
UTM

UNIVERSITI TEKNOLOGI MALAYSIA

# SESSION 2019/2020 SEMESTER 1

# PROJECT

# GROUP 2

**NAME**:

1) CHAI CHEAH WEN (A17CS0028)

2) GOH CHIANG CHENG (A17CS0048)

3) LIM BAO JING (A17CS0076)

4) LOW CHIA JING (A17CS0083)

4) TAN SEE JOU (A17CS0218)

**LECTURER NAME:** ASSOC PROF DR AZLAN MOHD ZAIN

**SUBJECT NAME:** ARTIFICIAL INTELLIGENCE

**SUBJECT CODE:** SCSJ 3553

**SECTION:** 08

Contents

## 1. Introduction

This is the project for the course of Artificial Intelligence (SCSJ3553 – 08). We are from a group of five students: Chai Cheah Wen, Goh Chian Cheng, Lim Bao Jing, Low Chia Jing and Tan See Jou. In this project, we have developed a tic tac toe game using Artificial Intelligence. We have studied some reference to the making of the game based on the heuristic approach, the game rules behind tic tac toe and also how can we apply AI in our game. The outline of our discussion is as follows; section 2 discusses the description of the game, section 3 discusses the tools and software we used in developing the game, section 4 highlights the flow chart of process the game, section 5 introduced the artificial intelligence heuristic used in the game production and its explanations section 6 shows the source code in the game production and its explanations, section 7 shows the user manual of our game and the last section is the conclusion.

## 2. Description of the Game

Tic-Tac-Toe (Figure 1) is a famous game around the world that is suitable for different age groups. It's a simple and fun two-player game. Before starting the game, a 3x3 grid must be formed by using two vertical and two horizontal lines. The players can fill the nine spaces with any symbol they like but the symbol must be different for both players. However, the commonly used symbols are crosses ('X') and noughts ('O'). The winning condition for this game is placing three similar marks in horizontal, vertical or diagonal rows. The game will be over when it is a draw or either one of players successfully placing three similar marks in any horizontal, vertical or diagonal rows.

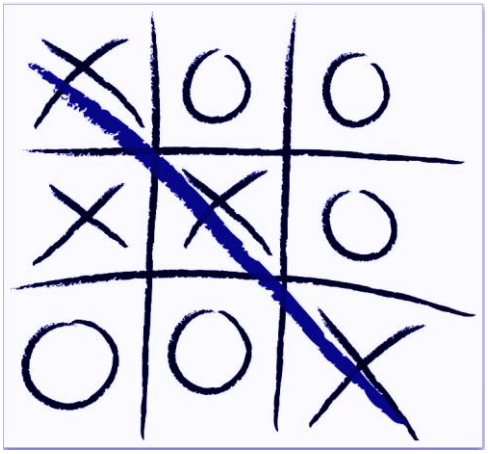


Figure 1: Tic-Tac-Toe game

## 3. Software and Tools We Used

As we know, there is a lot of software and tools which can be used to develop a game. In our case, we choose Visual Studio 2019 as our tool to develop this tic-tac-toe game. The reason we chose this tool to develop this game because most of us are familiar with it and this tool is easy to use. Although the game developed by using this tool doesn't come with a fancy user interface compared to other game development software such as unity, we tried to improve the game user interface by using the graphics library which provided for this tool.

## 4. Flow Chart

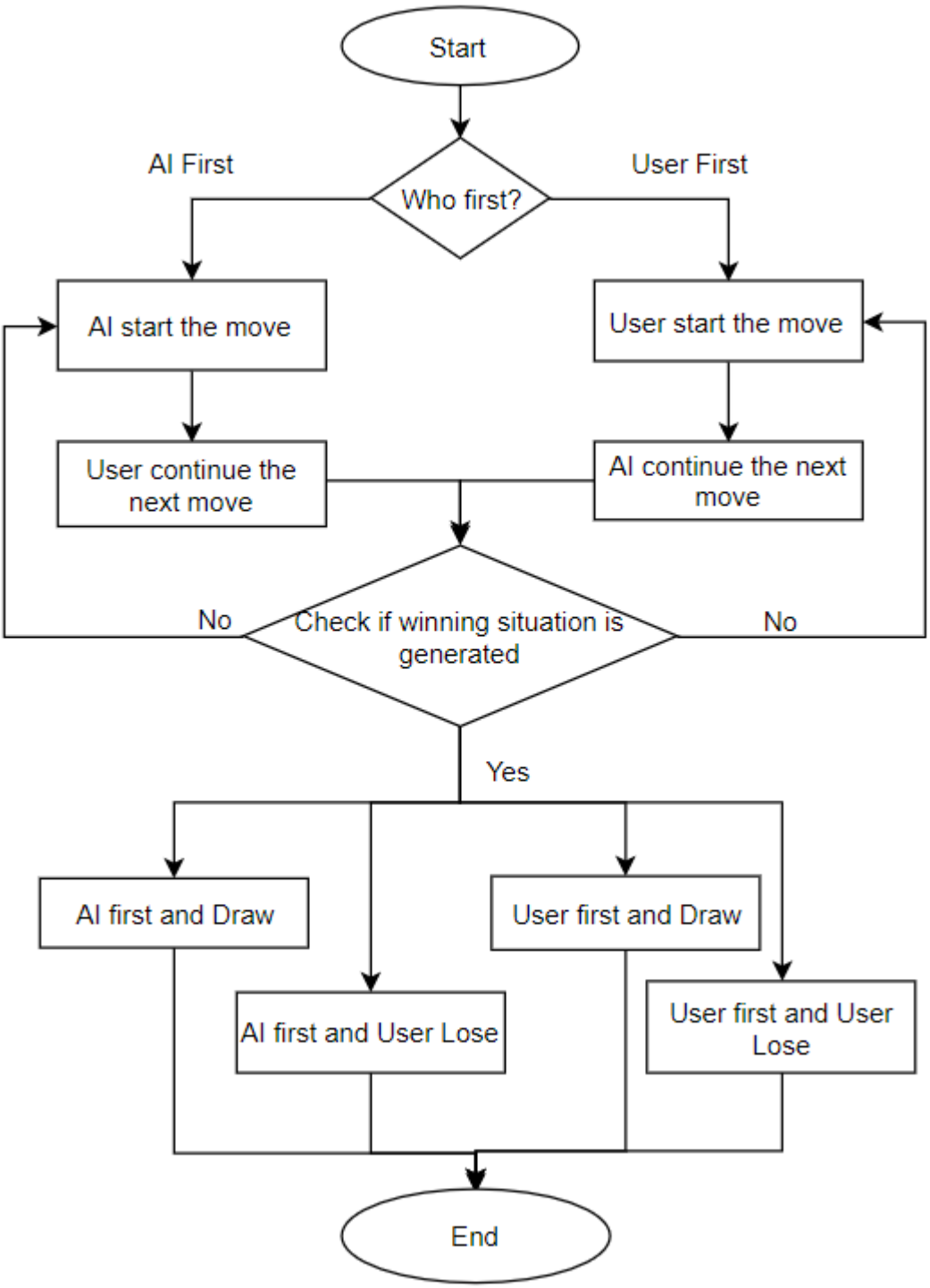



Figure 2: Flowchart of our tic-tac-toe game

The flowchart above will briefly explain the game flow of this tic-tac-toe game we created using visual studio 2019. After starting the game, we will prompt out a menu and let the user pick whether he/she wants to be the first to make a move or let the AI to make a move

first. Either one side makes a move first, the next turn will be for another side to make a move. Then, the winning situation is checked after both sides had made their move. If the winning condition is not generated, the game will continue with the same sequences as before. If the winning condition is generated, the final result is shown to the user. Then, the user will be prompted again whether he/she wants to restart the game or exit the game.

## 5. Artificial Intelligence Heuristic Used

### Minimax algorithm

Minimax algorithm is a kind of backtracking algorithm that is used in decision theory and game theory. It uses game theory, decision theory, statistics and philosophy to find the optimal move for a player, assuming that the opponent also plays optimally. It is commonly being used in two-player turn-based games such as Tic-Tac-Toe, Chess, etc.

The player needs to fulfill two conditions in order to win a game. First, he needs to maximize his own chance of a win. There are two methods to maximize profit: a fork or win. A fork is an opportunity where he can win in two ways. A win is when there are two same 'X' or 'O' of him in a row, then he only needs to play the third one to get three same sign at a row. Second, he needs to minimize the opponent's winning chance. To minimize loss, he can block his opponent if two 'X' or 'O' of the opponent are in a row, or bock opponent's fork.

The principle of the minimax search algorithm is to find the optimal path to minimize the maximum possible loss. As illustrated in Diagram 2, suppose that there are only one or two possible moves per player in each turn. +∞ shows if the computer wins, and -∞ shows if the computer loses. The AI or computer (or the maximizing player) is represented by circles while the opponent (or the minimizing player) is represented by squares. The lookahead is limited to four moves.

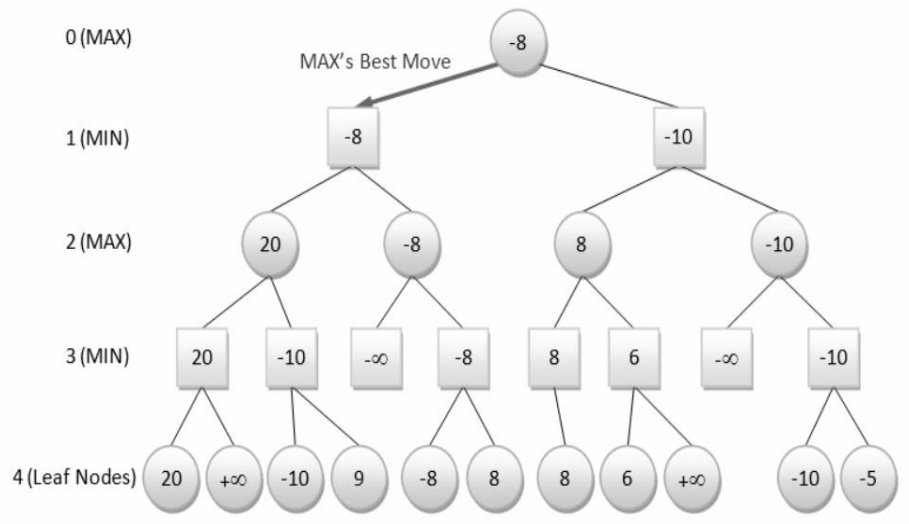



Figure 3: Illustration of the minimax algorithm

The algorithm evaluates the leaf nodes (terminating nodes or at a maximum depth of 4) using the heuristic evaluation function. At level 3, the minimizing player will choose, for each node, the minimum of its children. In level 2, the maximizing player chooses the maximum of the children. The algorithm continues evaluating the maximum and minimum values of the

child nodes alternately until it reaches the root node, where it chooses the move with the maximum value. This is the move that the player should make in order to minimize the maximum possible loss.

Another example of the minimax algorithm in the tic-tac-toe game is shown in Diagram 3 below. Suppose that O wants to calculate the minimax value of the action that leads it to the state at level 0 in the diagram below. At level 0, O wants to calculate the minimax value of that state. So, O has to generate all the possible moves X can make. So the states in level 1 are all the state X can reach through all his possible moves. At level 1, O thinks about the moves it can make in response to each of X's moves. So the states in level 2 are all the state O can reach from there by all possible actions. At level 2, O considers all of X's moves he can take from there, so it generates all the states that X can reach through all his possible actions (The Terminal States). After that, O starts climbing the tree up to the root state, which is the state we started off to calculate its minimax value. At level 2, O knows that X is the one playing next at this level. It also knows that at each possible state at this level, X will choose to go to the child state (at Level 3) with the largest score. So O backs up the minimax value of each state at Level 2 with the maximum score of its child states. At level 1, O knows that it's its turn to play at this level. It will choose a child state (from Level 2) that makes X's score as low as possible. So O backs up the minimax value of each state at level 1 with the minimum minimax value of its child states. At level 0, O follows the same reasoning it followed at level 2. It backs up the minimax value of the root state with the maximum minimax value of its child states (the ones at level 1). The algorithm now terminates and returns the minimax value of the desired state to be 0.

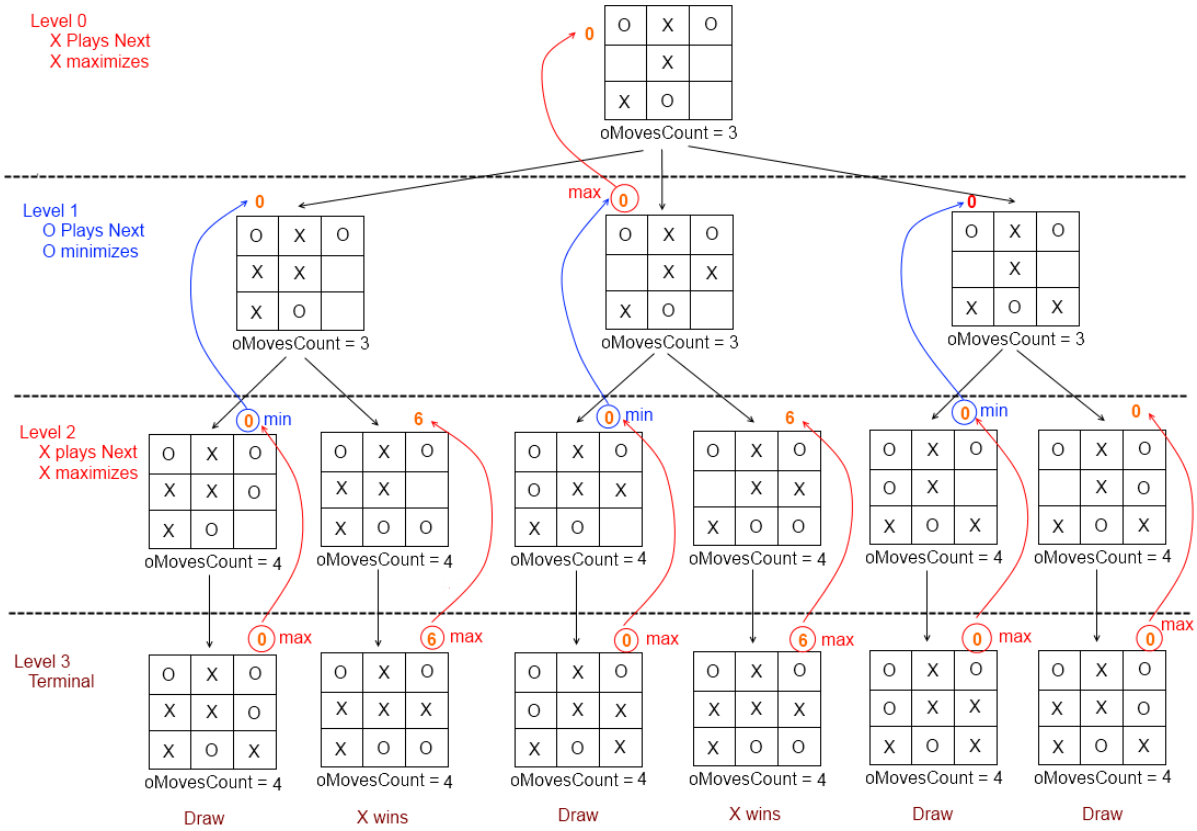



Figure 4: Example of the Minimax Algorithm in Tic-Tac-Toe

**Alpha-Beta Pruning**

Alpha-Beta Pruning algorithm is an optimization algorithm for the minimax algorithm. It reduces the computational time by a huge factor. It allows faster search and even goes into deeper levels in the game tree. It will cut off branches of game trees that need not be searched when there is a better move that exists. Alpha-beta pruning seeks to reduce the number of nodes that needs to be evaluated in the search tree by the minimax algorithm. For the example illustrated in Diagram 4, in the alpha cut-off, node C (MIN) cannot be more than 1 since node D returns 1. Since node B is 4 so there is no need to search the other children of node C, as node A will certainly pick node B over node C for the max node. As illustrated in Diagram 3, the remaining children can be aborted if alpha ≥ beta, for both the alpha cut-off and beta cut-off.

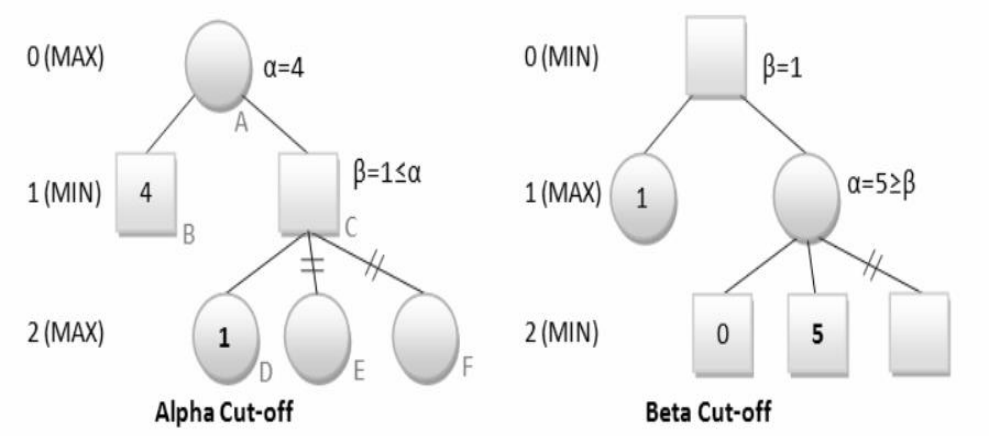



Figure 5: Illustration of the Alpha-Beta Pruning Algorithm

## 6. Source Code

```
#include "graphics.h"
#include <windows.h>
#include <mmsystem.h>
#include <iostream>
#include <vector>
#include <time.h>
#include <thread>
using namespace std;

int cpu_move_to_row=1, cpu_move_to_col=1;
const int no_of_page = 4;


const string WindowName = "Tic-Tac-Toe";
const string MainBackgroundImageName = "images/EntryBG.jpg";


bool isExit = false;

const int width = 780;
const int height = 780;

//total grid will be 9
const int total_row = 3, total_col = 3;
```

Figure 6: Header and variables used in the program

At the starting of the code, we included all of the libraries and sources we are going to use in the program. We also determine the name, background image, width and height of the playing window. The grid is 3x3 with 3 rows and 3 columns.

### Class Point

```
class Point {
public: int x, y; Point(int _x=0, int _y=0) :x(_x), y(_y) {}
};
class Grid {
public: Point topLeft, bottomRight; char occupiedType;
        Grid(int x1 = 0, int y1 = 0, int x2 = 0, int y2 = 0) :topLeft(Point(x1, y1)), bottomRight(Point(x2, y2))
        {occupiedType = '*'; }
        Grid(Point TL, Point BR) : topLeft(TL), bottomRight(BR) { occupiedType = '*'; }
        int getWidth() { return abs(topLeft.x - bottomRight.x); }
        int getHeight() { return abs(topLeft.y - bottomRight.y); }
};
```

Figure 7: Class Point and Class Grid

This class is used to determine the coordinates of points in the grid so that we can input the ‘X’ or ‘O’ symbol in the correct coordinates.

**drawGrid Function**

```
Grid grid[total_row][total_col];
void drawGrid(int width, int height, int margin)
{
    //draw 3 vertical line
    setcolor(WHITE);
    setlinestyle(0, 0, 3);

    int row_width = (width - margin * 2) / 3;
    int col_width = (height - margin * 2) / 3;

    //draw 4 horizontal line
    line(margin, margin, width - margin, margin);

    line(margin, margin + row_width, width - margin, margin + row_width);

    line(margin, margin + row_width * 2, width - margin, margin + row_width * 2);

    line(margin, height - margin, width - margin, height - margin);


    //draw 4 vertical line
    line(margin, margin, margin, height - margin);

    line(margin + col_width, margin, margin + col_width, height - margin);

    line(margin + col_width * 2, margin, margin + col_width * 2, height - margin);

    line(width - margin, margin, width - margin, height - margin);

    for (int i = 0; i < total_row; i++)
        for (int j = 0; j < total_col; j++)
            grid[i][j] = Grid(margin + j * col_width, margin +
                i * row_width, margin + (j+1) * col_width, margin + (i+1) * row_width);
}
```

Figure 8: drawGrid function

Here we set the row width and column width for the 3x3 grid. Later, we use a white color line to draw 4 horizontal lines and 4 vertical lines which later will form the outline of the grid. We also determine the top left and bottom right points for each of the elements in the grid so we can detect the region form of the grid elements.

**Me_won Function**

```
#pragma region Full Condition
int isFull()// grid is full
{
    for (int i=0; i<total_row; i++)
        for (int j=0; j<total_col; j++)
            if (grid[i][j].occupiedType!='X')
                if(grid[i][j].occupiedType != 'O')
                    return 0;
    return 1;
}
#pragma endregion


#pragma region WinningCondition
int me_won()
{
    //horizontal win condition
    if (grid[0][0].occupiedType == 'O' && grid[0][1].occupiedType == 'O' && grid[0][2].occupiedType == 'O')
        return 1;
    if (grid[1][0].occupiedType == 'O' && grid[1][1].occupiedType == 'O' && grid[1][2].occupiedType == 'O')
        return 1;
    if (grid[2][0].occupiedType == 'O' && grid[2][1].occupiedType == 'O' && grid[2][2].occupiedType == 'O')
        return 1;

    //vertical win condition
    if (grid[0][0].occupiedType == 'O' && grid[1][0].occupiedType == 'O' && grid[2][0].occupiedType == 'O')
        return 1;
    if (grid[0][1].occupiedType == 'O' && grid[1][1].occupiedType == 'O' && grid[2][1].occupiedType == 'O')
        return 1;
    if (grid[0][2].occupiedType == 'O' && grid[1][2].occupiedType == 'O' && grid[2][2].occupiedType == 'O')
        return 1;

    //sliding win condition
    if (grid[0][0].occupiedType == 'O' && grid[1][1].occupiedType == 'O' && grid[2][2].occupiedType == 'O')
        return 1;
    if (grid[0][2].occupiedType == 'O' && grid[1][1].occupiedType == 'O' && grid[2][0].occupiedType == 'O')
        return 1;
    return 0;
}
```

Figure 9: isFull and me_won function

The isFull() function is used to determine the condition of the grid. If the grid is fully filled, it will return 1.

The me_won() function is used to determine the winning condition of the user. The user is using the symbol ‘O’. There are 3 possible winning conditions for the user:

Horizontal win condition means there are three continuous ‘O’ in a horizontal line either in row 0, row 1 or row 2.

Vertical win condition means there are three continuous ‘O’ in a vertical line either in column 0, column 1 or column 2.

Sliding win condition means there are three continuous ‘O’ in a diagonal way.

If any one of the conditions is fulfilled, the user is won.

### Cpu_won Function

```
int cpu_won()
{
    //horizontal win condition
    if (grid[0][0].occupiedType == 'X' && grid[0][1].occupiedType == 'X' && grid[0][2].occupiedType == 'X')
        return 1;
    if (grid[1][0].occupiedType == 'X' && grid[1][1].occupiedType == 'X' && grid[1][2].occupiedType == 'X')
        return 1;
    if (grid[2][0].occupiedType == 'X' && grid[2][1].occupiedType == 'X' && grid[2][2].occupiedType == 'X')
        return 1;

    //vertical win condition
    if (grid[0][0].occupiedType == 'X' && grid[1][0].occupiedType == 'X' && grid[2][0].occupiedType == 'X')
        return 1;
    if (grid[0][1].occupiedType == 'X' && grid[1][1].occupiedType == 'X' && grid[2][1].occupiedType == 'X')
        return 1;
    if (grid[0][2].occupiedType == 'X' && grid[1][2].occupiedType == 'X' && grid[2][2].occupiedType == 'X')
        return 1;

    //sliding win condition
    if (grid[0][0].occupiedType == 'X' && grid[1][1].occupiedType == 'X' && grid[2][2].occupiedType == 'X')
        return 1;
    if (grid[0][2].occupiedType == 'X' && grid[1][1].occupiedType == 'X' && grid[2][0].occupiedType == 'X')
        return 1;

    return 0;
}
#pragma endregion
```

Figure 10: cpu_won function

The cpu_won() function is used to determine the winning condition of the computer AI. The AI is using the symbol ‘X’. There are 3 possible winning conditions for the AI:

Horizontal win condition means there are three continuous ‘X’ in a horizontal line either in row 0, row 1 or row 2.

Vertical win condition means there are three continuous ‘X’ in a vertical line either in column 0, column 1 or column 2.

Sliding win condition means there are three continuous ‘X’ in a diagonal way.

If any one of the conditions is fulfilled, the AI is won.

### Tic-Tac-Toe Game: A Zero-Sum Game

Tic-Tac-Toe is a game of a special kind, a kind called zero-sum games. In this type of game, the scores of all players’ sum to 0, which means that in a two-player game the score of one player is negative of the score of the other player. If player 1 gets a score of 5 then AI gets a score of -5, making the sum of all scores to be 5 + (-5) = 0, hence: a zero-sum game. Zero-sum games are pure competition games, there’s no cooperation of any kind between the players.

Because Tic-Tac-Toe is a zero-sum game, the AI can spend all its life minimizing User’s score and at the same time be maximizing its score.

**Minimax Function**

```
//here is the minimax for computer brain
int minimax(bool flag)// The minimax function
{

    int max_val = -1000, min_val = 1000;
    //-100 is our assumed infinite max value, 100 is our assumed infinite min value
    int i, j, value = 1;
    if (cpu_won() == 1)
    {
        return 10;
    }
    else if (me_won() == 1)
    {
        return -10;
    }
    else if (isFull() == 1)
    {
        return 0;
    }

    int score[total_row][total_col] = { {1,1,1}, {1,1,1}, {1,1,1} };//if score[i]=1 then it is empty

    for (i = 0; i < total_row; i++)
        for (j=0; j<total_col; j++)
        {
            if (grid[i][j].occupiedType == '*')
            {
                if (min_val > max_val) // reverse of pruning condition
                {
                    if (flag == true)
                    {
                        grid[i][j].occupiedType = 'X';
                        value = minimax(false);
                    }
                    else
                    {
                        grid[i][j].occupiedType = 'O';
                        value = minimax(true);
                    }
                    grid[i][j].occupiedType = '*';
                    score[j][i] = value;
                }
            }
        }
```

Figure 11: minimax function part 1

We now take the first step in implementing our AI, which is implementing the score concept.

The score concept is the way the AI can know the benefit of a specific action it can take.

Here is the score concept:

SCORE = 10 (If result = AI won)
-10 (If result = User won)
0 (If result = Draw)

Also, we test it when the grid is being occupied using the pruning condition.

```
        if (flag == true)
        {
            max_val = -1000;
            for (int i = 0; i < total_row; i++)
                for (int j = 0; j < total_col; j++)
                {
                    if (score[j][i] > max_val && score[j][i] != 1)
                    {
                        max_val = score[j][i];
                        cpu_move_to_row = i;
                        cpu_move_to_col = j;
                    }
                }
            return max_val;
        }
        if (flag == false)
        {

            min_val = 1000;

            for (i=0; i<total_row; i++)
                for (j = 0; j < total_col; j++)
                {
                    if (score[j][i] < min_val && score[j][i] != 1)
                    {
                        min_val = score[j][i];
                        cpu_move_to_row = i;
                        cpu_move_to_col = j;
                    }
                }
            return min_val;
        }
}
```

Figure 12: minimax function part 2

Now the AI needs a way in making an optimal decision, and this is where minimax comes to help. The Basic intuition behind the minimax decision algorithm is that at a given state, the AI will think about the possible moves it can make, and about the moves the opponent can make after that. This process continues until terminal states are reached which is the grid is fully filled (not in the actual game, but in the AI's thought process). The AI then chooses the action that leads to the best possible terminal state according to its score.

The algorithm is used to calculate the minimax value of a specific state (or the action that leads to that state), and it works by knowing that someone wants to minimize the score function and the other wants to maximize the score function (and that's why it's called minimax).

The minimax algorithm is a recursive algorithm and its base case is reaching a terminal state. We can implement it with a recursive function that recurs down to the terminal states and backs up the minimax value as the recursion unwinds.

**Class Player**

```
class Player{
public:
    bool hasMoved;
    Player() { hasMoved = false; }
    void Control()
    {
        int x, y;
        getmouseclick(WM_LBUTTONDOWN, x, y);

        int crop_image_to_offset = 5;  //we need to reduce the image size becoz it hide our lines.

        for (int i = 0; i < total_row; i++) {
            for (int j = 0; j < total_col; j++)
            {
                //first, check for a valid grid
                if (grid[i][j].occupiedType == '*') {
                    if (x >= grid[i][j].topLeft.x
                        && x <= grid[i][j].bottomRight.x
                        && y >= grid[i][j].topLeft.y
                        && y <= grid[i][j].bottomRight.y
                        )
                    {

                        cout << endl << endl;
                        cout << "\nYou press grid [" << i << "][" << j << "]\n";
                        readimagefile("images/O.jpg", grid[i][j].topLeft.x + crop_image_to_offset
                            , grid[i][j].topLeft.y + crop_image_to_offset
                            , grid[i][j].bottomRight.x - crop_image_to_offset
                            , grid[i][j].bottomRight.y - crop_image_to_offset);

                        grid[i][j].occupiedType = 'O';

                        PlaySound("sounds/Button.wav", NULL, SND_FILENAME);
                        hasMoved = true;
                        return;
                    }
                }
                else
                {
                    hasMoved = false;
                }

            }
        }
    }
};
```

Figure 13: class Player

This class is used to detect whether the user has used a mouse click to click the valid regions in any 9 regions in the grid. If the user does so, the function is able to determine which region does the user clicked and put an ‘O’ image in the region the user clicked. A sound effect will be played each time the user clicked the valid region. Game cannot be proceed until the user make a move.

**Class Computer**

```
class Computer {
public:
    Computer() { }
    void Control()
    {
        int crop_image_to_offset = 5;
            cout<<"\nMinimax: "<<minimax(true);
            if (grid[cpu_move_to_row][cpu_move_to_col].occupiedType == '*')
            {
                grid[cpu_move_to_row][cpu_move_to_col].occupiedType = 'X';
                readimagefile("images/X.jpg", grid[cpu_move_to_row]
                    [cpu_move_to_col].topLeft.x + crop_image_to_offset
                    , grid[cpu_move_to_row][cpu_move_to_col].topLeft.y
                    + crop_image_to_offset
                    , grid[cpu_move_to_row][cpu_move_to_col].bottomRight.x
                    - crop_image_to_offset
                    , grid[cpu_move_to_row][cpu_move_to_col].bottomRight.y
                    - crop_image_to_offset);

                cout << "\nCPU select grid[" << cpu_move_to_row
                    << "][" << cpu_move_to_col << "]";


                PlaySound("sounds/CPUSelect.wav", NULL, SND_FILENAME);
            }
    }
};
```

Figure 14: class Computer

This class is used to put a ‘X’ image in the region the AI calculated based on previous minimax function. A sound effect will be played each time AI places a move.

**Class PageController**

```
class PageController {
private:
    bool isMeFirst;
    int currPageIndex;
public:
    PageController()
    {
        currPageIndex = 0;
        srand(time(NULL));
        cpu_move_to_row = rand() % total_row;
        cpu_move_to_col = rand() % total_col;
        isMeFirst = false;
    }

    void switchPage()
    {
        currPageIndex = (currPageIndex + 1) % no_of_page;
    }

    void displayCurrentPage()
    {
        cout << "Displaying page: " << currPageIndex<<endl;

        DisplayPage(currPageIndex);
    }

    void DisplayPage(int index)
    {

            switch (index)
            {
            case 0:
            {
                EntryPageScene(width, height);
                break;
            }
            case 1:
            {
                SelectFirstPageScene(width, height);
                break;
            }
            case 2:
            {

                PlayPageScene(width, height);
                break;
            }
            case 3:
                ResultPageScene(width, height);
                break;
            }
    }
```

Figure 15: class PageController

This function is used to control the page display in window. It uses a case switch function with the parameter index representing different button pressed. Once the specific button is pressed, this function will link users to a different page.

**EntryPageScene Function**

```
void EntryPageScene(int width, int height)
{
    int marginX = 100;
    readimagefile(MainBackgroundImageName.c_str(), 0, 0, width, height / 2);

    readimagefile("images/playBtn.jpg", width / 2 - marginX,
        height / 12*7, width / 2 + marginX, height / 12 *8);
    readimagefile("images/exitBtn.jpg", width / 2 - marginX,
        height / 12*10 , width / 2 + marginX, height / 12*11);
    setmousequeuestatus(WM_LBUTTONDOWN);

//while (!ismouseclick(WM_LBUTTONDOWN)) { /*wait for press*/}
    while (1)
    {
        int x, y;
        getmouseclick(WM_LBUTTONDOWN, x, y);

        if (x >= width / 2 - marginX
            && x <= width / 2 + marginX
            && y >= height / 12 * 7
            && y <= height / 12 * 8)
        {
            PlaySound("sounds/Button.wav", NULL, SND_FILENAME);

            cleardevice();
            switchPage();
            break;
        }
        if (x >= width / 2 - marginX
            && x <= width / 2 + marginX
            && y >= height / 12 * 10
            && y <= height / 12 * 11)
        {
            PlaySound("sounds/Button.wav", NULL, SND_FILENAME);

            cleardevice();
            closegraph();
            isExit = true;
            break;
        }

    }
}
```

Figure 16: EntryPageScene function

Here is the coding for the entry page scene. First, it loads the background image and background music for the scene and set two buttons: PLAY and EXIT button on the scene. If user clicked on any of the button, this function will detect the mouse click and link users to different pages. Also, a sound effect will be played each time user clicked on the button.

**SelectFirstPageScene Function**

```
void SelectFirstPageScene(int width, int height)
{
    mciSendString("play sounds/bgm.wav", NULL, 0, 0);
    int buttonWidth = 200;


    readimagefile("images/WhoFirstImg.jpg", width / 2 - buttonWidth, (1 / 12.0)
        * height, width / 2 + buttonWidth, (3 / 12.0) * height);
    readimagefile("images/computerBtn.jpg", width / 2 - buttonWidth, (5 / 12.0)
        * height, width / 2 + buttonWidth, (7 / 12.0) * height);
    readimagefile("images/meBtn.jpg", width / 2 - buttonWidth, (8 / 12.0)
        * height, width / 2 + buttonWidth, (10 / 12.0) * height);


    while (1)
    {
        int x, y;
        getmouseclick(WM_LBUTTONDOWN, x, y);

        if (x >= width / 2 - buttonWidth
            && x <= width / 2 + buttonWidth
            && y >= (5 / 12.0) * height
            && y <= (7 / 12.0) * height)
        {
            PlaySound("sounds/Button.wav", NULL, SND_FILENAME);

            cleardevice();
            switchPage();
            isMeFirst = false;
            break;
        }

        if (x >= width / 2 - buttonWidth
            && x <= width / 2 + buttonWidth
            && y >= (8 / 12.0) * height
            && y <= (10 / 12.0) * height)
        {
            PlaySound("sounds/Button.wav", NULL, SND_FILENAME);

            cleardevice();
            switchPage();
            isMeFirst = true;
            break;
        }

    }
}
```

Figure 17: SelectFirstPageScene function

Here is the coding for the select first page scene. The user needs to decide who will be the first turn in the game here. First, it loads the background image and background music for the scene and set two buttons: COMPUTER and ME button on the scene. If user clicked on any of the button, this function will detect the mouse click and link users to different pages. Also, a sound effect will be played each time user clicked on the button.

**PlayPageScene Function**

```
void PlayPageScene(int width, int height)
{
    drawGrid(width, height, 100);
    int font_size = 1;
    settextstyle(10, HORIZ_DIR, font_size);
    char cpu_say[] = "Plese do not click too fast.";
    outtextxy(20, height-60, cpu_say);
    delay(1000);
    char me_reply[] = "Me: okay...";
    outtextxy(20, height - 40, me_reply);

    Player me;
    Computer enemy;

    int isMyTurn = false;

    cout << "\n\nMe first? " << ((isMeFirst) ? "\nYa, me first\n\n" : "\nNo, cpu first\n\n")<<endl;

    cout << "----------Game Start---------\n\n";

    if (isMeFirst)
    {
        while (1)
        {
            me.Control();
            if (me.hasMoved)
            {
                isMyTurn = false;
                break;
            }

        }
    }
```

Figure 18: PlayPageScene function part 1

Here is the coding for the Play Page scene. This function draws a grid in the middle of the scene. Also, it shows a very simple user guideline to tell the user do not click on the screen to fast. At the same time, the console window will display the integer of the player moves.

```
        //looping the game until sb win
        while (1)
        {
            //computer turn
            if (!isMyTurn)
            {
                enemy.Control();
                isMyTurn = true;
            }

            if (cpu_won() || isFull())
            {
                cout << "\nLoading Result...\n\n";

                int font_size = 3;

                settextstyle(10, HORIZ_DIR, 3);
                setcolor(WHITE);
                outtextxy((width - 100) / 2 - 17 * 3, 0, "\nLoading Result...");

                delay(5000);
                cleardevice();
                switchPage();
                break;
            }

            if (isMyTurn)//here is our gameplay
            {
                me.Control();
                if (me.hasMoved)
                {
                    me.hasMoved = false;
                    isMyTurn = false;

                }
            }

            if (me_won()||cpu_won()||isFull())
            {
                settextstyle(10, HORIZ_DIR, 3);
                setcolor(WHITE);
                outtextxy((width-100)/2-17*3, 0, "\nLoading Result...");
                delay(5000);
                cleardevice();
                switchPage();
                break;
            }
        }
    }
```

Figure 19: PlayPageScene function part 2

This part control the game play process. If there is no user's turn, AI will control and place a move and vice versa. Since there are only two results of the game: AI wins or Draw, once it reached the condition, it brings to a Loading Result header and switch to the Result Page scene.

**ResultPageScene Function**

```
void ResultPageScene(int width, int height)
{
    int buttonWidth = 120;
    mciSendString("close sounds/bgm.wav", NULL, 0, 0);

    readimagefile("images/restartBtn.jpg", width / 2 - buttonWidth, (5 / 12.0)
        * height, width / 2 + buttonWidth, (7 / 12.0) * height);
    readimagefile("images/exitBtn.jpg", width / 2 - buttonWidth, (8 / 12.0)
        * height, width / 2 + buttonWidth, (10 / 12.0) * height);

    if (me_won() == 1)
    {
        cout << "\n\n---------Result---------\nMe Win!\n\n";
        readimagefile("images/youWinImg.jpg", 50, (1 / 12.0) * height, width - 50, (3 / 12.0) * height);
        PlaySound("sounds/Win.wav", NULL, SND_FILENAME);
    }
    else if (cpu_won() == 1)
    {

        cout << "\n\n---------Result---------\nCPU Win!\n\n";

        readimagefile("images/youLoseImg.jpg", 50, (1 / 12.0) * height, width - 50, (3 / 12.0) * height);

        PlaySound("sounds/Lose.wav", NULL, SND_FILENAME);
    }
    else if (isFull() == 1)
    {

        cout << "\n\n---------Result---------\nDraw!\n\n";

        readimagefile("images/drawImg.jpg", 50, (1 / 12.0) * height, width - 50, (3 / 12.0) * height);

        PlaySound("sounds/Draw.wav", NULL, SND_FILENAME);
    }
```

Figure 20: ResultPageScene function part 1

Here is the Result Page Scene. It will load a background music and show any of 3 results: User Win(almost impossible to happen), AI WIN or Draw.

```
while (1)
{
    int x, y;
    getmouseclick(WM_LBUTTONDOWN, x, y);

    if (x >= width / 2 - buttonWidth
        && x <= width / 2 + buttonWidth
        && y >= (5 / 12.0) * height
        && y <= (7 / 12.0) * height)
    {
        PlaySound("sounds/Button.wav", NULL, SND_FILENAME);


        cleardevice();
        currPageIndex = 1;
        system("cls");
        break;
    }

    if (x >= width / 2 - buttonWidth
        && x <= width / 2 + buttonWidth
        && y >= (8 / 12.0) * height
        && y <= (10 / 12.0) * height)
    {
        PlaySound("sounds/Button.wav", NULL, SND_FILENAME);

        cleardevice();
        closegraph();
        isExit = true;
        break;
    }



}
```

Figure 21: ResultPageScene function part 2

In the Result Page, there are still two buttons: RESTART and EXIT. This part of the function detect user's mouse click, if user clicked RESTART, it will bring the user to a new game, else user will click EXIT to exit the game and close the program.

**Main Function**

```
int main()
{

    int winPosX = (getmaxwidth() - width) / 2;
    int winPosY = (getmaxheight() - height) / 2;


    //centre our window at the middle
    initwindow(width, height, WindowName.c_str(), winPosX, winPosY);

    PageController c;

    while (!isExit)
    {
        c.displayCurrentPage();
    }
    return 0;
}
```

Figure 22: main function

Here is the main function of the program. It will create a game window with set width and height. It also allows other game playing functions.

## 7. User Manual

### 7.1 Open the game to play only

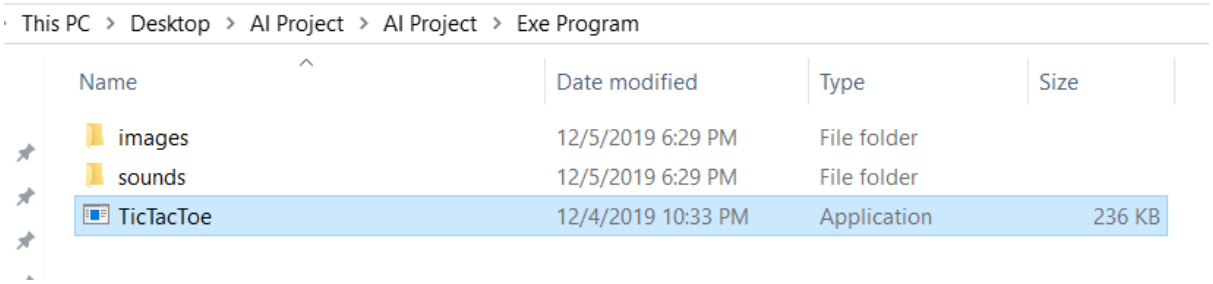



Figure 23: Exe Program File

1. User needs to download and open the TicTacToe.exe in the Exe Program File.

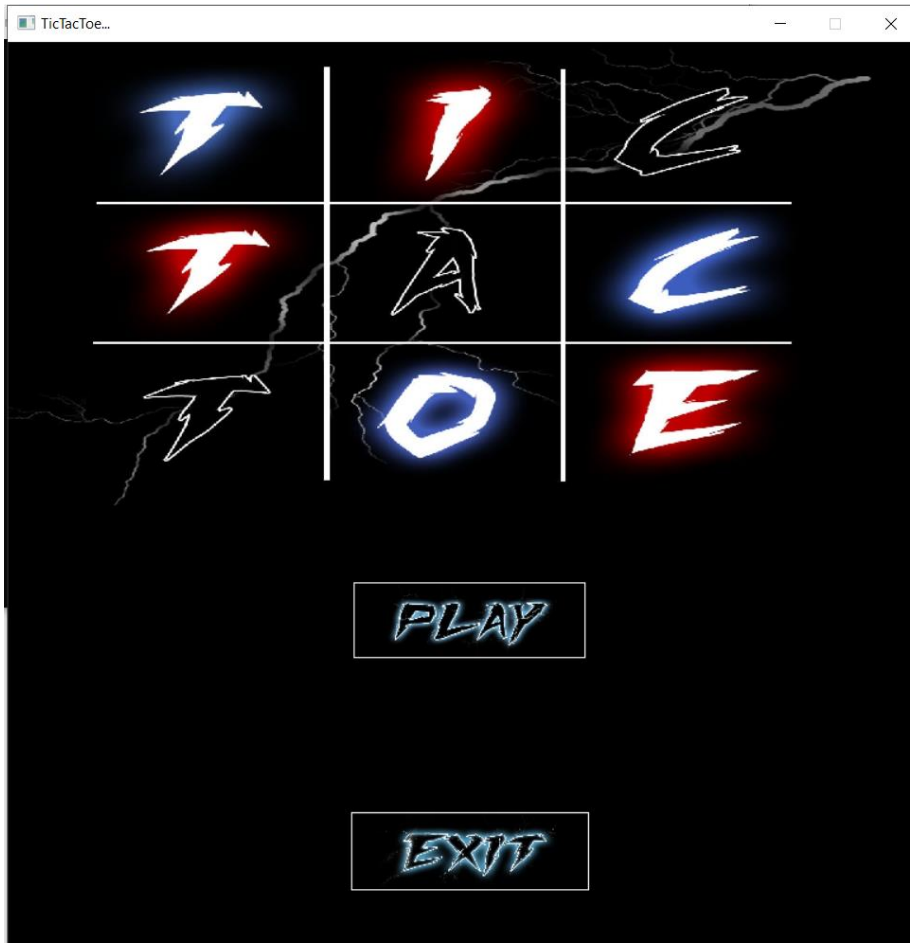



Figure 24: Main Menu

2. Now user can play the game in the game window.

### 7.2 Open the program to view code

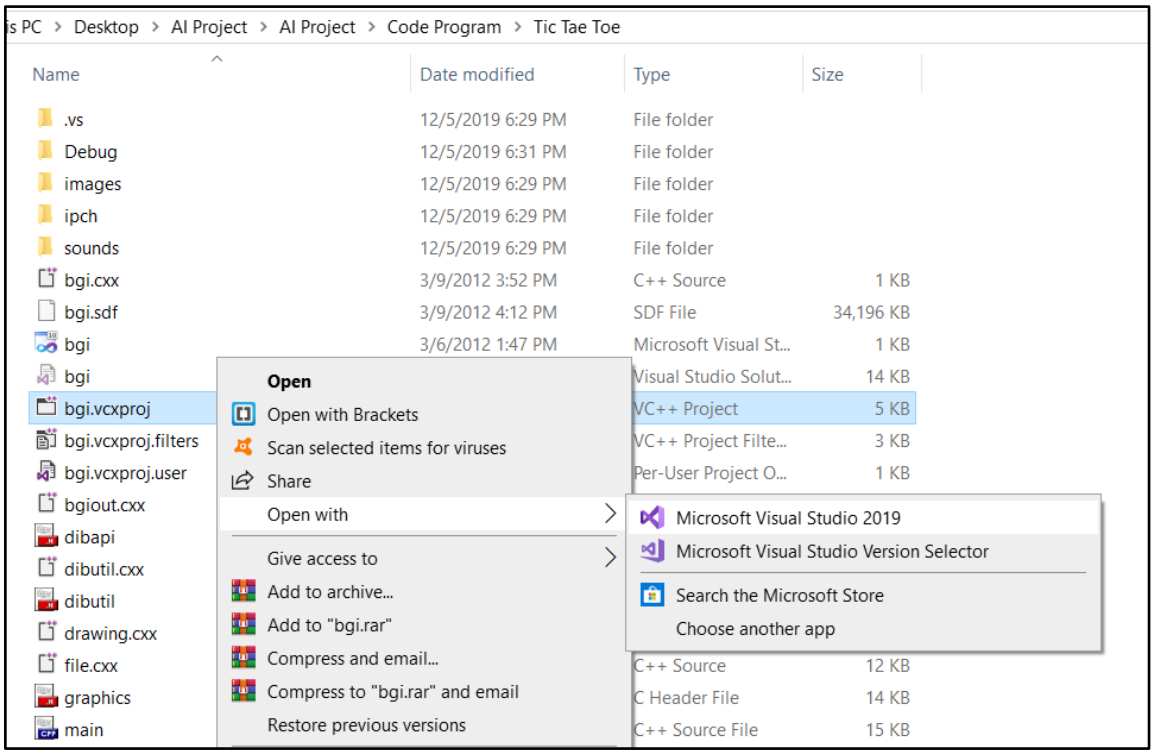



Figure 25: Tic Tae Toe file

1. First, the user needs to download and open the program file (bgi.vcxproj) by using Microsoft Visual Studio 2019.

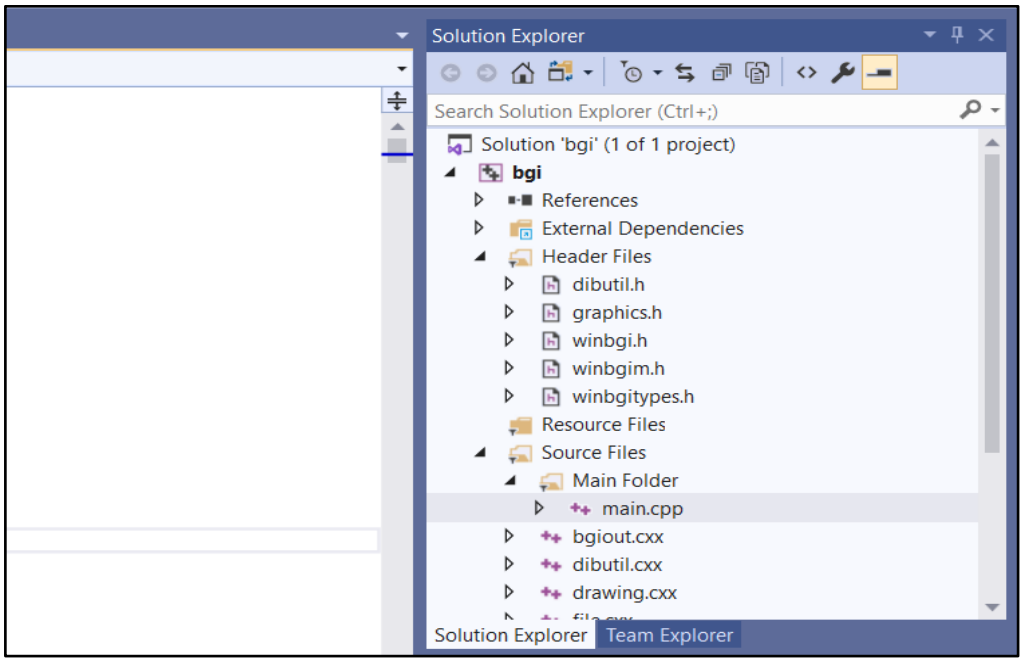



Figure 26: Solution Explorer in Microsoft Visual Studio 2019

2. Then, in Visual Studio 2019, on the Solution Explorer panel, user needs to open the main.cpp in the Main Folder.

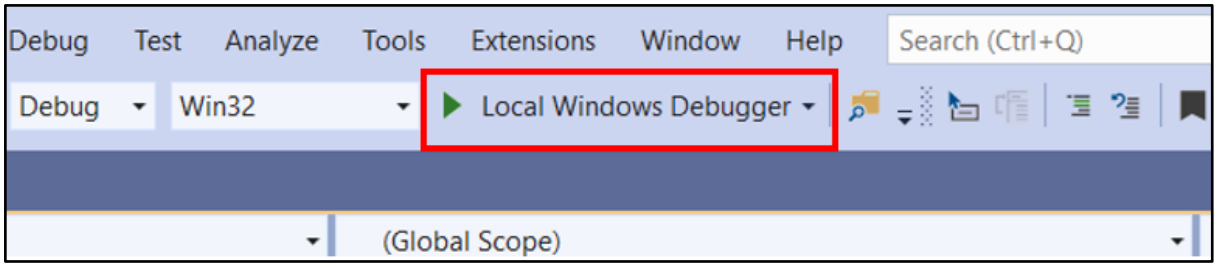



Figure 27: Local Windows Debugger in Microsoft Visual Studio 2019

3. The user needs to click on Local Windows Debugger and start to debug the main.cpp.

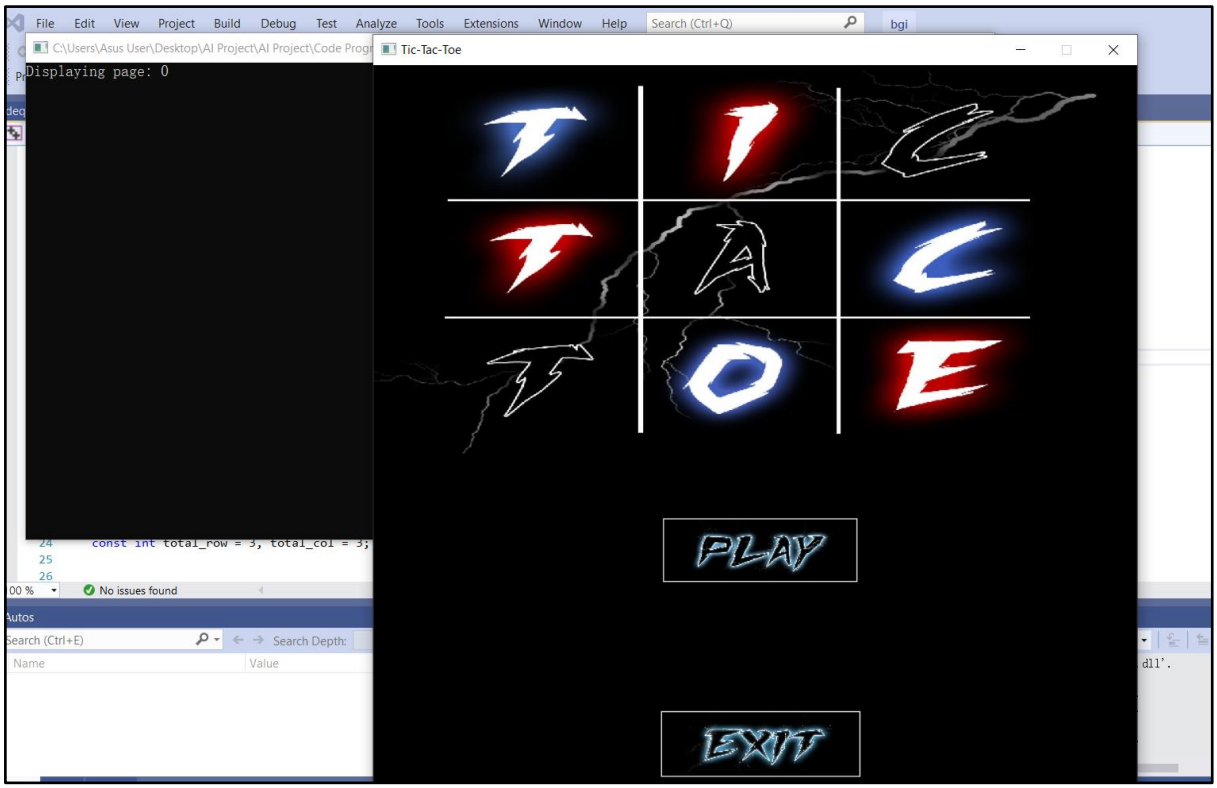



Figure 28: Console and Exe run in Microsoft Visual Studio 2019

4. The interface of the game will appear and the user can start to play the game.

### 7.3 Game Interface

**Main Page**

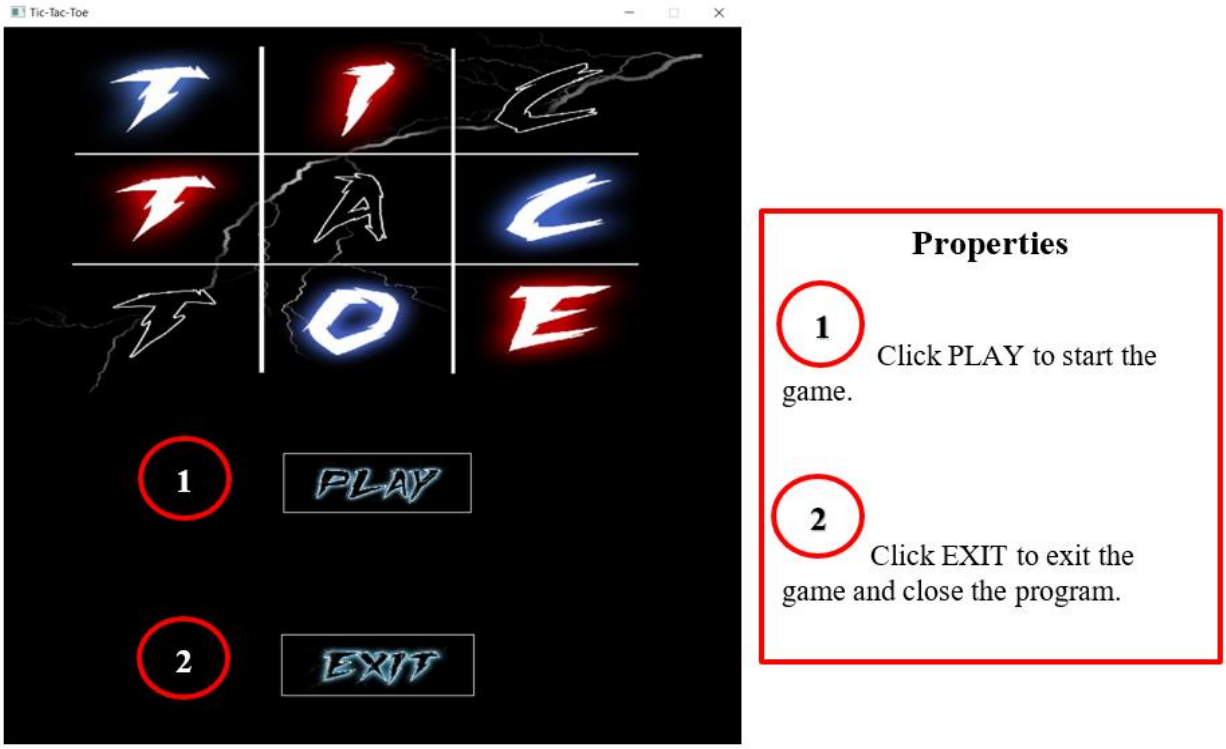



Figure 29: Main Interface

**If PLAY is clicked - The Game will start**

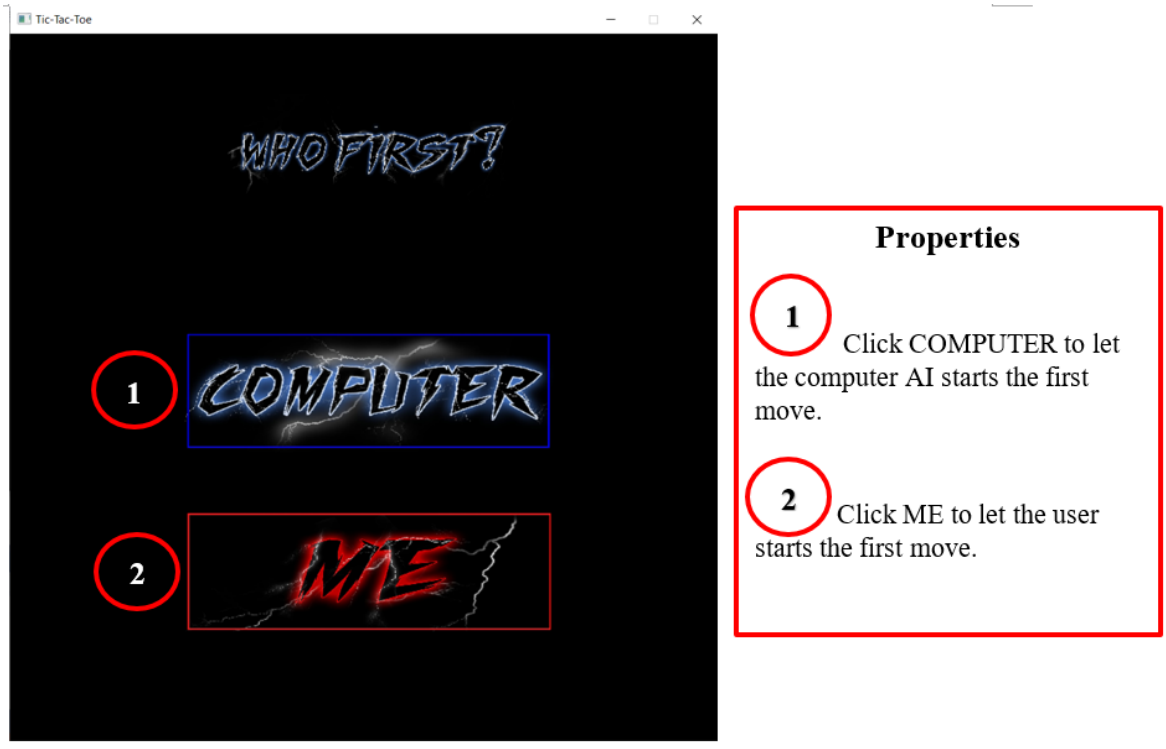



Figure 30: Who First Interface

In the Tic-Tac-Toe game, the players can fill the nine spaces with any 2 different symbols to differentiate both teams and the commonly used symbols are crosses (‘X’) and noughts (‘O’). In our game, the AI will hold a ‘X’ and user will hold ‘O’.

Here are the rules, in the order of importance, are:

| Rules | Description |
|---|---|
| Rule 1 | If the player has a winning move, take it. |
| Rule 2 | If the opponent has a winning move, block it. |
| Rule 3 | If the player can create a fork(two winning ways) after this move, take it. |
| Rule 4 | Do not let the opponent create a fork after the player’s move. |
| Rule 5 | Move in a way such as the player may win the most number of possible ways. |

Figure 31: Rules and Description Table

The winning condition for this game is placing three similar marks in horizontal, vertical or diagonal rows.

**If COMPUTER is clicked**

**Computer AI will start the first move**

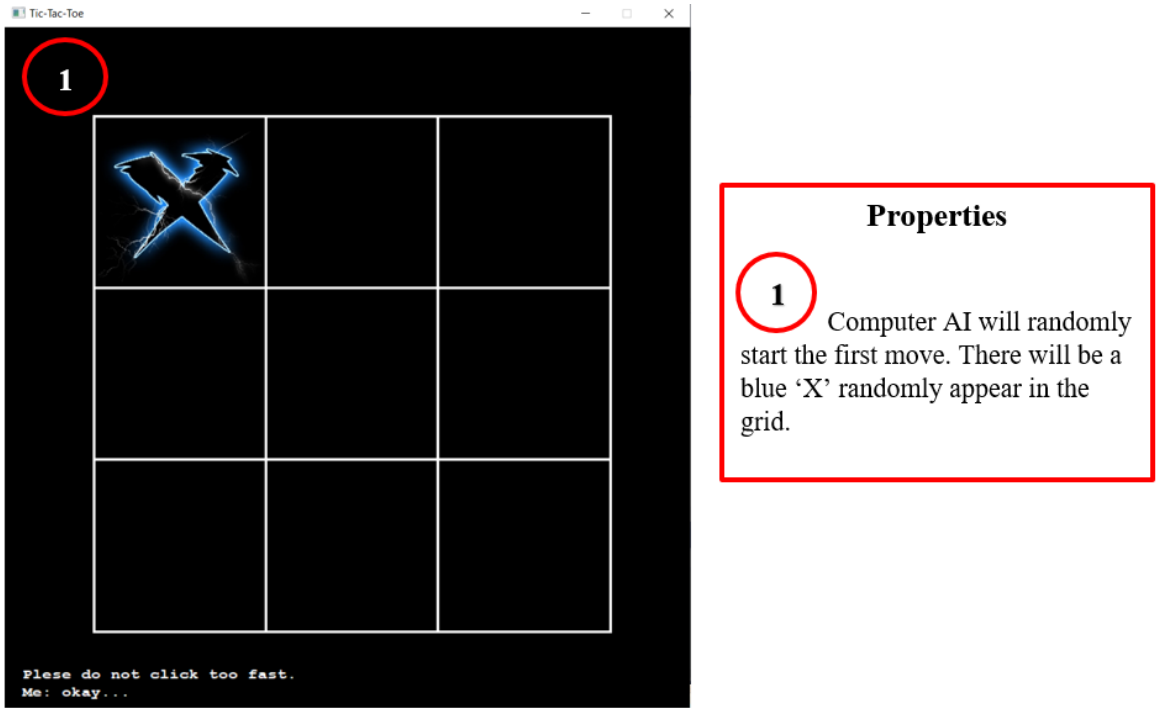



Figure 32: Playing Scene 1

**User clicks the second move and AI continues the third move**

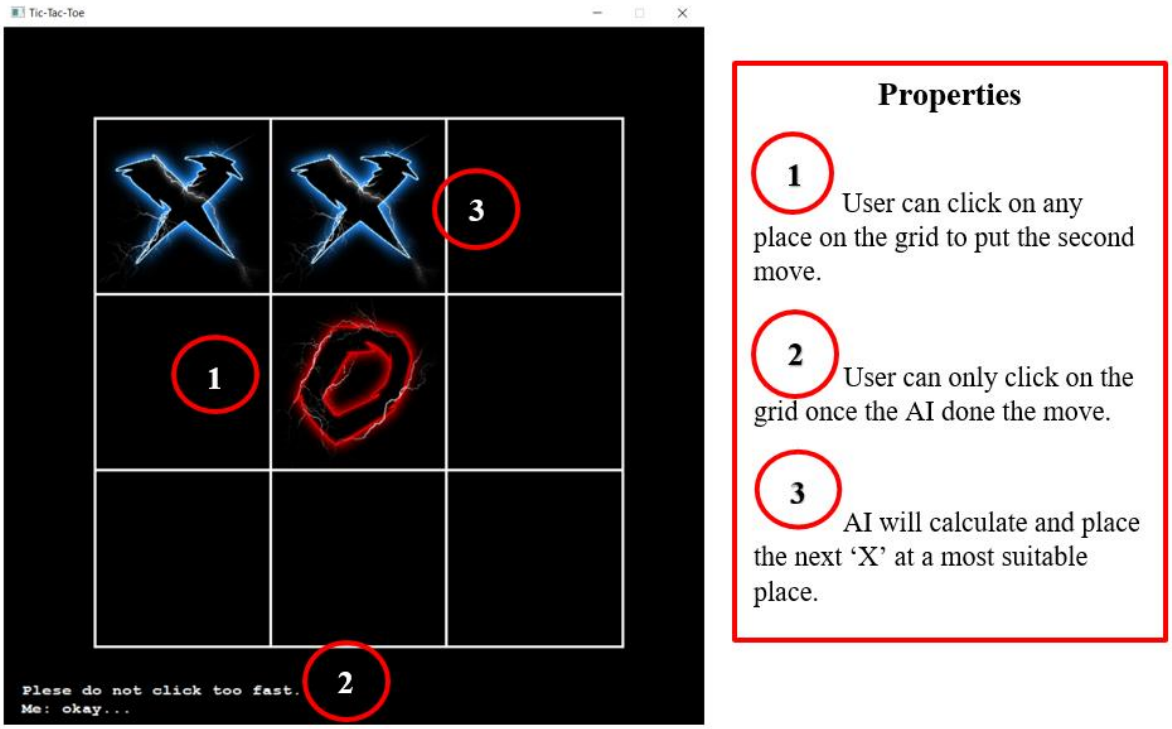



Figure 33: Playing Scene 2

**User blocks the winning path of opponent(AI)**

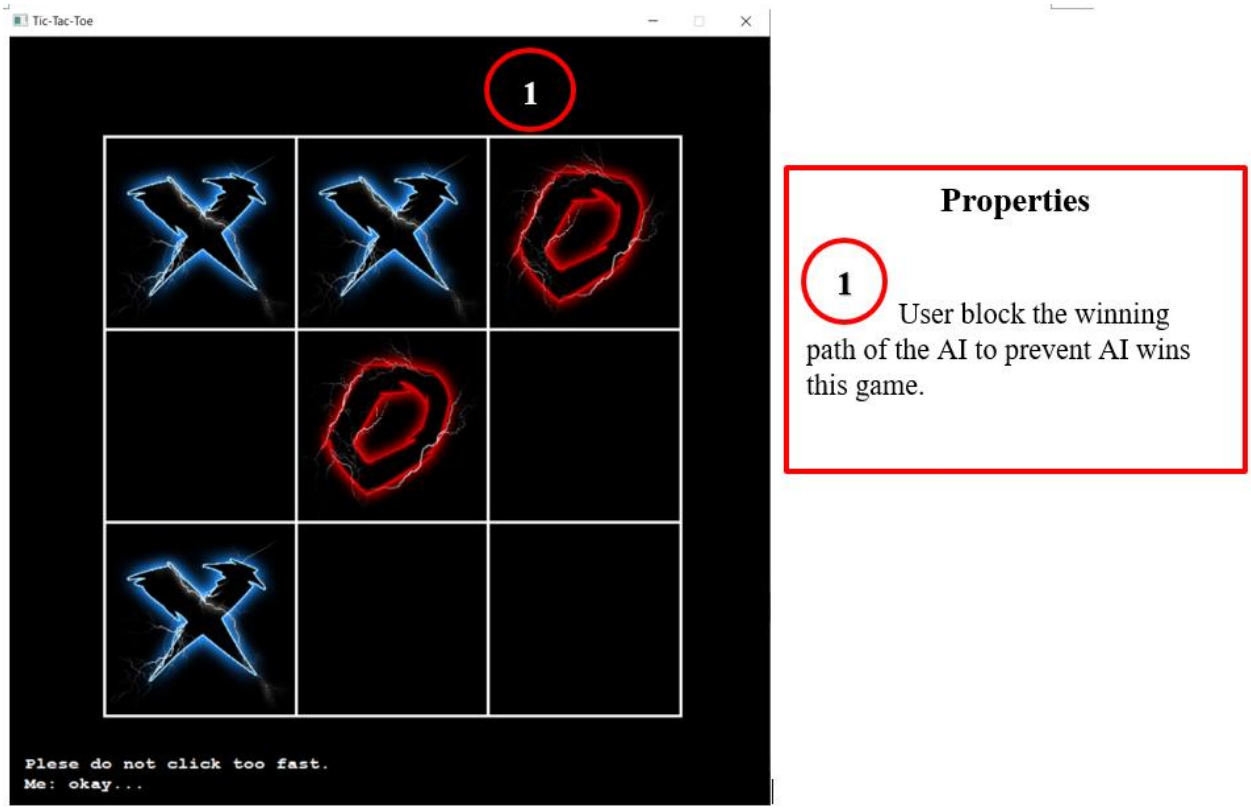



Figure 34: Playing Scene 3

**Loading Result after all the move**

**Condition 1: AI First & A Draw Result**

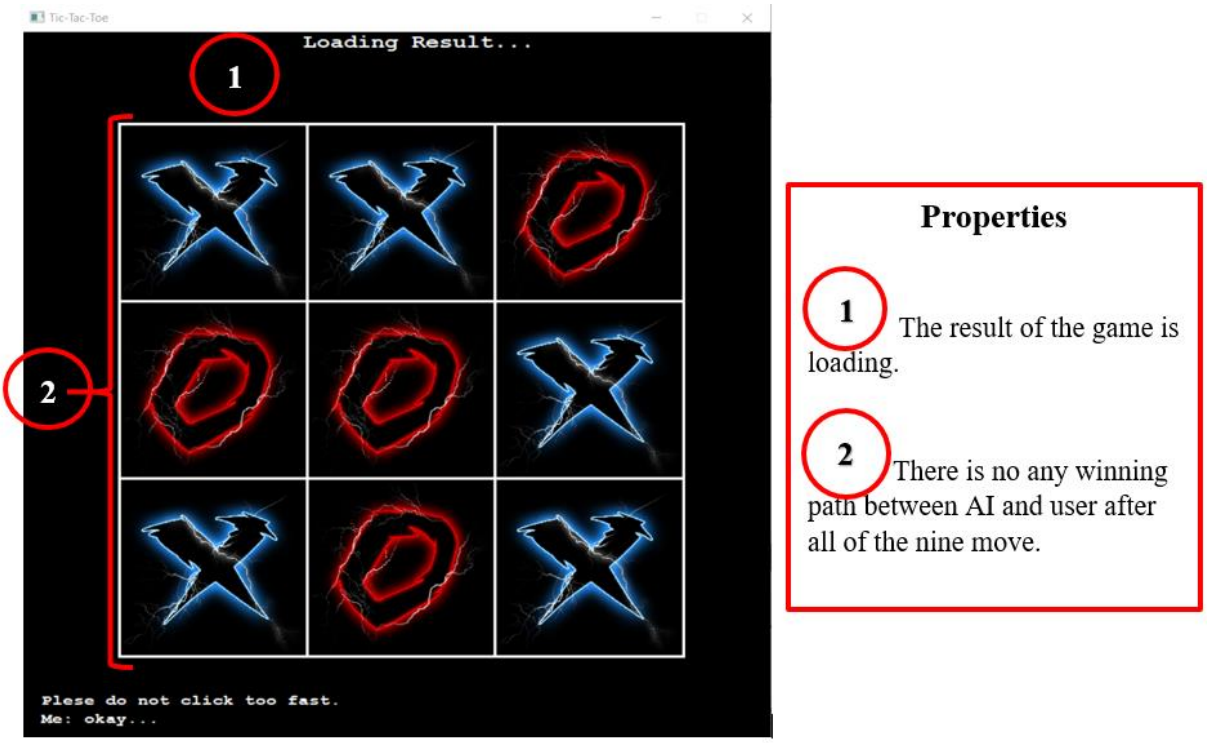



Figure 35: Playing Scene 4

**Draw Result Interface**

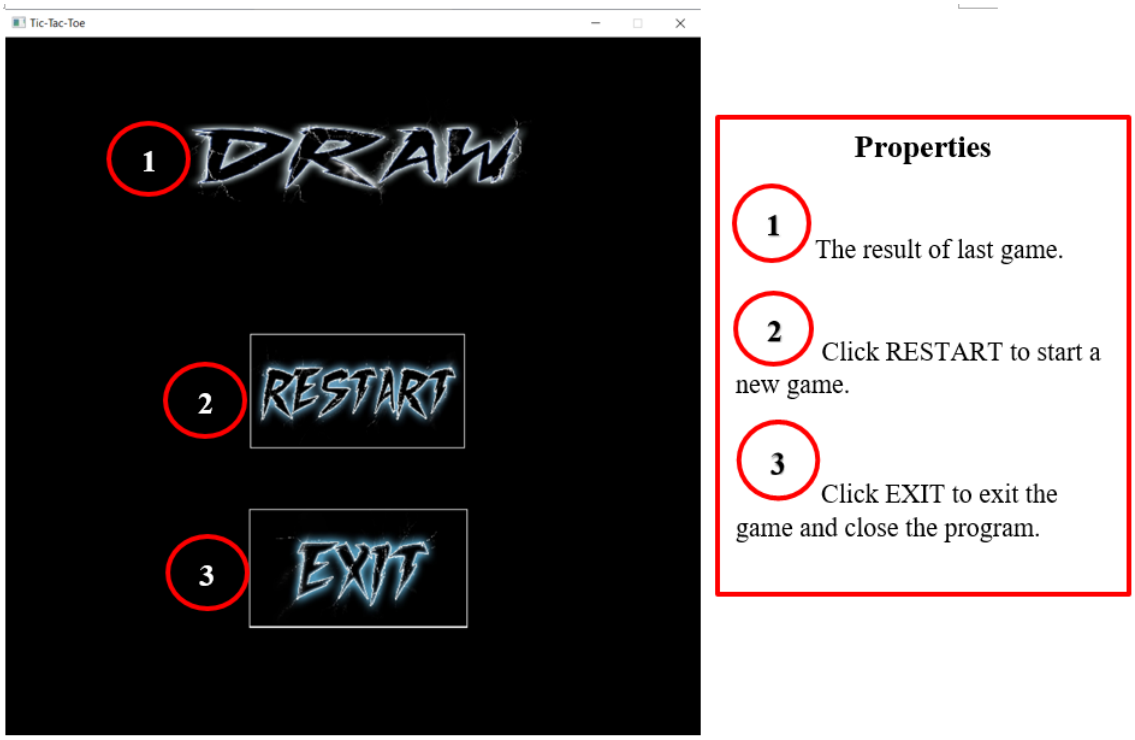



Figure 36: Draw Result Scene

**Condition 2: AI First & User Lose the Game**

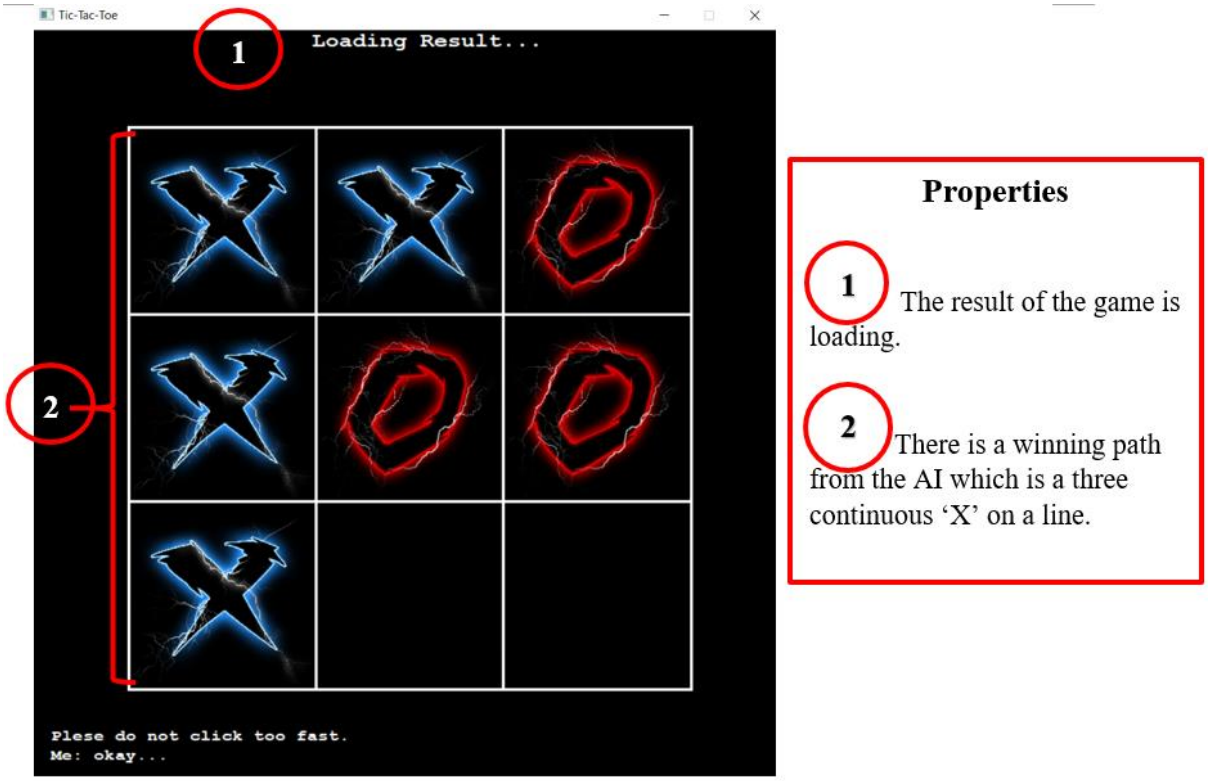



Figure 37: Playing Scene 5

**Lose Result Interface**

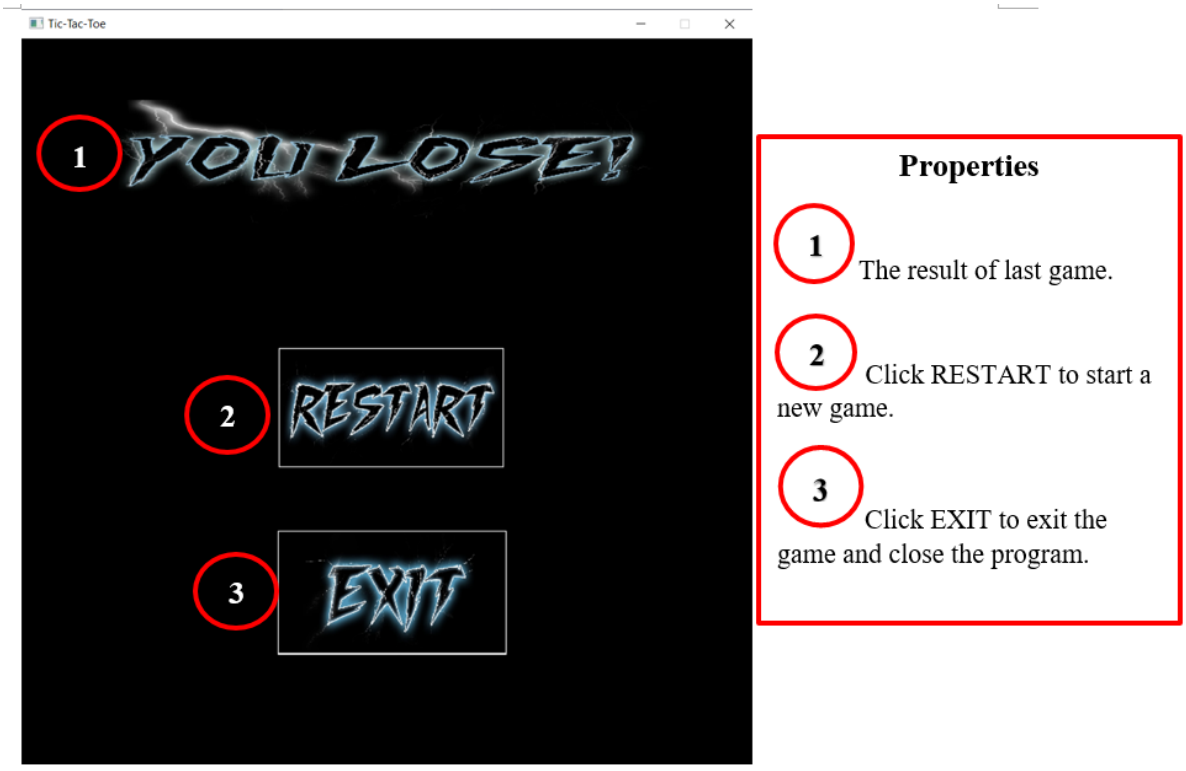



Figure 38: Lose Result Scene

**If ME is clicked**

**User will start the first move**

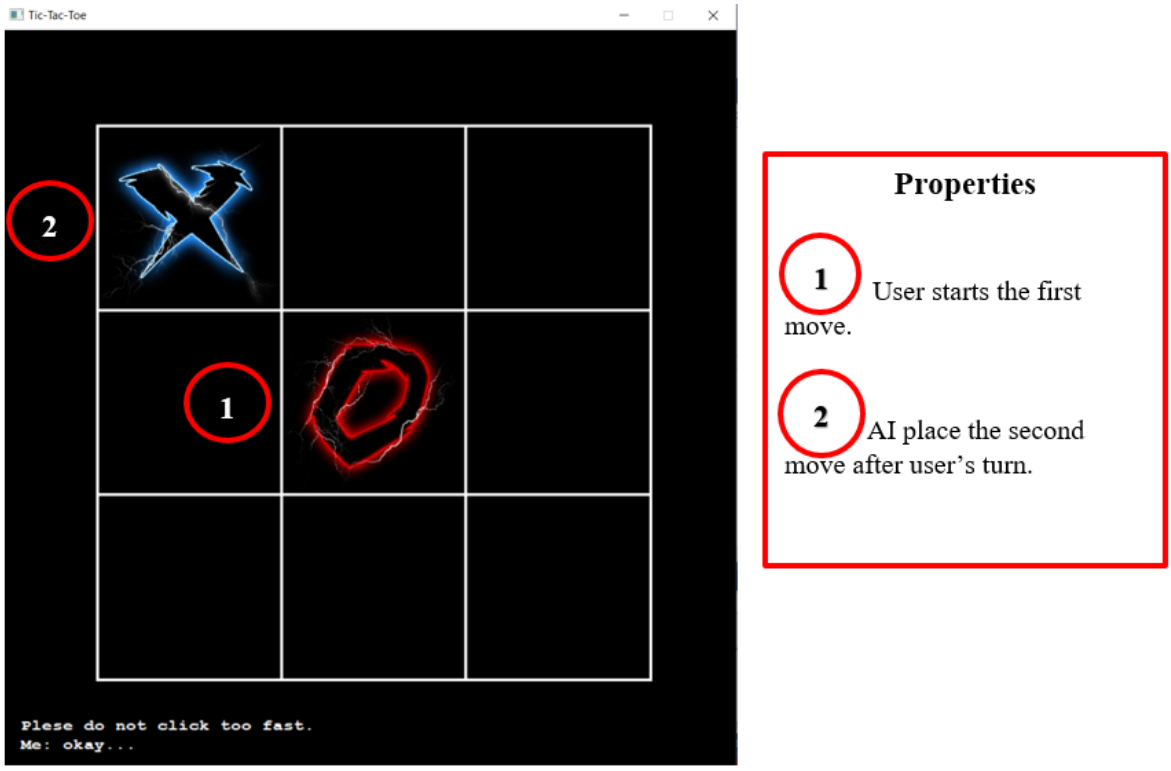



Figure 39: Playing Scene 6

**Continue playing - blocking winning path of each other**

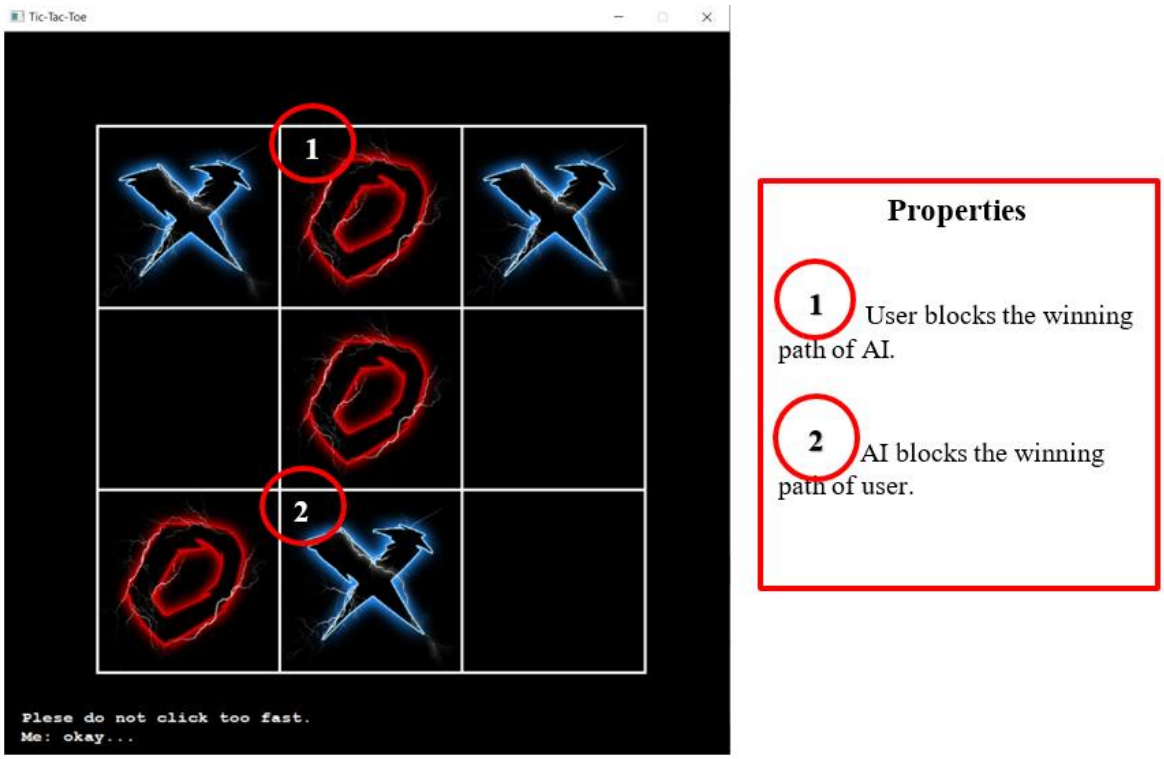



Figure 40: Playing Scene 7

**Loading Result after all the move**

**Condition 1: User First & A Draw Result**

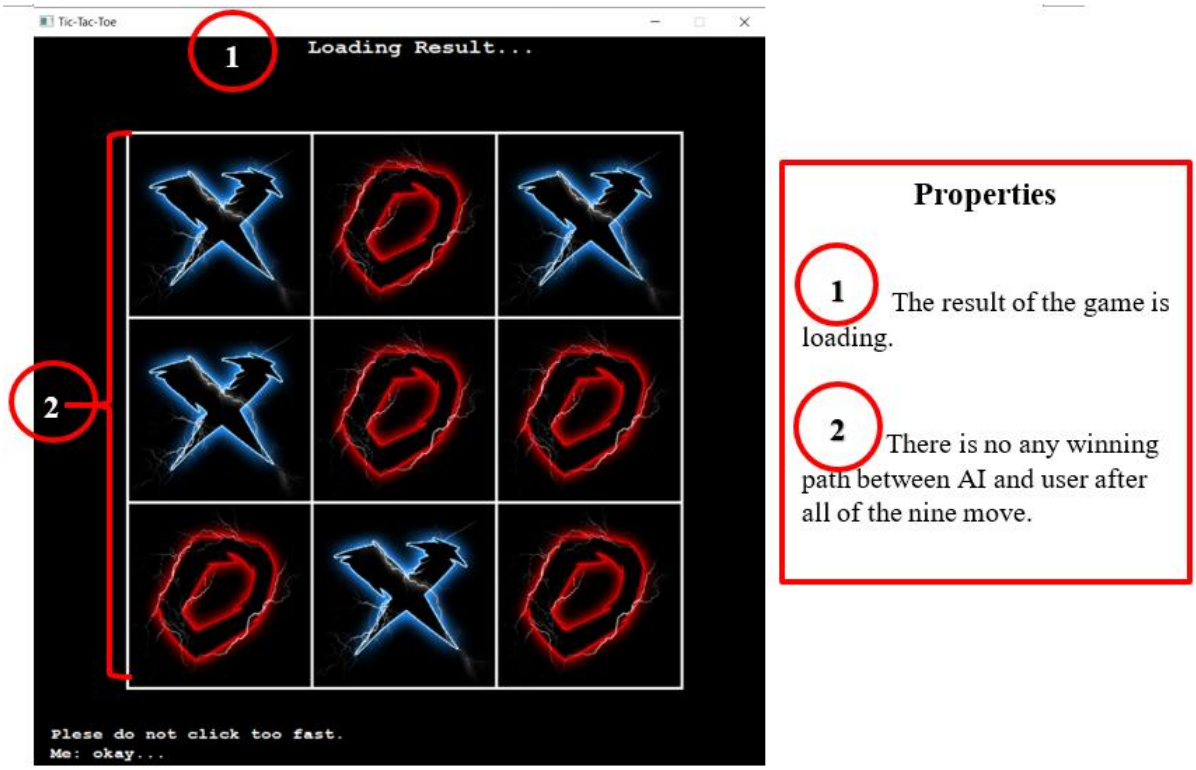



Figure 41: Playing Scene 8

**Draw Result Interface**

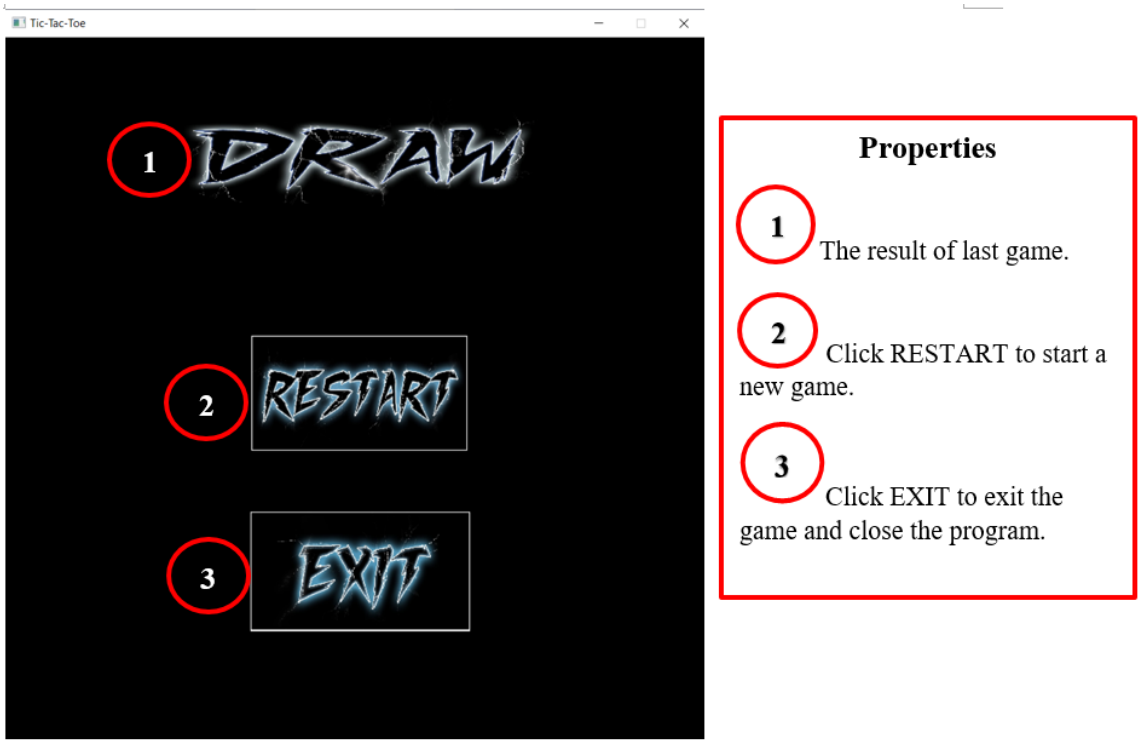
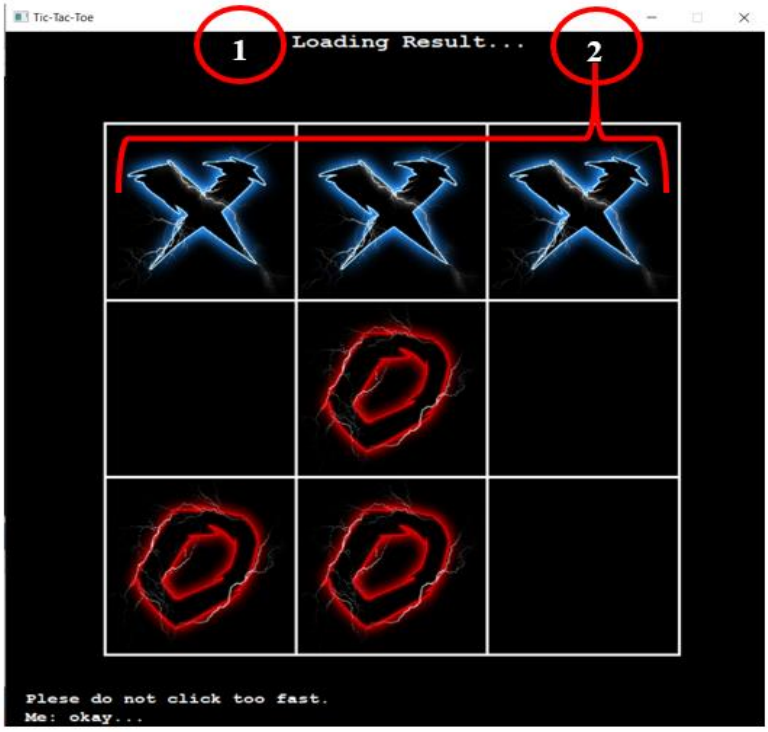



Figure 42: Draw Result Scene 2

**Condition 2: User First & User Lose the Game**



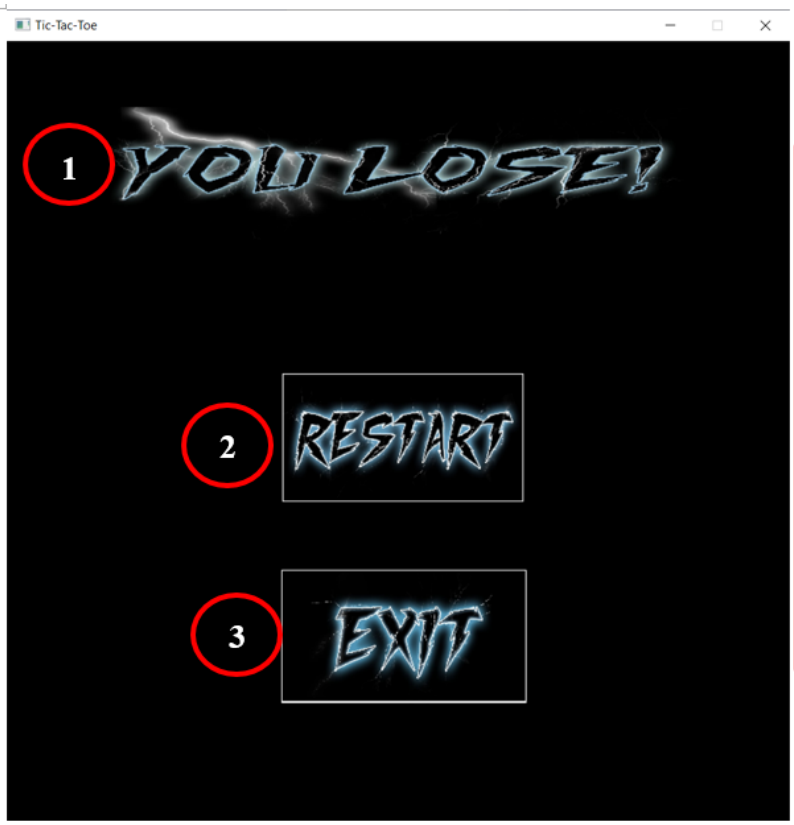

**Properties**

1 The result of the game is loading.

2 There is a winning path from the AI which is a three continuous 'X' on a line.

Figure 43: Playing Scene 9

**Lose Result Interface**




**Properties**

1 The result of last game.

2 Click RESTART to start a new game.

3 Click EXIT to exit the game and close the program.

Figure 44: Lose Result Scene 2

**Win Situation (Almost impossible to happen)**


Figure 44: Win Result Scene

Although we made a win interface, it will never appear as a good AI will not let user to win the game. When there are no players who can win against artificial intelligence, then it is proved that the artificial intelligent heuristic approach (min-max algorithm) applied can really assist it in performing the best move in every different situation.

## 8. Conclusion

In conclusion, by applying artificial intelligence heuristic approach in 2 players board game can really be helpful when you are developing any 2 players board game that consists of a computer which is the AI. After completing this project, we have understood more about the artificial intelligence heuristic approach such as the minimax algorithm and the alpha-beta pruning algorithm. We also learnt how to work together as a group while doing this project as we have to divide our tasks among us. Besides, we also learnt how to communicate with each other while we are having discussions regarding this project. We also learnt how to help each other. When any of us having any trouble while doing this project, all the team members are willing to help by giving suggestions or useful comments.

We also want to take the opportunity to thank our lovely lecturer, Assoc Prof Dr Azlan who had taught us in Artificial Intelligence so that we can have the knowledge to understand AI and apply artificial intelligence heuristic approach in this tic-tac-toe game. We also want to thank him for guiding us when we met some problems and troubles in this project so that we can complete this project successfully and on time. Lastly, we also want to thank our friends and our course mates who were willing to share their knowledge and give us some ideas and guidance to complete this project. Without them, it is impossible for us to complete this project.