int count=0;
    int [] winrow= new int[MainActivity.size];
    int [] wincol= new int[MainActivity.size];
    int  windiag1=0;
    int  windiag2=0;
    int foundwin=0;
    int checkdraw=0;
    for(int i=0;i<MainActivity.size;i++)
    {
        for(int j=0;j<MainActivity.size;j++)
        {
            if(matrix[i][j]=='X')
            {
                checkdraw++;
                winrow[i]++;
                wincol[j]++;
                if(i==j)
                    windiag1++;
                if((i+j)==MainActivity.size-1)
                    windiag2++;
            }
            else if(matrix[i][j]=='O')
            {
                checkdraw++;
                winrow[i]--;
                wincol[j]--;
                if(i==j)
                    windiag1--;
                if((i+j)==MainActivity.size-1)
                    windiag2--;
            }
        }
    }
    for(int i=0; i<MainActivity.size && (foundwin!=1 || foundwin!=2);i++ )
    {
        if(winrow[i]==MainActivity.size)
```

```
            {
                foundwin = 1;
                break;
            }
            else if(winrow[i]==(-1*MainActivity.size))
            {
                foundwin = 2;
                break;
            }
            else if(wincol[i]==MainActivity.size)
            {
                foundwin = 1;
                break;
            }
            else if(wincol[i]==(-1*MainActivity.size))
            {
                foundwin = 2;
                break;
            }
        }
        if(windiag1==MainActivity.size || windiag2==MainActivity.size)
        {
            foundwin = 1;
        }
        else if(windiag1==(-1*MainActivity.size) || windiag2==(-1*MainActivity.size))
        {
            foundwin = 2;
        }
        if((foundwin!=1 || foundwin!=2)&& checkdraw==(MainActivity.size*MainActivity.size))
            return 3;
        return foundwin;
    }

}
```

- **NodeAlphaBetaNoDepth.java**

```
package com.example.venkatesh.istictactoe;

import android.util.Log;

import java.util.ArrayList;

import java.util.List;

/**

* Created by Venkatesh on 11/12/2016.

*/

public class NodeAlphaBetaNoDepth {

  char [][] matrix;

  //creates matrix of size of N*N where n is the size specified by the user. Assigns the ‘\0’ as the value to the matrix

  public NodeAlphaBetaNoDepth() {

      this.matrix = new char[MainActivity.size][MainActivity.size];

      for(int i=0;i<MainActivity.size;i++)

      {

          for (int j=0;j<MainActivity.size;j++)

          {

              matrix[i][j]='\0';

          }

      }

  }

  //place() → places X in the matrix to identify as the computer’s move and places O in the matrix to identify as the player’s move

  public void place(int row, int col,MainActivity.Seed x)

  {

      if(x==MainActivity.myseed)

          matrix[row][col]='X';

      else

          matrix[row][col]='O';

  }

  //getCompNextMove() → Checks if the solution state has been reached else calls the minimax function to get the computer’s next move

  int[] getCompNextMove()

  {
    int ifend = checkend();
    if(ifend>0)
    {
        return new int[] {-1,-1,ifend};
    }
    else
    {
        int[] temp = new int[2];
        temp = minimax(MainActivity.myseed,Integer.MIN_VALUE,Integer.MAX_VALUE);
        return new int[] {temp[1],temp[2],0};
    }
}
//minimax() → it's a recursive function which implements the minimax algorithm with alpha beta pruning
private int[] minimax(MainActivity.Seed player, int alpha, int beta) {
    // Generate possible next moves in a List of int[2] of {row, col}.
    List<int[]> nextMoves = generateMoves();
    // mySeed is maximizing; while oppSeed is minimizing
    int currentScore;
    int bestRow = -1;
    int bestCol = -1;
    Log.d("debug1","doingwork "+alpha+" "+beta);
    if (nextMoves.isEmpty()) {
        // Gameover, evaluate score
        currentScore = evaluate();
        return new int[] {currentScore, bestRow, bestCol};
    } else {
        for (int[] move : nextMoves) {
            // Try this move for the current "player"
            MainActivity.countnodes++;
            if(player==MainActivity.Seed.Cross)
                matrix[move[0]][move[1]]='X';
            else
                matrix[move[0]][move[1]]='O';
            if (player == MainActivity.myseed) {  // mySeed (computer) is maximizing player
```

```
                currentScore = minimax(MainActivity.oppseed,alpha,beta)[0];
                if (currentScore > alpha) {
                    alpha=currentScore;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            } else {  // oppSeed is minimizing player
                currentScore = minimax(MainActivity.myseed,alpha,beta)[0];
                if (currentScore < beta) {
                    beta = currentScore;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            }
            // Undo move
            matrix[move[0]][move[1]] = '\0';
        }
    }
    if(player==MainActivity.myseed)
        return new int[] {alpha, bestRow, bestCol};
    else
        return new int[] {beta, bestRow, bestCol};
}
//generateMoves() → generates the next set of available moves that the computer can play.
private List<int[]> generateMoves() {
    List<int[]> nextMoves = new ArrayList<int[]>(); // allocate List
    // If gameover, i.e., no next move
    if (checkend()>0) {
        return nextMoves;   // return empty list
    }
    // Search for empty cells and add to the List
    for (int row = 0; row < MainActivity.size; ++row) {
        for (int col = 0; col < MainActivity.size; ++col) {
            if (matrix[row][col] == '\0') {
                    nextMoves.add(new int[] {row, col});
                }
            }
        }
        return nextMoves;
    }
    //evaluate() → It is called by the minimax function. Evaluates each move or the state of the matrix and returns the evaluated value back to the minimax function.
    private int evaluate() {
        int score = 0;
        for(int i=0;i<MainActivity.size;i++)
        {
            score=score+evaluateeachrow(i);
        }
        for(int i=0;i<MainActivity.size;i++)
        {
            score=score+evaluateeachcol(i);
        }
        score = score + evaluateeachdiag1();
        score =  score + evaluateeachdiag2();
        return score;
    }
    //evaluateeachdiag1() → This function is called by the evaluate() function. This function evaluates the primary diagonal of the matrix and returns the value back to the evaluate function.
    public int evaluateeachdiag1()
    {
        int score=0;
        for (int i=0;i<MainActivity.size;i++)
        {
            for (int j=0;j<MainActivity.size;j++)
            {
                if(i==j)
                {
                    if(matrix[i][j]=='X')
                {
                    if(score>0)
                    {
                        score=score*10;
                    }
                    else if(score<0)
                    {
                        return 0;
                    }
                    else
                    {
                        score=10;
                    }
                }
                else if(matrix[i][j]=='O')
                {
                    if(score<0)
                    {
                        score=score*10;
                    }
                    else if(score>0)
                    {
                        return 0;
                    }
                    else
                    {
                        score=-10;
                    }
                }
            }
        }
    }
    return score;
}
```

```
//evaluateeachdiag2() → This function is called by the evaluate() function. This function evaluates the secondary diagonal of the matrix and returns the value back to the evaluate function.

public int evaluateeachdiag2()

{

    int score=0;

    for (int i=0;i<MainActivity.size;i++)

    {

        for (int j=0;j<MainActivity.size;j++)

        {

            if(i+j==MainActivity.size-1)

            {

                if(matrix[i][j]=='X')

                {

                    if(score>0)

                    {

                        score=score*10;

                    }

                    else if(score<0)

                    {

                        return 0;

                    }

                    else

                    {

                        score=10;

                    }

                }

                else if(matrix[i][j]=='O')

                {

                    if(score<0)

                    {

                        score=score*10;

                    }

                    else if(score>0)

                    {
                        return 0;
                    }
                    else
                    {
                        score=-10;
                    }
                }
            }
        }
    }
    return score;
}
//evaluateeachrow() → This function is called by the evaluate() function. This function evaluates the each row of the matrix and returns the value back to the evaluate function.
public int evaluateeachrow(int i)
{
    int score=0;
    int emptycells=0;
    for(int j=0;j<MainActivity.size;j++)
    {
        if(matrix[i][j]=='X'){
            if(score>0)
            {
                    score=score*10;
            }
            else if(score<0)
            {
                return 0;
            }
            else {
                score = 10;
            }
        }
        else if(matrix[i][j]=='O')
```

```
            {
                if(score<0)
                {
                    score=score*10;
                }
                else if(score>0)
                {
                    return 0;
                }
                else {
                    score=-10;
                }
            }
            else if(matrix[i][j]=='\0')
            {
                emptycells++;
            }
        }
        return score+emptycells;
    }
    //evaluateeachcol() → This function is called by the evaluate() function. This function evaluates the each column of the matrix and returns the value back to the evaluate function.
    public int evaluateeachcol(int j)
    {
        int score=0;
        int emptycells=0;
        for(int i=0;i<MainActivity.size;i++)
        {
            if(matrix[i][j]=='X'){
                if(score>0)
                {
                    score=score*10;
                }
                else if(score<0)
                {
                    return 0;
                }
                else {
                    score = 10;
                }
            }
            else if(matrix[i][j]=='O')
            {
                if(score<0)
                {
                    score=score*10;
                }
                else if(score>0)
                {
                    return 0;
                }
                else {
                    score=-10;
                }
            }
            else if(matrix[i][j]=='\0')
            {
                emptycells++;
            }
        }
        return score;
    }
    //checkend() → This function checks the matrix to see if the goal state has been reached. Like Computer Won or Player Won or Game Drawn or still the game on.
    public int checkend()
    {
        int count=0;
```

```
int [] winrow= new int[MainActivity.size];
int [] wincol= new int[MainActivity.size];
int  windiag1=0;
int  windiag2=0;
int foundwin=0;
int checkdraw=0;
for(int i=0;i<MainActivity.size;i++)
{
    for(int j=0;j<MainActivity.size;j++)
    {
        if(matrix[i][j]=='X')
        {
            checkdraw++;
            winrow[i]++;
            wincol[j]++;
            if(i==j)
                windiag1++;
            if((i+j)==MainActivity.size-1)
                windiag2++;
        }
        else if(matrix[i][j]=='O')
        {
            checkdraw++;
            winrow[i]--;
            wincol[j]--;
            if(i==j)
                windiag1--;
            if((i+j)==MainActivity.size-1)
                windiag2--;
        }
    }
}
for(int i=0; i<MainActivity.size && (foundwin!=1 || foundwin!=2);i++ )
{
```

```
            if(winrow[i]==MainActivity.size)
            {
                foundwin = 1;
                break;
            }
            else if(winrow[i]==(-1*MainActivity.size))
            {
                foundwin = 2;
                break;
            }
            else if(wincol[i]==MainActivity.size)
            {
                foundwin = 1;
                break;
            }
            else if(wincol[i]==(-1*MainActivity.size))
            {
                foundwin = 2;
                break;
            }
        }
        if(windiag1==MainActivity.size || windiag2==MainActivity.size)
        {
            foundwin = 1;
        }
        else if(windiag1==(-1*MainActivity.size) || windiag2==(-1*MainActivity.size))
        {
            foundwin = 2;
        }
        if((foundwin!=1 || foundwin!=2)&& checkdraw==(MainActivity.size*MainActivity.size))
            return 3;
        return foundwin;
    }
}
```

- **NodeAlphaBetaWithDepth.java**

```
package com.example.venkatesh.istictactoe;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.Log;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
/**
* Created by Venkatesh on 11/12/2016.
*/
public class NodeAlphaBetaWithDepth {
  char [][] matrix;
  //creates matrix of size of N*N where n is the size specified by the user. Assigns the '\0' as the value to the matrix
  public NodeAlphaBetaWithDepth() {
      this.matrix = new char[MainActivity.size][MainActivity.size];
      for(int i=0;i<MainActivity.size;i++)
      {
          for (int j=0;j<MainActivity.size;j++)
          {
              matrix[i][j]='\0';
          }
      }
  }
  //place() → places X in the matrix to identify as the computer's move and places O in the matrix to identify as the player's move
  public void place(int row, int col,MainActivity.Seed x)
  {
      if(x==MainActivity.myseed)
          matrix[row][col]='X';
      else
          matrix[row][col]='O';
  }
```

```
//getCompNextMove() → Checks if the solution state has been reached else calls the minimax function to get the computer's next move

int[] getCompNextMove()

{

    int ifend = checkend();

    if(ifend>0)

    {

        return new int[] {-1,-1,ifend};

    }

    else

    {

        int[] temp = new int[2];

        if(MainActivity.size==3)

            temp = minimax(MainActivity.size+1,MainActivity.myseed,Integer.MIN_VALUE,Integer.MAX_VALUE);

        else

            temp = minimax(MainActivity.size,MainActivity.myseed,Integer.MIN_VALUE,Integer.MAX_VALUE);

        return new int[] {temp[1],temp[2],0};

    }

}

//minimax() → it's a recursive function which implements the minimax algorithm with alpha beta pruning

private int[] minimax(int depth,MainActivity.Seed player, int alpha, int beta) {

    // Generate possible next moves in a List of int[2] of {row, col}.

    List<int[]> nextMoves = generateMoves();

    // mySeed is maximizing; while oppSeed is minimizing

    int currentScore;

    int bestRow = -1;

    int bestCol = -1;

    if (nextMoves.isEmpty() || depth==0) {

        // Gameover or depth reached, evaluate score

        currentScore = evaluate();

        return new int[] {currentScore, bestRow, bestCol};

    } else {

        for (int[] move : nextMoves) {

            // Try this move for the current "player"
            MainActivity.countnodes++;
            if(player==MainActivity.Seed.Cross)
                matrix[move[0]][move[1]]='X';
            else
                matrix[move[0]][move[1]]='O';
            if (player == MainActivity.myseed) {  // mySeed (computer) is maximizing player
                currentScore = minimax(depth-1,MainActivity.oppseed,alpha,beta)[0];
                if (currentScore > alpha) {
                    alpha=currentScore;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            } else {  // oppSeed is minimizing player
                currentScore = minimax(depth-1,MainActivity.myseed,alpha,beta)[0];
                if (currentScore < beta) {
                    beta = currentScore;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            }
            // Undo move
            matrix[move[0]][move[1]] = '\0';
        }
    }
    if(player==MainActivity.myseed)
        return new int[] {alpha, bestRow, bestCol};
    else
        return new int[] {beta, bestRow, bestCol};
}
//generateMoves() → generates the next set of available moves that the computer can play.
private List<int[]> generateMoves() {
    List<int[]> nextMoves = new ArrayList<int[]>(); // allocate List

    // If gameover, i.e., no next move
```

```
      if (checkend()>0) {
          return nextMoves;   // return empty list
      }
      // Search for empty cells and add to the List
      for (int row = 0; row < MainActivity.size; ++row) {
          for (int col = 0; col < MainActivity.size; ++col) {
              if (matrix[row][col] == '\0') {
                  nextMoves.add(new int[] {row, col});
              }
          }
      }
      return nextMoves;
  }
  //evaluate() → It is called by the minimax function. Evaluates each move or the state of the matrix and returns the evaluated value back to the minimax function.
  private int evaluate() {
      int score = 0;
      for(int i=0;i<MainActivity.size;i++)
      {
          score=score+evaluateeachrow(i);
      }
      for(int i=0;i<MainActivity.size;i++)
      {
          score=score+evaluateeachcol(i);
      }
      score = score + evaluateeachdiag1();
      score =  score + evaluateeachdiag2();
      return score;
  }
  //evaluateeachdiag1() → This function is called by the evaluate() function. This function evaluates the primary diagonal of the matrix and returns the value back to the evaluate function.
  public int evaluateeachdiag1()
  {
      int score=0;
```

```
for (int i=0;i<MainActivity.size;i++)

{

    for (int j=0;j<MainActivity.size;j++)

    {

        if(i==j)

        {

            if(matrix[i][j]=='X')

            {

                if(score>0)

                {

                    score=score*10;

                }

                else if(score<0)

                {

                    return 0;

                }

                else

                {

                    score=10;

                }

            }

            else if(matrix[i][j]=='O')

            {

                if(score<0)

                {

                    score=score*10;

                }

                else if(score>0)

                {

                    return 0;

                }

                else

                {

                    score=-10;
```

```
                        }
                    }
                }
            }
        }
        return score;
    }
    //evaluateeachdiag2() → This function is called by the evaluate() function. This function evaluates the secondary diagonal of the matrix and returns the value back to the evaluate function.
    public int evaluateeachdiag2()
    {
        int score=0;
        for (int i=0;i<MainActivity.size;i++)
        {
            for (int j=0;j<MainActivity.size;j++)
            {
                if(i+j==MainActivity.size-1)
                {
                    if(matrix[i][j]=='X')
                    {
                        if(score>0)
                        {
                            score=score*10;
                        }
                        else if(score<0)
                        {
                            return 0;
                        }
                        else
                        {
                            score=10;
                        }
                    }
                    else if(matrix[i][j]=='O')
                    {
                        if(score<0)
                        {
                            score=score*10;
                        }
                        else if(score>0)
                        {
                            return 0;
                        }
                        else
                        {
                            score=-10;
                        }
                    }
                }
            }
        }
        return score;
    }
```

//evaluateeachrow() → This function is called by the evaluate() function. This function evaluates the each row of the matrix and returns the value back to the evaluate function.

```
public int evaluateeachrow(int i)
{
    int score=0;
    int emptycells=0;
    for(int j=0;j<MainActivity.size;j++)
    {
        if(matrix[i][j]=='X'){
            if(score>0)
            {
                    score=score*10;
            }
            else if(score<0)
            {
                return 0;
            }
            else {
                score = 10;
            }
        }
        else if(matrix[i][j]=='O')
        {
            if(score<0)
            {
                score=score*10;
            }
            else if(score>0)
            {
                return 0;
            }
            else {
                score=-10;
            }
        }
        else if(matrix[i][j]=='\0')
        {
            emptycells++;
        }
    }
    return score+emptycells;
}
//evaluateeachcol() → This function is called by the evaluate() function. This function evaluates the each column of the matrix and returns the value back to the evaluate function.
public int evaluateeachcol(int j)
{
    int score=0;
    int emptycells=0;
```

```
for(int i=0;i<MainActivity.size;i++)

{

    if(matrix[i][j]=='X'){

        if(score>0)

        {

            score=score*10;

        }

        else if(score<0)

        {

            return 0;

        }

        else {

            score = 10;

        }

    }

    else if(matrix[i][j]=='O')

    {

        if(score<0)

        {

            score=score*10;

        }

        else if(score>0)

        {

            return 0;

        }

        else {

            score=-10;

        }

    }

    else if(matrix[i][j]=='\0')

    {

        emptycells++;

    }

}
```

```
        return score;

    }

    //checkend() → This function checks the matrix to see if the goal state has been reached. Like Computer Won or Player Won or Game Drawn or still the game on.

    public int checkend()

    {

        int count=0;

        int [] winrow= new int[MainActivity.size];

        int [] wincol= new int[MainActivity.size];

        int  windiag1=0;

        int  windiag2=0;

        int foundwin=0;

        int checkdraw=0;

        for(int i=0;i<MainActivity.size;i++)

        {

            for(int j=0;j<MainActivity.size;j++)

            {

                if(matrix[i][j]=='X')

                {

                    checkdraw++;

                    winrow[i]++;

                    wincol[j]++;

                    if(i==j)

                        windiag1++;

                    if((i+j)==MainActivity.size-1)

                        windiag2++;

                }

                else if(matrix[i][j]=='O')

                {

                    checkdraw++;

                    winrow[i]--;

                    wincol[j]--;

                    if(i==j)

                        windiag1--;
                if((i+j)==MainActivity.size-1)
                    windiag2--;
            } } }
    for(int i=0; i<MainActivity.size && (foundwin!=1 || foundwin!=2);i++ )
    {
        if(winrow[i]==MainActivity.size)
        {
            foundwin = 1;
            break;
        }
        else if(winrow[i]==(-1*MainActivity.size))
        {
            foundwin = 2;
            break;
        }
        else if(wincol[i]==MainActivity.size)
        {
            foundwin = 1;
            break;
        }
        else if(wincol[i]==(-1*MainActivity.size))
        {
            foundwin = 2;
            break;  }}
    if(windiag1==MainActivity.size || windiag2==MainActivity.size)
    {
        foundwin = 1;}
    else if(windiag1==(-1*MainActivity.size) || windiag2==(-1*MainActivity.size))
    {
        foundwin = 2;}
    if((foundwin!=1 || foundwin!=2)&& checkdraw==(MainActivity.size*MainActivity.size))
        return 3;
    return foundwin;
}}
```

- **NodeAlphaBetaWithDepth3.java**

```
package com.example.venkatesh.istictactoe;

import java.util.ArrayList;

import java.util.List;

/**

* Created by Venkatesh on 11/12/2016.

*/

public class NodeAlphaBetaWithDepth3 {

  char [][] matrix;

  //creates matrix of size of N*N where n is the size specified by the user. Assigns the '\0' as the value to the matrix

  public NodeAlphaBetaWithDepth3() {

    this.matrix = new char[MainActivity.size][MainActivity.size];

    for(int i=0;i<MainActivity.size;i++)

    {

      for (int j=0;j<MainActivity.size;j++)

      {

        matrix[i][j]='\0';

      }

    }

  }

  //place() → places X in the matrix to identify as the computer's move and places O in the matrix to identify as the player's move

  public void place(int row, int col,MainActivity.Seed x)

  {

    if(x==MainActivity.myseed)

      matrix[row][col]='X';

    else

      matrix[row][col]='O';

  }

  //getCompNextMove() → Checks if the solution state has been reached else calls the minimax function to get the computer's next move

  int[] getCompNextMove()

  {

    int ifend = checkend();

    if(ifend>0)
    {
        return new int[] {-1,-1,ifend};
    }
    else
    {
        int[] temp = new int[2];
        if(MainActivity.size==3)
            temp = minimax(MainActivity.size+1,MainActivity.myseed,Integer.MIN_VALUE,Integer.MAX_VALUE);
        else
            temp = minimax(3,MainActivity.myseed,Integer.MIN_VALUE,Integer.MAX_VALUE);
        return new int[] {temp[1],temp[2],0};
    }
}
//minimax() → it's a recursive function which implements the minimax algorithm with alpha beta pruning
private int[] minimax(int depth,MainActivity.Seed player, int alpha, int beta) {
    // Generate possible next moves in a List of int[2] of {row, col}.
    List<int[]> nextMoves = generateMoves();
    // mySeed is maximizing; while oppSeed is minimizing
    int currentScore;
    int bestRow = -1;
    int bestCol = -1;
    if (nextMoves.isEmpty() || depth==0) {
        // Gameover or depth reached, evaluate score
        currentScore = evaluate();
        return new int[] {currentScore, bestRow, bestCol};
    } else {
        for (int[] move : nextMoves) {
            // Try this move for the current "player"
            MainActivity.countnodes++;
            if(player==MainActivity.Seed.Cross)
                matrix[move[0]][move[1]]='X';
            else
                matrix[move[0]][move[1]]='O';
            if (player == MainActivity.myseed) {  // mySeed (computer) is maximizing player
```

```
                currentScore = minimax(depth-1,MainActivity.oppseed,alpha,beta)[0];
                if (currentScore > alpha) {
                    alpha=currentScore;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            } else {  // oppSeed is minimizing player
                currentScore = minimax(depth-1,MainActivity.myseed,alpha,beta)[0];
                if (currentScore < beta) {
                    beta = currentScore;
                    bestRow = move[0];
                    bestCol = move[1];
                }
            }
            // Undo move
            matrix[move[0]][move[1]] = '\0';
        }
    }
    if(player==MainActivity.myseed)
        return new int[] {alpha, bestRow, bestCol};
    else
        return new int[] {beta, bestRow, bestCol};
}
//generateMoves() → generates the next set of available moves that the computer can play.
private List<int[]> generateMoves() {
    List<int[]> nextMoves = new ArrayList<int[]>(); // allocate List
    // If gameover, i.e., no next move
    if (checkend()>0) {
        return nextMoves;   // return empty list
    }
    // Search for empty cells and add to the List
    for (int row = 0; row < MainActivity.size; ++row) {
        for (int col = 0; col < MainActivity.size; ++col) {
            if (matrix[row][col] == '\0') {
                nextMoves.add(new int[] {row, col});
            }
        }
    }
    return nextMoves;
}
//evaluate() → It is called by the minimax function. Evaluates each move or the state of the matrix and returns the evaluated value back to the minimax function.
private int evaluate() {
    int score = 0;
    for(int i=0;i<MainActivity.size;i++)
    {
        score=score+evaluateeachrow(i);
    }
    for(int i=0;i<MainActivity.size;i++)
    {
        score=score+evaluateeachcol(i);
    }
    score = score + evaluateeachdiag1();
    score =  score + evaluateeachdiag2();
    return score;
}
//evaluateeachdiag1() → This function is called by the evaluate() function. This function evaluates the primary diagonal of the matrix and returns the value back to the evaluate function.
public int evaluateeachdiag1()
{
    int score=0;
    for (int i=0;i<MainActivity.size;i++)
    {
        for (int j=0;j<MainActivity.size;j++)
        {
            if(i==j)
            {
                if(matrix[i][j]=='X')
                {
                    if(score>0)
                    {
                        score=score*10;
                    }
                    else if(score<0)
                    {
                        return 0;
                    }
                    else
                    {
                        score=10;
                    }
                }
                else if(matrix[i][j]=='O')
                {
                    if(score<0)
                    {
                        score=score*10;
                    }
                    else if(score>0)
                    {
                        return 0;
                    }
                    else
                    {
                        score=-10;
                    }
                }
            }
        }
    }
    return score;
}
```

//evaluateeachdiag2() → This function is called by the evaluate() function. This function evaluates the secondary diagonal of the matrix and returns the value back to the evaluate function.

```
public int evaluateeachdiag2()
{
    int score=0;
    for (int i=0;i<MainActivity.size;i++)
    {
        for (int j=0;j<MainActivity.size;j++)
        {
            if(i+j==MainActivity.size-1)
            {
                if(matrix[i][j]=='X')
                {
                    if(score>0)
                    {
                        score=score*10;
                    }
                    else if(score<0)
                    {
                        return 0;
                    }
                    else
                    {
                        score=10;
                    }
                }
                else if(matrix[i][j]=='O')
                {
                    if(score<0)
                    {
                        score=score*10;
                    }
                    else if(score>0)
                    {
                        return 0;
                    }
                    else
                    {
                        score=-10;
                    }
                }
            }
        }
    }
    return score;
}
//evaluateeachrow() → This function is called by the evaluate() function. This function evaluates the each row of the matrix and returns the value back to the evaluate function.
public int evaluateeachrow(int i)
{
    int score=0;
    int emptycells=0;
    for(int j=0;j<MainActivity.size;j++)
    {
        if(matrix[i][j]=='X'){
            if(score>0)
            {
                    score=score*10;
            }
            else if(score<0)
            {
                return 0;
            }
            else {
                score = 10;
            }
        }
        else if(matrix[i][j]=='O')
```

```
            {
                if(score<0)
                {
                    score=score*10;
                }
                else if(score>0)
                {
                    return 0;
                }
                else {
                    score=-10;
                }
            }
            else if(matrix[i][j]=='\0')
            {
                emptycells++;
            }
        }
        return score+emptycells;
    }
```

//evaluateeachcol() → This function is called by the evaluate() function. This function evaluates the each column of the matrix and returns the value back to the evaluate function.

```
    public int evaluateeachcol(int j)
    {
        int score=0;
        int emptycells=0;
        for(int i=0;i<MainActivity.size;i++)
        {
            if(matrix[i][j]=='X'){
                if(score>0)
                {
                    score=score*10;
                }
                else if(score<0)
                {
                    return 0;
                }
                else {
                    score = 10;
                }
            }
            else if(matrix[i][j]=='O')
            {
                if(score<0)
                {
                    score=score*10;
                }
                else if(score>0)
                {
                    return 0;
                }
                else {
                    score=-10;
                }
            }
            else if(matrix[i][j]=='\0')
            {
                emptycells++;
            }
        }
        return score;
    }
    //checkend() → This function checks the matrix to see if the goal state has been reached. Like Computer Won or Player Won or Game Drawn or still the game on.
    public int checkend()
    {
        int count=0;
        int [] winrow= new int[MainActivity.size];
```

```
int [] wincol= new int[MainActivity.size];
int  windiag1=0;
int  windiag2=0;
int foundwin=0;
int checkdraw=0;
for(int i=0;i<MainActivity.size;i++)
{
    for(int j=0;j<MainActivity.size;j++)
    {
        if(matrix[i][j]=='X')
        {
            checkdraw++;
            winrow[i]++;
            wincol[j]++;
            if(i==j)
                windiag1++;
            if((i+j)==MainActivity.size-1)
                windiag2++;
        }
        else if(matrix[i][j]=='O')
        {
            checkdraw++;
            winrow[i]--;
            wincol[j]--;
            if(i==j)
                windiag1--;
            if((i+j)==MainActivity.size-1)
                windiag2--;
        }
    }
}
for(int i=0; i<MainActivity.size && (foundwin!=1 || foundwin!=2);i++ )
{
    if(winrow[i]==MainActivity.size)
```

```
            {
                foundwin = 1;
                break;
            }
            else if(winrow[i]==(-1*MainActivity.size))
            {
                foundwin = 2;
                break;
            }
            else if(wincol[i]==MainActivity.size)
            {
                foundwin = 1;
                break;
            }
            else if(wincol[i]==(-1*MainActivity.size))
            {
                foundwin = 2;
                break;
            }
        }
        if(windiag1==MainActivity.size || windiag2==MainActivity.size)
        {
            foundwin = 1;
        }
        else if(windiag1==(-1*MainActivity.size) || windiag2==(-1*MainActivity.size))
        {
            foundwin = 2;
        }
        if((foundwin!=1 || foundwin!=2)&& checkdraw==(MainActivity.size*MainActivity.size))
            return 3;
        return foundwin;
    }
}
```

- **SettingsPreference.java**

```
package com.example.venkatesh.istictactoe;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.EditTextPreference;
import android.preference.ListPreference;
import android.preference.Preference;
import android.preference.PreferenceActivity;
import android.preference.PreferenceManager;
import android.widget.Toast;
/**
* Created by Venkatesh on 11/24/2016.
*/
public class SettingsPreference extends PreferenceActivity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferences);
    ListPreference listPreference = (ListPreference) findPreference("algoPreference");
    final ListPreference listPreferencematrix = (ListPreference) findPreference("matrixPreference");
    ListPreference listPreferencesymbol = (ListPreference) findPreference("symbolPreference");
    ListPreference listPreferencefirstplayer = (ListPreference) findPreference("playfirstPreference");
    final EditTextPreference editcustommatrixsize = (EditTextPreference) findPreference("custommatrixsize");
    listPreferencefirstplayer.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
      @Override
      public boolean onPreferenceChange(Preference preference, Object newValue) {
        if(newValue.equals("computer"))
        {
            Toast.makeText(getApplicationContext(),"Computer will make the first move. You play after computer's turn",Toast.LENGTH_LONG).show();
            Intent i;
            i = new Intent(getApplicationContext(),MainActivity.class);
            finish();
```

```
                    startActivity(i);
                }else if(newValue.equals("player")){
                    Toast.makeText(getApplicationContext(),"You will make the first move. Computer will play after your turn",Toast.LENGTH_LONG).show();
                    Intent i;
                    i = new Intent(getApplicationContext(),MainActivity.class);
                    finish();
                    startActivity(i);
                }
                return true;
            }
        });
        listPreferencesymbol.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                if(newValue.equals("X"))
                {
                    Toast.makeText(getApplicationContext(),"Your symbol changed to X, Computer's symbol is O now",Toast.LENGTH_LONG).show();
                    Intent i;
                    i = new Intent(getApplicationContext(),MainActivity.class);
                    finish();
                    startActivity(i);
                }
                else if(newValue.equals("O"))
                {
                    Toast.makeText(getApplicationContext(),"Your symbol changed to O, Computer's symbol is X now",Toast.LENGTH_LONG).show();
                    Intent i;
                    i = new Intent(getApplicationContext(),MainActivity.class);
                    finish();
                    startActivity(i);
                }
                return true;
            }
```

```
});
listPreferencematrix.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
    @Override
    public boolean onPreferenceChange(Preference preference, Object newValue) {
        SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        String tempType = pref.getString("matrixPreference","");
        if(newValue.equals("three"))
        {
            Toast.makeText(getApplicationContext(),"Selected grid size is 3",Toast.LENGTH_LONG).show();
            editcustommatrixsize.setText("3");
            Intent i;
            i = new Intent(getApplicationContext(),MainActivity.class);
            finish();
            startActivity(i);
        }
        else if(newValue.equals("four"))
        {
            Toast.makeText(getApplicationContext(),"Selected grid size is 4",Toast.LENGTH_LONG).show();
            Intent i;
            editcustommatrixsize.setText("4");
            i = new Intent(getApplicationContext(),MainActivity.class);
            finish();
            startActivity(i);
        }
        else if(newValue.equals("five"))
        {
            Toast.makeText(getApplicationContext(),"Selected grid size is 5",Toast.LENGTH_LONG).show();
            Intent i;
            editcustommatrixsize.setText("5");
            i = new Intent(getApplicationContext(),MainActivity.class);
            finish();
            startActivity(i);
        }
        else if(newValue.equals("six"))
```

```
                {
                    Toast.makeText(getApplicationContext(),"Selected grid size is 6",Toast.LENGTH_LONG).show();
                    Intent i;
                    editcustommatrixsize.setText("6");
                    i = new Intent(getApplicationContext(),MainActivity.class);
                    finish();
                    startActivity(i);
                }
                else if(newValue.equals("seven"))
                {
                    Toast.makeText(getApplicationContext(),"Selected grid size is 7",Toast.LENGTH_LONG).show();
                    Intent i;
                    editcustommatrixsize.setText("7");
                    i = new Intent(getApplicationContext(),MainActivity.class);
                    finish();
                    startActivity(i);
                }
                return true;
            }
        });
        listPreference.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
                String tempType = pref.getString("algoPreference","");
                if(newValue.equals("AD"))
                {
                    Toast.makeText(getApplicationContext(),"Selected algorithm is MiniMax Alpha Beta with CutOff",Toast.LENGTH_LONG).show();
                    Intent i;
                    i = new Intent(getApplicationContext(),MainActivity.class);
                    finish();
                    startActivity(i);
                }
                else if(newValue.equals("AN"))
                {
                    Toast.makeText(getApplicationContext(),"Selected algorithm is MiniMax Alpha Beta without CutOff",Toast.LENGTH_LONG).show();
                    Intent i;
                    i = new Intent(getApplicationContext(),MainActivity.class);
                    finish();
                    startActivity(i);
                }
                else if(newValue.equals("MD"))
                {
                    Toast.makeText(getApplicationContext(),"Selected algorithm is MiniMax with CutOff",Toast.LENGTH_LONG).show();
                    Intent i;
                    i = new Intent(getApplicationContext(),MainActivity.class);
                    finish();
                    startActivity(i);
                }
                else if(newValue.equals("MN"))
                {
                    Toast.makeText(getApplicationContext(),"Selected algorithm is MiniMax without CutOff",Toast.LENGTH_LONG).show();
                    Intent i;
                    i = new Intent(getApplicationContext(),MainActivity.class);
                    finish();
                    startActivity(i);
                }
                else if(newValue.equals("ADD"))
                {
                    Toast.makeText(getApplicationContext(),"Selected algorithm is MiniMax Alpha Beta with CutOff 3",Toast.LENGTH_LONG).show();
                    Intent i;
                    i = new Intent(getApplicationContext(),MainActivity.class);
                    finish();
                    startActivity(i);
                }
```

```
                return true;
            }
        });
        editcustommatrixsize.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                Toast.makeText(getApplicationContext(),"Selected grid size is "+newValue.toString(),Toast.LENGTH_LONG).show();
                editcustommatrixsize.setText(newValue.toString());
                Intent i;
                i = new Intent(getApplicationContext(),MainActivity.class);
                finish();
                startActivity(i);
                return false;
            }
        });
    }
    public void onBackPressed()
    {
        Intent i;
            i = new Intent(getApplicationContext(),MainActivity.class);
        finish();
        startActivity(i);
        return;
    }
}
```

- **WelcomePage.java**

```
package com.example.venkatesh.istictactoe;
import android.content.Intent;
import android.media.Image;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ImageView;
public class WelcomePage extends AppCompatActivity {
  ImageView imageView;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_welcome_page);
    imageView = (ImageView) findViewById(R.id.imageView2);
    Animation myFadeInAnimation = AnimationUtils.loadAnimation(this, R.anim.fadein);
    imageView.startAnimation(myFadeInAnimation);
    Button playgame = (Button) findViewById(R.id.btnplay);
    playgame.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent i = new Intent(getApplicationContext(),MainActivity.class);
        finish();
        startActivity(i);
      }
    });
  }
}
```

- **activity_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

  xmlns:tools="http://schemas.android.com/tools"

  android:layout_width="match_parent"

  android:layout_height="match_parent"

  android:padding="10dp"

  android:orientation="vertical"

  android:id="@+id/root"

  tools:context="com.example.venkatesh.istictactoe.MainActivity">

</LinearLayout>
```

- **activity_welcome_page.xml**

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

  xmlns:tools="http://schemas.android.com/tools"

  android:layout_width="match_parent"

  android:layout_height="match_parent"

  android:orientation="vertical"

  android:weightSum="5"

  android:background="#ffffff"

  tools:context="com.example.venkatesh.istictactoe.WelcomePage">


  <TextView

     android:layout_width="wrap_content"

     android:layout_height="wrap_content"

     android:text="Tic Tac Toe"

     android:gravity="center"

     android:layout_gravity="center"

     android:textSize="40dp"


     android:textColor="#F2352F"

     android:layout_marginTop="20dp"

     android:layout_weight="1"
```

```
        android:id="@+id/textView" />

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/imageView2"
        android:src="@mipmap/tictactoe3"
        android:layout_weight="3"
         />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Play Game"
        android:id="@+id/btnplay"

        android:layout_gravity="center"
        android:layout_centerHorizontal="true" />



</LinearLayout>
```

- **menu.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/settings"
        android:title="Settings"
        app:showAsAction="never"
        android:orderInCategory="1"/>

</menu>
```

- **preferences.xml**

```
<?xml version="1.0" encoding="utf-8"?>

<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

  <ListPreference

    android:key="matrixPreference"

    android:entries="@array/matrixTyper"

    android:entryValues="@array/matrixTypeValues"

    android:defaultValue="three"

    android:title="@string/preference1"

    android:summary="@string/preferenceSummary1"

    />

    <ListPreference

      android:key="algoPreference"

      android:entries="@array/algoTyper"

      android:entryValues="@array/algoTypeValues"

      android:defaultValue="AD"

      android:title="@string/preference"

      android:summary="@string/preferenceSummary"/>

  <ListPreference

    android:key="symbolPreference"

    android:entries="@array/symbolTyper"

    android:entryValues="@array/symbolTypeValues"

    android:defaultValue="O"

    android:title="@string/preference2"

    android:summary="@string/preferenceSummary2"

    />

  <ListPreference

    android:key="playfirstPreference"

    android:entries="@array/playfirstTyper"

    android:entryValues="@array/playfirstValues"

    android:defaultValue="player"

    android:title="@string/preference3"

    android:summary="@string/preferenceSummary3"

    />
  <EditTextPreference
    android:key="custommatrixsize"
    android:inputType="number"
    android:defaultValue="3"
    android:title="Enter a custom matrix size"
    android:summary="This is not recommended as choosing a size more than 7 will be not clearer to play if the screen size is small"
    />
</PreferenceScreen>
```