LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

REPORT FOR TIC-TAC-TOE GAME

AS a Project work for Course

PYTHON PROGRAMMING (INT 213)

Name	- Adarsh Singh
Registration Number	-12005520
Name	- Akash Singh
Registration Number	- 12008739
Name	- Abhay Shukla
Registration Number	- 12008245
Semester	- Third
University	- Lovely Professional University
Date of submission	- 20th NOV 2021

Table of Contents

TEAM MEMBERS: -

-ADARSH SINGH

-AKASH SINGH

- ABHAY SHUKLA

1. <u>ABSTRACT</u>

The game TIC TAC TOE had an early variant which began in the first century in the Roman Empire. During that time, it was called Terni Lapilli where the players only had three pieces and had to move around the empty spaces to play. The actual game of TIC TAC TOE could be traced back to ancient Egypt. The game was also known during that time as "Three Men's Morris" . The first reference to Noughts and crosses was made in 1864 in a British novel called can You Forgive Her. For many years the game was referred to as noughts and crosses but was changed in the 20th century by the United States to TIC TAC TOE.

The purpose of this documentation is to capture all the requirements by which the user can play a game of tic-tac-toe in Graphical User Interface as well as they can develop a logic that what is actually happening.

2. INTRODUCTION

2.1 OBJECTIVE:

One of the most universally played childhood games is TIC TAC TOE. An Interactive TIC TAC TOE game is developed where two players will be able to play against each other in a suitable GUI by using proper mouse movements. This game will start by showing a simple display, prompt the user for a move and then prints the new board. The board looks like a large hash symbol (#) with nine slots that can each contain an X, an O, or a blank. There are two players, the X player and the o player. By default, players (the O player) takes the initiative. The game will end in two situations: a win, when one player gets three in a row, horizontally, vertically or diagonally. A draw, when there are no remaining places to choose and neither of them has won. Here are some rules for the game:

✓ The player with symbol 'o' goes first

✓ Players alternate placing X s and os on the board until either

✓one player has three in a row, horizontally, vertically or diagonally; or

✓ all nine squares are filled.

✓ The player who can draw three X s or three O's in a row wins.

✓ If all nine squares are filled and none of the players have three in a row, the game is a draw.

3. **SOFTWARE DESCRIPTION**

3.1 PYTHON 3.10.0 IDLE

IDLE stands for Integrated Development and Learning Environment is an integrated development environment for python. It has been bundled with the default implementation of the language since 1.5.2b 1 It is packaged as an optional part of the python packaging with many Linux distribution. It is completely written in python3 and Tkinter GUI toolkit IDLE IS intended to be a simple IDE and suitable for beginners as well as advanced users. To that end it is cross platform and avoids feature clutter. The features provided by the IDLE includes

✓Python shell with syntax highlighting.

Integrated debugger with stepping, persistent breakpoints and call stack visibility

Multi-window text editor with syntax highlighting, auto completion, smart indenting etc

3.2 DEVELOPMENT TOOLS AND TECHNOLOGIES

3.2.1 PYTHON 3:

Python is a general purpose interpreted, interactive, object-oriented, and high-level programming language it was created by Guido Van Rossum during 1985-1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). Python is named after a TV show called "Monty Python's Flying Circus

Python 3.0(also called "Python 3000" or "Py3K) was released in December 3, 2008. The latest version of python accumulated new and redundant ways to program the same task, Python 3.6 had an emphasis on removing duplicative constructs and modules, in keeping with There should be one and preferable only one - obvious way to do it Python's dynamic typing encourage the programmer to write a code that is clear, well structured as well as easy to understand. The features of dynamic typing are

Values are inspect-able.

✓ Types are bound to values but not to variables.

✓ Values are inspect-able.

✓ Function and method lookup is done at runtime

✓ There is an interactive interpreter more than one, in fact.

✓ You can list the methods supported by any given object

Because code is automatically compiled to byte code and executed. Python is suitable use as a scripting language Web application implementation language. etc. Because of its strong structuring constructs (nested code blocks, functions, classes modules and packages) and its consistent use of objects and OOP, Python enables you to write clear and logical code for small and large projects.

3.2.2 GUI (GRAPHICAL USER INTERFACE):

Our goal in this article is to provide you with an introductory of GUI programming. In order to learn the GUI programming, you must first understand a few core aspects of GUI So let's begin the discussion The primary GUI toolkit will be using Tk, Python's default GUI. We'll access Tk from its python interface called Tkinter. It is not the latest and greatest nor does it have the most robust set of GUI building blocks, but it is fairly simple to use and using it, you can build GUI's that run on most platforms Setting up GUI application is similar to how an artist produces a painting, Conventio nally there is a single canvas onto which the artist must put all the work. In GUI programming a toplevel root windowing object contains all of the little windowing objects that will be a part of your GUI application. These can be text labels, buttons, list boxes etc These individual little GUI components are known as widgets Top level windows are those that show up stand alone as part of your application. Interestingly, you can have more than one top level window for your GUI, but only one of them should be your root window.

The top level window can be created using this

```
import tkinter top-tkinter.Tk ()
```

The object returned by tkinter.Tk() is usually referred to as the root window.Within this window you can place multiple component pieces together to form your GUL Tk has three geometry managers that help with positioning your widget set

✓Packer: It packs widgets into the correct places

✓Grid: It is used to specify GUI widget placement based on grid coordinates

✓Placer: You provide the size of the widgets and locations to place them, this manager then places them for you

Now once the packer has determined the sizes and alignments of your widgets, it will then place them on the screen for you. When all the widgets are in place we instruct the application to infinite main loop in Tkinter the code that does it is

```
Tkinter.mainloop())
```

This is normally the last piece of sequential code your program runs

4. MODULE DESCRIPTION

4.1 INTIALIZATION:

In the main class we've created a constructor which will hold all the tkinter widgets that we are using in our program First of all, we've used a highly versatile widget called canvas to draw graphs and plots on our root window. We set the background colour as white. Now we want to realize the effects that we are made in our root window For that we have to use the pack geometry manager. Then we've called the bind function, which is actually used to interact the GUI widget with our computer program in the bind function we've passed two objects namely <x>, self.exit. The later one is used to exit from the root window. in the root window there should be three options displaying at the top level. The minimize button(); the maximize button () and the close button (X) Now whenever the user clicks on the X button the bind function should capture that and accordingly the root window will destroy that means it could not be displayed any more. Then we set our game state to zero (0) and call the title screen module. After that we create a list with nine strings and initiate each of them with a value zero (0) and store it in a variable called board.

4.2 TITLE SCREEN:

In this module, what we've done is we use the delete method to clear the canvas We use the special tag "all to delete all the items on the canvas. Then we create three rectangles on our root window. We create these rectangles by using canvas supported standard item call create rectangle For drawing the first rectangle we've passed the rectangle bounding size which is nothing but the original window size and the rectangle options which includes fill and outline as an object We choose the white colour as the rectangle boundary colour Applying the same procedure, we have created the second rectangle. The only additional thing that is done here is regardless of sending the original window size we'll send the adjusted window size as an object That means we'll just modity the X and Y axis is such a way that the beauty of the window is preserved Now, for the third rectangle we've applied the same procedure and we select appropriate colour for displaying all the texts that will be appeared in this rectangle Now we want to display the title of the game and besides that. because we also would like to increase the usability of the software that we are developing so we also have to create another text that will increase the usability of the software. For that we are using canvas supported standard item called create_text that will enable us to create informative text on the screen Intuitively, the title of the game should be TIC TAC TOE and just below to that another text that will be appearing is Click to play)

4.3 GAMEOVER SCREEN:

This module is concerned with displaying the result according to the outcome on the screen. It takes outcome as an object and displays the appropriate result if the outcome is in favour of player 1, O Wins appears on the screen in the form of a text. Similarly, if the outcome is in favour of player2, X Wins should appear on the screen and if the outcome is a draw. then both the players will see the text "Draw on the screen We add one extra feature in the game over module by introducing a text called Click to play again and we're very optimistic that it should increase the usability of the developed software, For displaying the texts, we've used the canvas supported standard item call create_text function in the game over screen firstly we create a rectangle on the canvas and fills it with appropriate colour and inside that rectangle all the texts will appear. We've created the texts by adjusting the text size, font style text height, text width and text colour

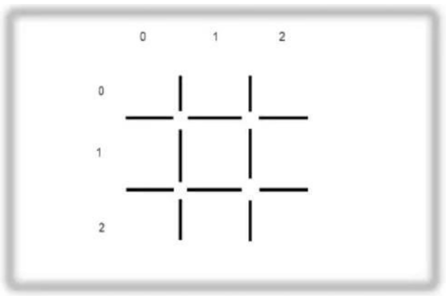
4.4 CLICK:

This module handles most of the game logic A tkinter application runs most of its time inside an event loop, which entered via the main loop method. It waits for an event to happen. Events are generally key pressed or mouse operations by the users We've already set the game states globally by assigning the values STATE_TITLE_SCREEN to 0, STATE X _TURN to 1, STATE_O_TURN to 2 and STATE_GAME OVER to 3. In the click method we've passed the first argument implicitly (called self) which is the reference to our main class itself and it should have the event object that merely describes what is actually. happened. We begin our task by converting the pixels into gird coordinates and then we have to check for the current game state condition and accordingly the game will continue Now if game state is 0, then it will draw a new board on the screen and sets the game states as the value of the first player. If the game state is 1 e X's tum and the cell is empty then the control goes to new move module (where we've to mandatorily pass the current player and the board coordinates as arguments) and the appropriate cell is filled up with symbol X. Now it checks all the possible conditions (ie whether player X has won or it is a araw) one by one. So, at first the control moves to has won module and if the result is true then it sets the game state to 3 and calls the game over module by passing the parameter X wins. If it is false then the control goes to is a draw module and if the result is true then again it will set the game state to 3 and calls the game over module with the parameter Draw If the result of both the cases is false then only it will set the game state as 2 which is nothing but player O can now take its turn. Now If the game state is 2 ie. O's tum and the cell is empty then the control goes to new move module and the appropriate cell is filled up.

with symbol O. Now, it checks all the possible conditions (ie whether player O has won or it is a draw) one by one. So, at first the control moves to has won module and if the result is true then it sets the game state to 3 and calls the game over module by passing the parameter O wirts If it is false then the control goes to is a draw module and if the result is true then again it will set the game state to 3 and calls the game over module with the parameter Draw. If the result of both the cases is false then only it will set the game state as 1 which is nothing but player 1 can now again take its tum Finally, it checks for if the game state is being at 3 or not. If it is 3, then it will reset the board by calling the module new board and sets the game state according to the first player With this we're completing most of the work that is behind our game logic.

4.5 NEW MOVE :

This module is concerned with pacing the X's and O's in the selected cells on the board, it receives the current player and the grid coordinates from the click module as an argument and places the symbol of the player in the desired cell. Now let us consider our board structure



Here X and Y are 0 based grid coordinates. If the current player is X then it will call the draw X module (discussed later) with the grid coordinates and places the X symbol with appropriate size and colour into that grid. Now if the current is 0 then it will call the draw O module with the 0 based grid coordinates and places the O symbol into that cell.

4.6 DRAW X:

This module deals with drawing X on the canvas So, for that what we've to do is, first of all we've to convert the grid into pixels and then we can start our drawing using canvas supported standard packages namely create line. For that reason, we've called the grid to pixel module twice and we get the converted pixels. Now we must have to adjust the width and height of the X so that no extra outline would appear in the right-hand side and left-hand side. We divide he cell size by 2 and multiplied it with the symbol size and store it in a variable called delta By adjusting the delta, choosing the colour and selecting the width in the create_line function we've completed our task

4.7 DRAW O:

This module is concerned with drawing an O on the canvas. Like the previous module, here also we've to convert the grid coordinates into pixels and then we've to adjust the O's in the board. Again, we divided the cell size by 2 and multiplied it with the symbol size and assigned it in a vanable called delta For drawing an O we've to use the canvas supported standard package called create_oval function in create oval function, pixel coordinates are adjusted in such a way that the O's have no odd lines on the left-hand side as well as night hand side Then we select the appropriate width and the colour for O's so that it looks better

4.8 HAS WON:

This module is concerned with checking for the winner and returns the result to an appropriate calling function in the click module we've already converted the grids into pixels and store them into some variables. Now the has won function takes those pixels as an argument and accordingly the board will fill up with the winning symbols. The checking for the winner is quite simple If any of the players can able to fill all the cells in a row vertically or horizontally or diagonally, he /she will consider to be the clear winner So, in the first for loop, we basically check that whether all the cells of a row (vertically or horizontally or diagonally) are filled with the same symbol or not. Now if this condition satisfies, then the player with that symbol is the winner In the second for loop we need to check those positions that are conventionally reserved for the winning position. So, we need to check the positions ([0][0][0][1][02]) or ([10][111][12]) or ([210] [211] [212]) or ((00][1][0][210]) or ([011][11][21]) or ([012] ['12][22]) or ([0][0][111][22]) or ([0][2][111] [210]) and by checking these positions we will certainly get the winner

4.9 IS A DRAW:

In this module, we're just checking the result is a be or not. The result is a tie when all the boxes/cells are filled. Checking whether it is a draw is fairly straightforward. All we ve to do is check whether all the board boxes are filled. We can do this by iterating through the board arra

4.10 GRID PIXEL:

The main objective of this module is to transform the grid coordinate into pixels. Intuitively we've passed the grid coordinate as an argument and this module returns accurate pixel coordinate of the centre. The pixel coordinate should be calculated with the help of grid coordinate and the cell size. Thus, we add the original cell size with the half of it and multiplied it with the gird coordinate and stored the result in a variable called pixel_coord. By doing this, the grid to pixel transformation is mostly done.

4.11 PIXELS TO GRID:

It is just the reverse of the previous module. This module performs exactly the opposite task, taking a pixel coordinate as an argument and converting this into its equivalent grid coordinate. Somehow the canvas has a few extra pixels on the right and bottom side so we've to fix this. For clipping the extra pixels, we must check the condition that if the pixel coordinate is somehow greater than the original window size then we've to reduce the window size. The transformation can be achieved by dividing the pixel coordinates by the cell size. The resultant grid can now be stored into a variable say grid_coord.

EXIT

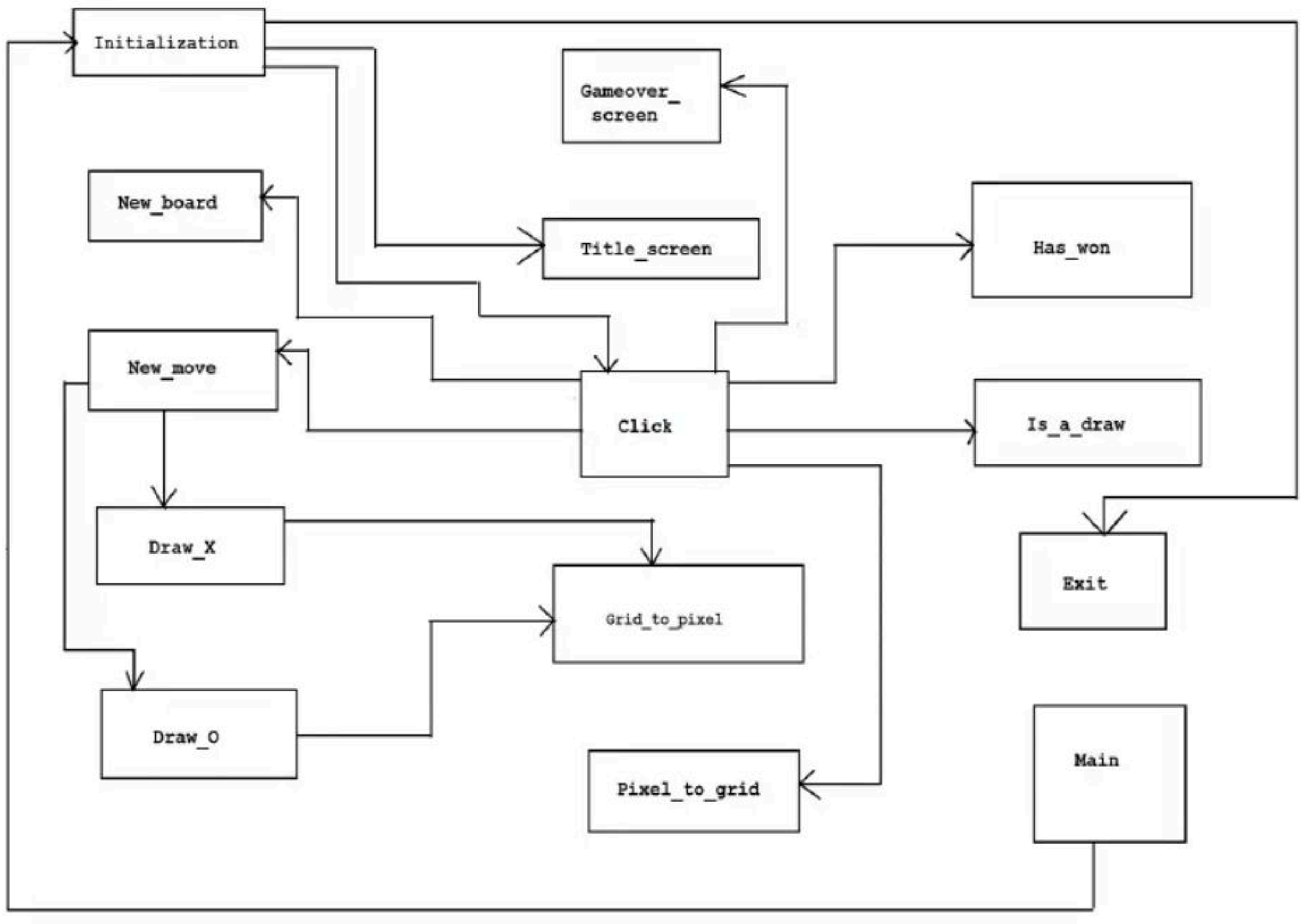
This module is used for closing the root window permanently. In this module we've passed event as an argument based upon this event the window can be closed. We want the access from the current instance in our exit () method which is done through self.

```
self.destroy()
```

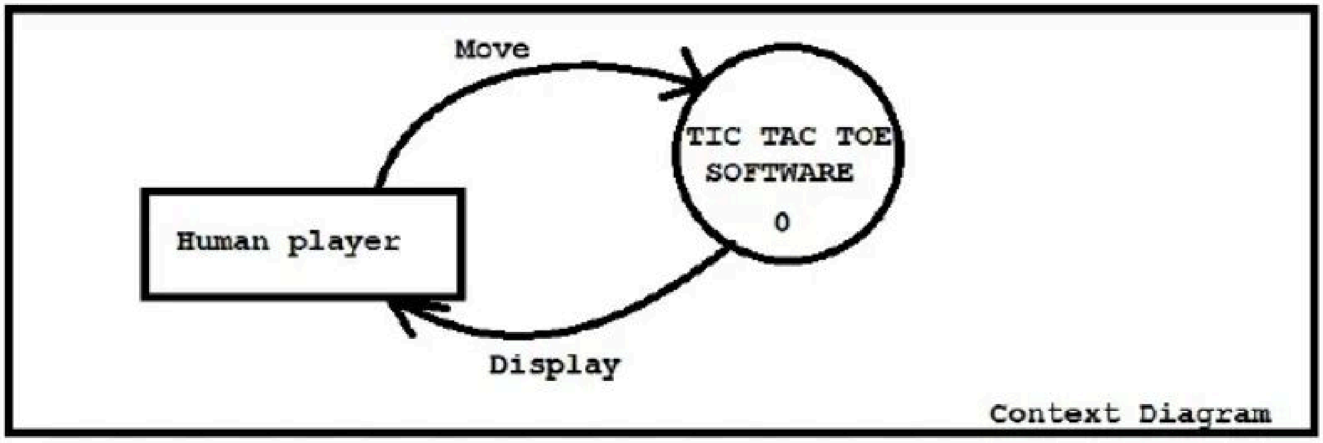
However, this only destroys the frame and its child windows, like the canvas,

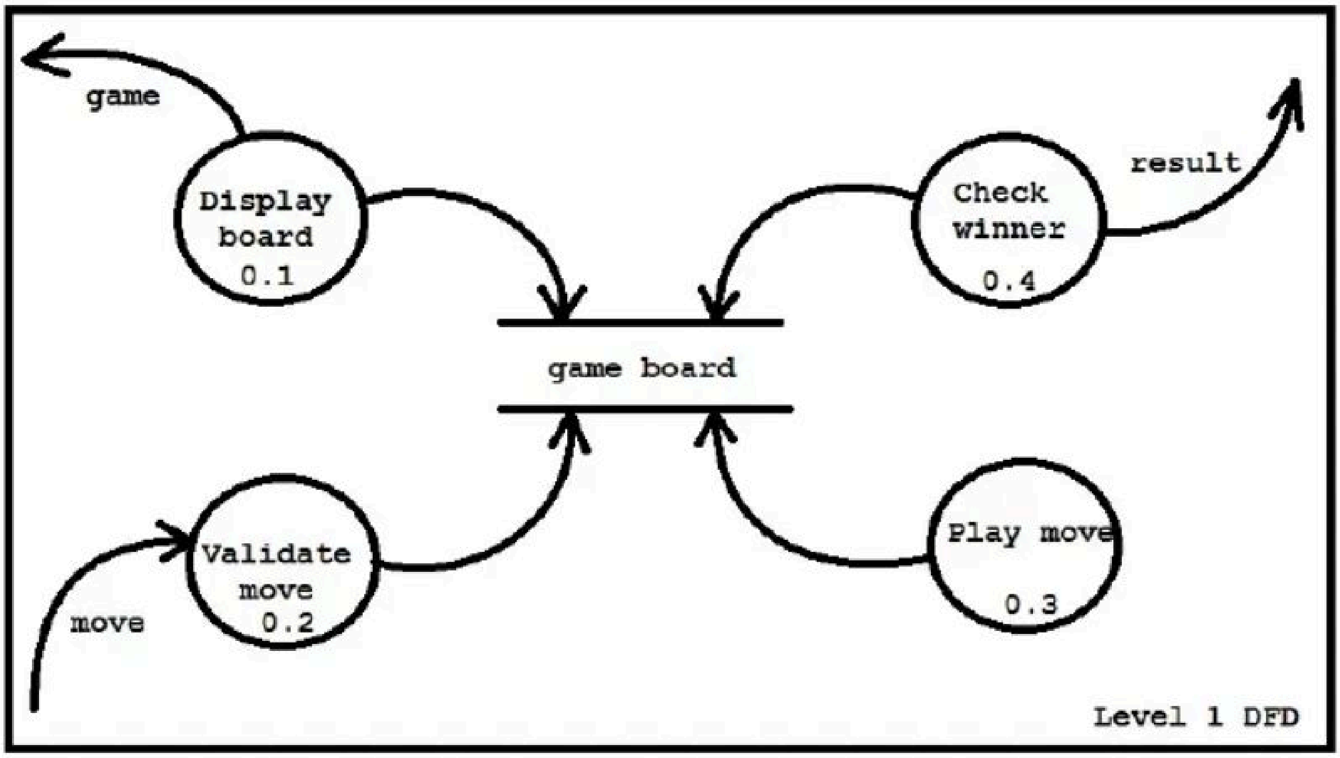
not the top-level window

SYSTEM FLOW DIAGRAM

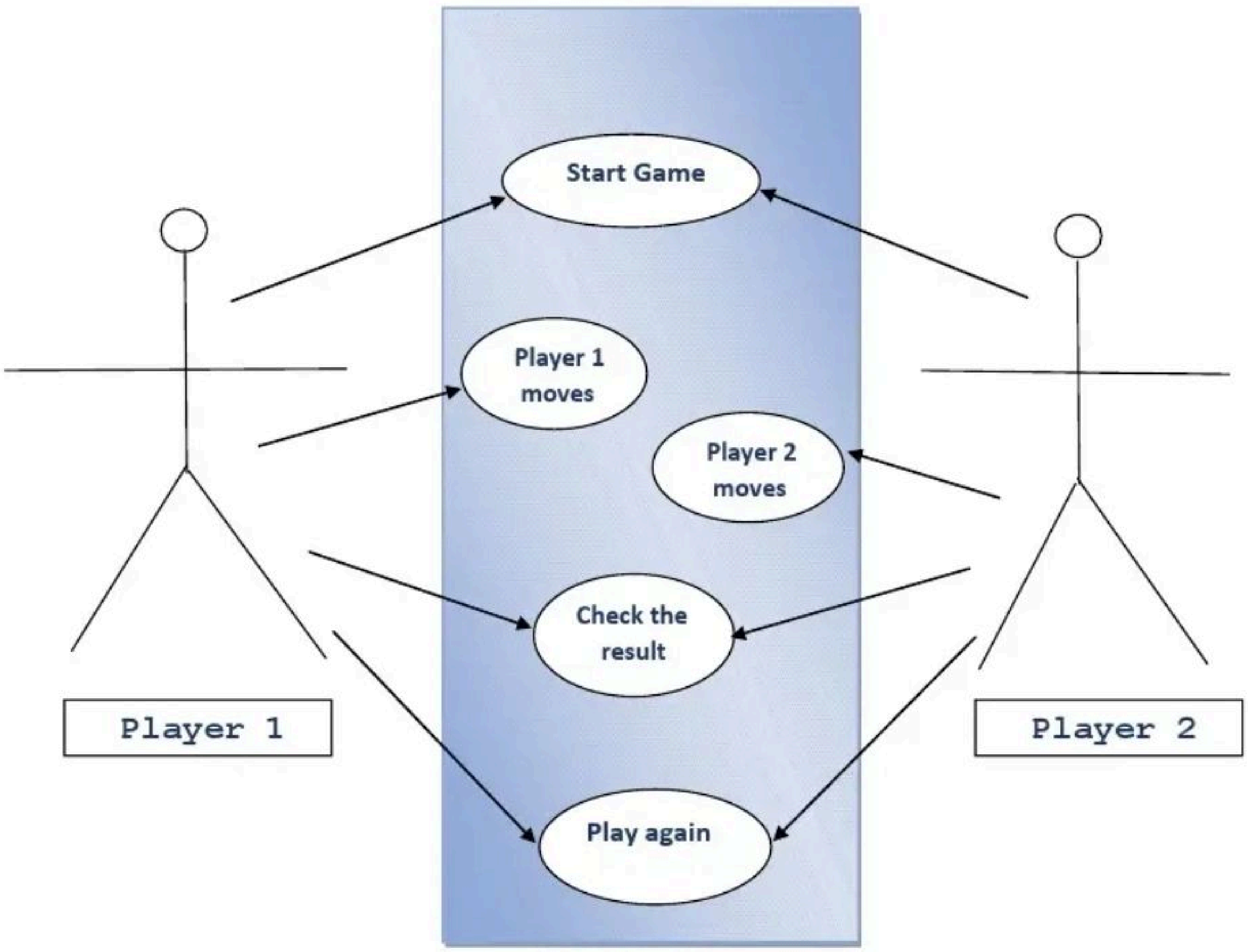


DATA FLOW DIAGRAM

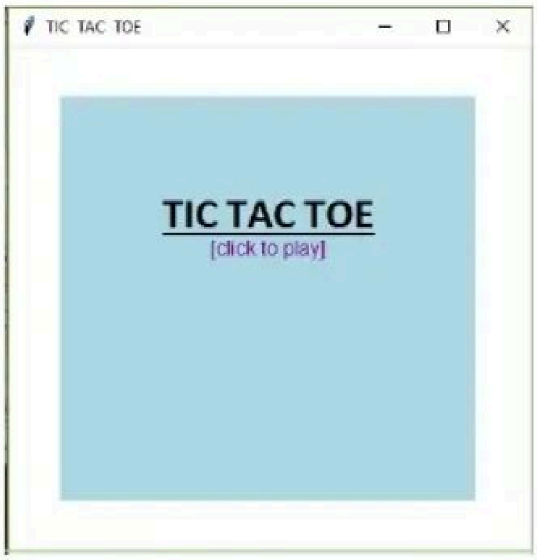


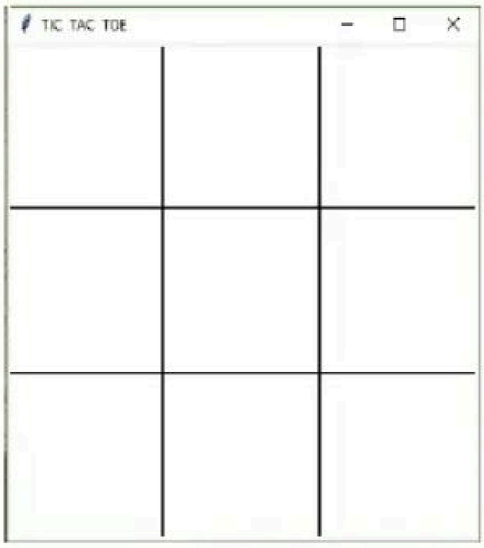


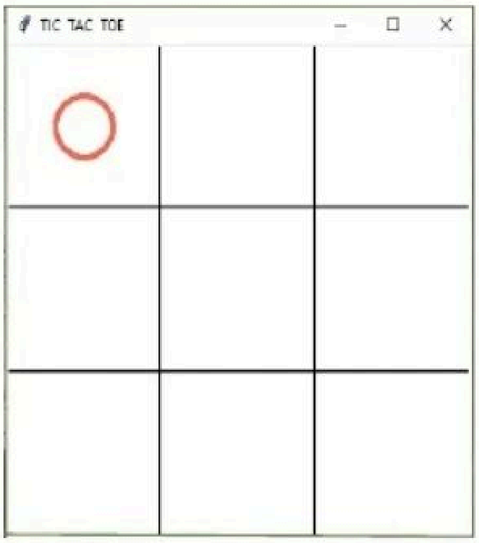
USE CASE DIAGRAM

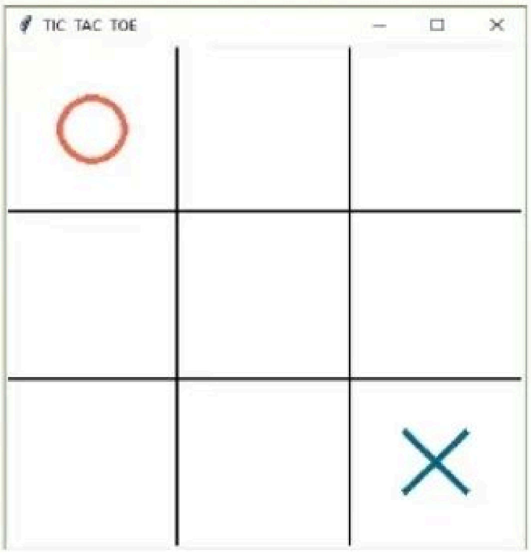


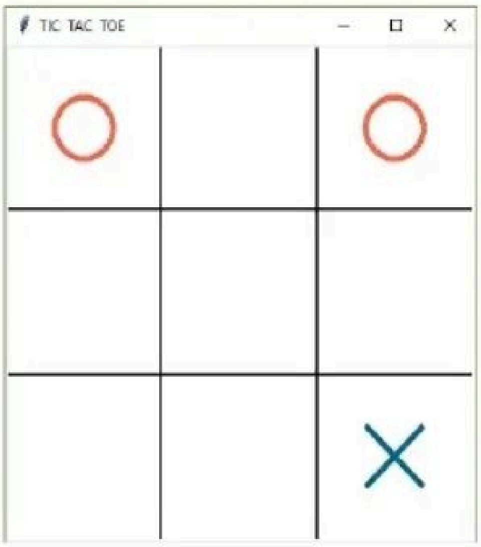
SCREEN SHOTS:

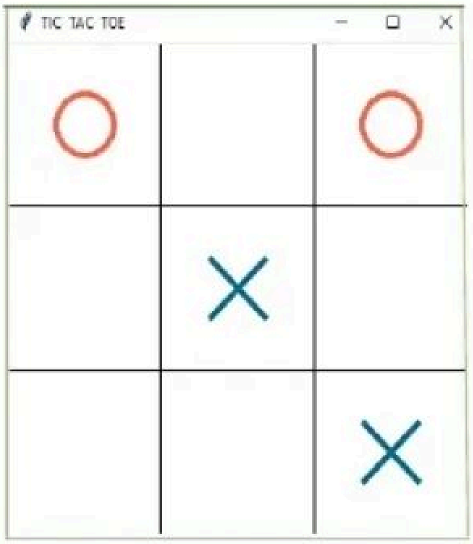


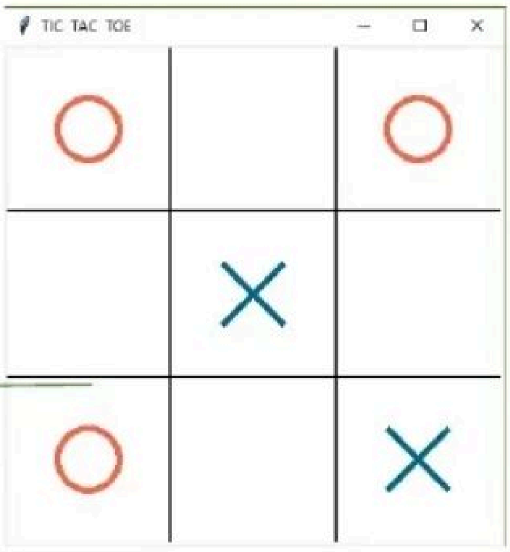


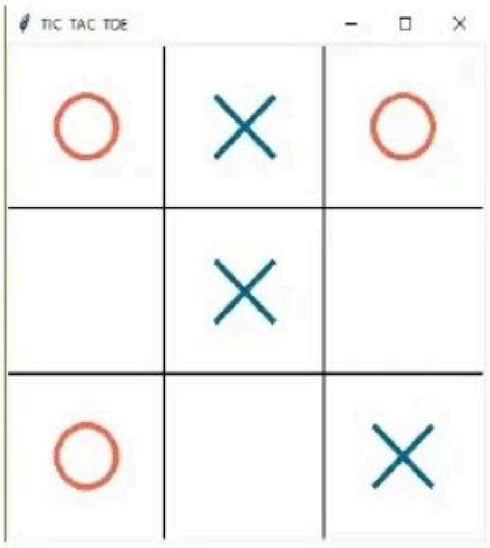


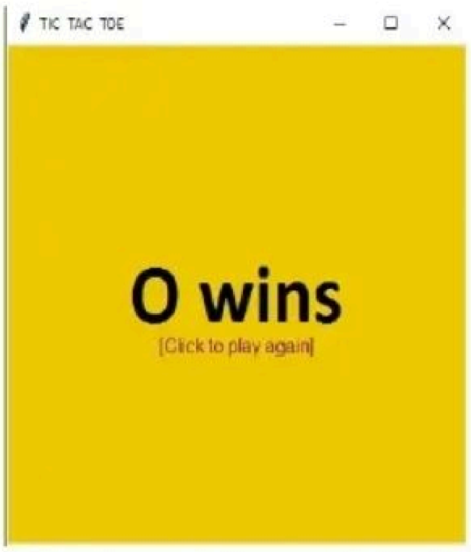


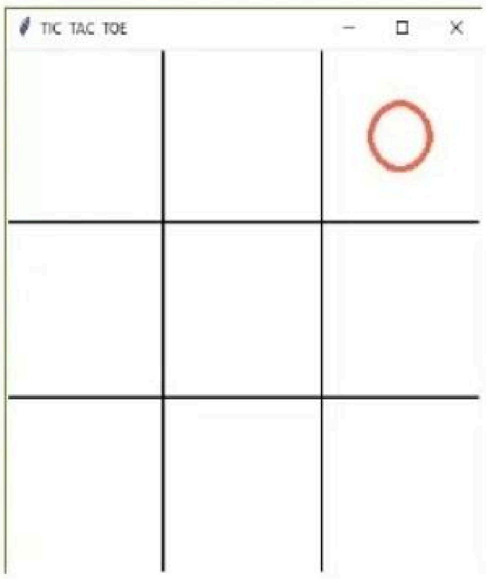


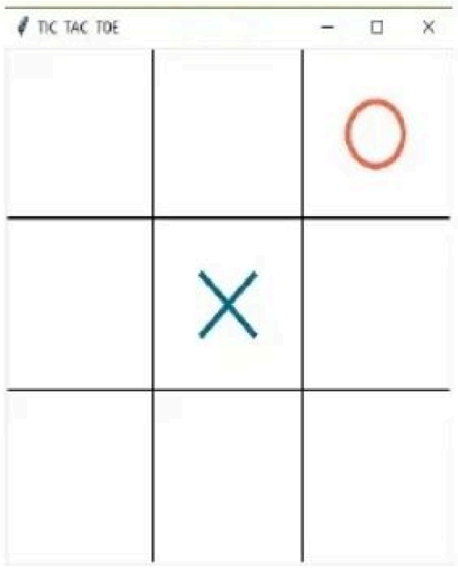


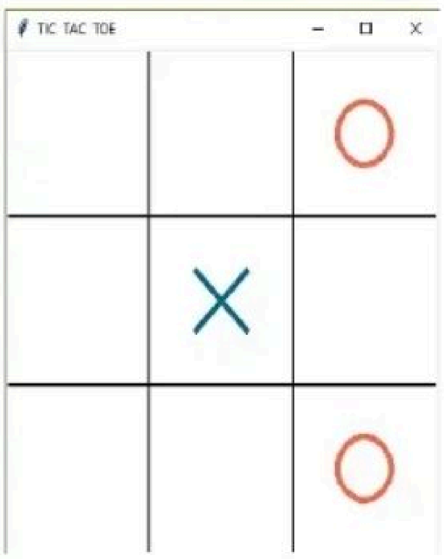


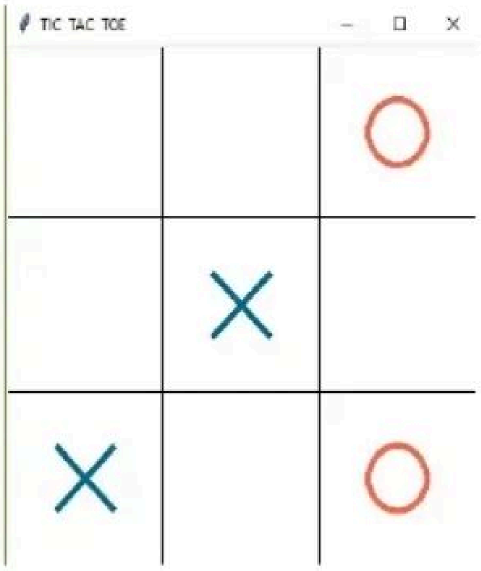


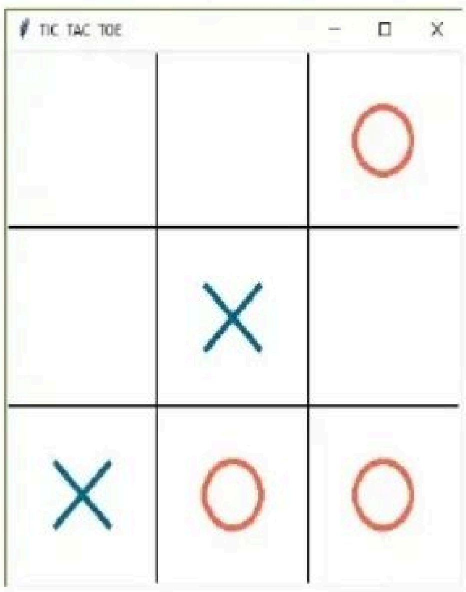


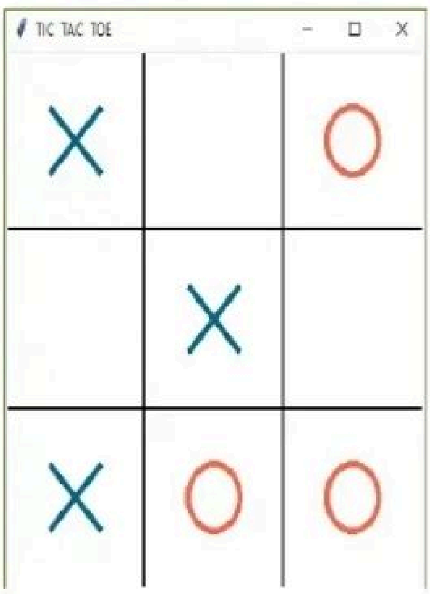


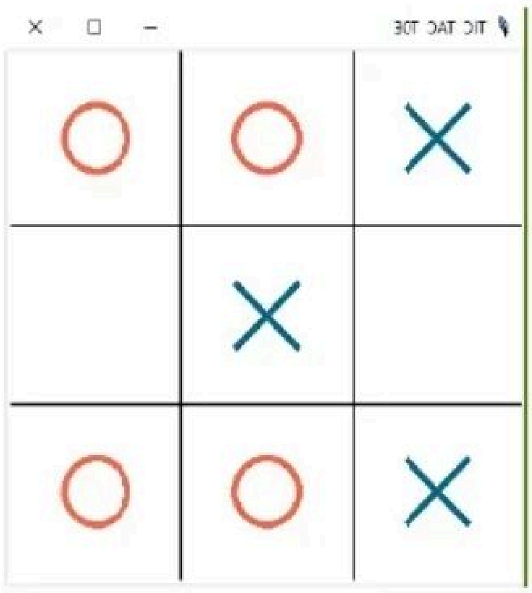


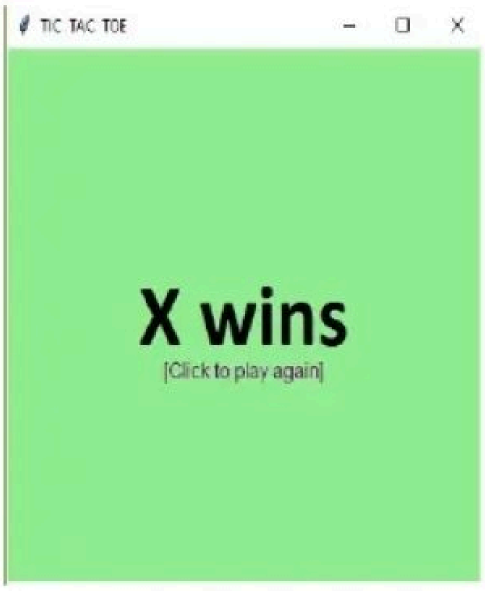






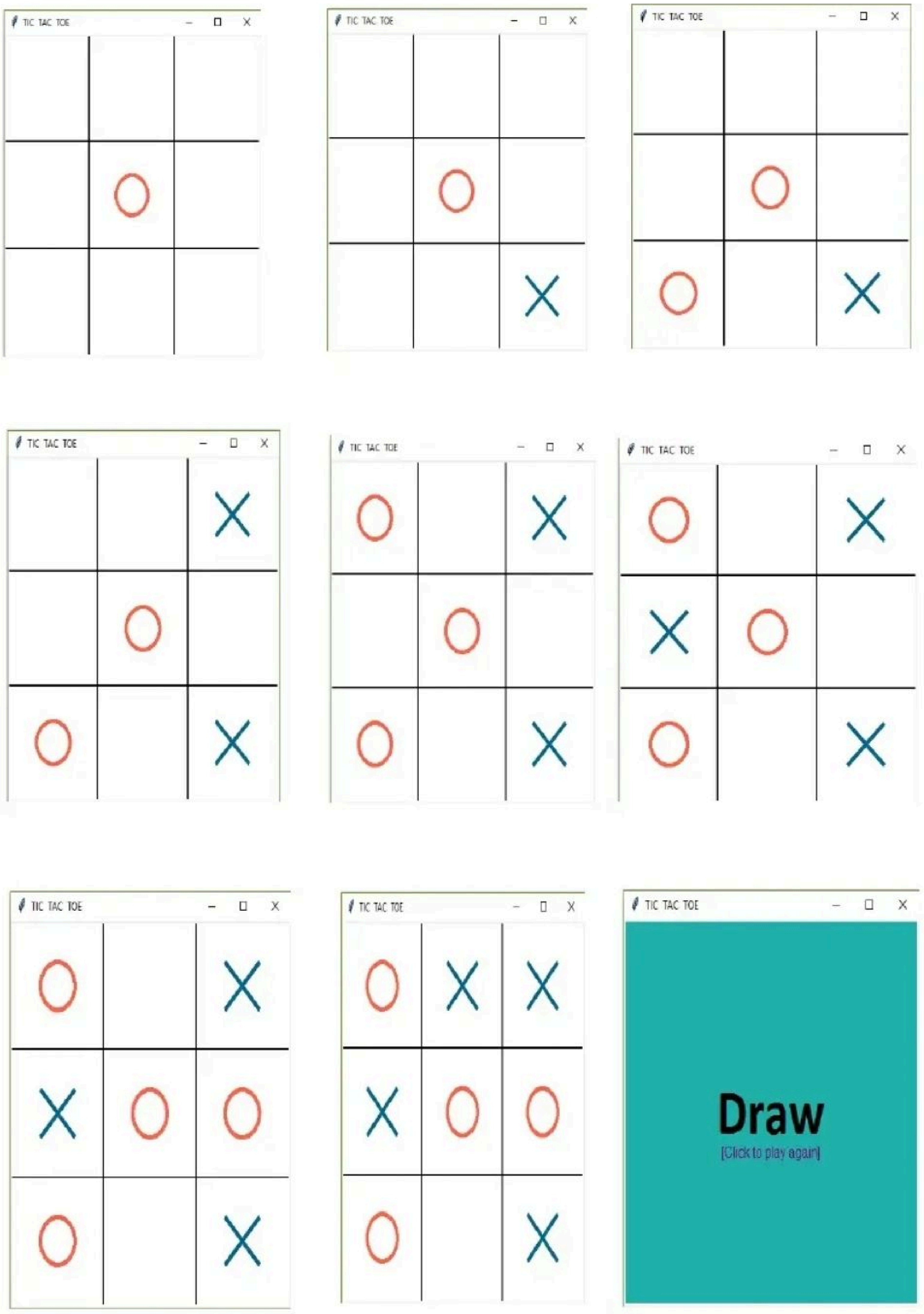






X wins

[Click to play again]


TIC TAC TOE
Draw
[Click to play again]