# CHAPTER 1

# INTRODUCTION

## 1.1 COMPUTER GRAPHICS

Computer Graphics are the pictures created using computers. It is the creation and manipulation of picture with the aid of computers. It is divided into two broad classes. It is also called passive graphics. Here the user has no control over the pictures produced on the screen. Interactive graphics provides extensive user-computer interaction. It provides a two-way communication between computer and user. It provides a tool called "motion dynamics" using which the user can move objects. It is also able to produce audio feedback to make the simulated environment more realistic. C language helps to implement different graphics objects which are interactive and non-interactive.

Computer Graphics is one of the most powerful and interesting facets of computers. There is a lot we can do in graphics apart from drawing figures of various shapes. All video games, animation, multimedia predominantly works using computer graphics. There are so many applications of computer graphics, which make it significant. Computer graphics is concerned with all aspects of producing pictures or images using a computer.

## 1.2 OpenGL Technology

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment. OpenGL Available Everywhere: Supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and MacOS PC, no other graphics API operates on a wider range of hardware platforms and software environments. OpenGL runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, OPENStep, and BeOS; it also works with every major windowing system, including Win32, MacOS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies. The OpenGL interface :Our application will be designed to access OpenGL directly through functions in three libraries namely: gl,glu,glut.

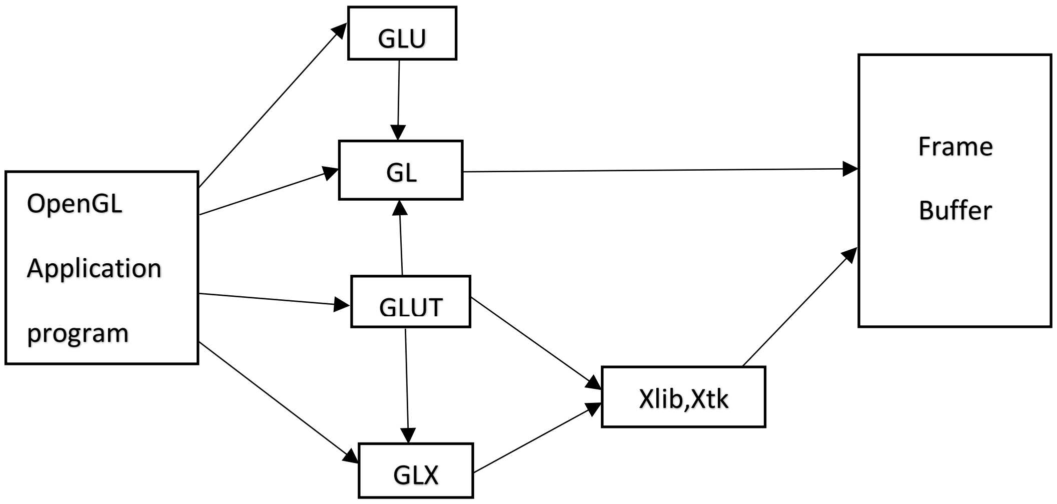


Fig 1.1 :OpenGL interface structure

OpenGL serves two main purposes:

• To hide the complexities of interfacing with different 3D accelerators, by presenting the programmer with a single, uniform API.

• To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set (using software emulation if necessary).

## 1.3 Overview of project

Tic toe game OpenGL project is created using computer graphics. X's and O's are arranged horizontally, vertically or diagonal to win the game. The player chooses either X or O to play. The computer automatically plays as the remaining symbol. The player plays with the computer as the opponent.

## 1.4 Problem statement

OpenGL software consists of distinct commands which can be used to specify the objects and operations needed to produce interactive three-dimensional applications. This is a project developed to demonstrate different commands and primitives of OpenGL.

# CHAPTER 2

# REQUIREMENTS AND SPECIFICATION

## 2.1 Hardware Requirements

The standard output device is assumed to be a Color Monitor. It is quite essential for any graphics package to have this, as provision of color options to the user is a must. The mouse, the main input device, has to be functional i.e. used to give input in the game. A keyboard is used for controlling and inputting data in the form of characters, numbers i.e. to change the user views . Apart from these hardware requirements there should be sufficient hard disk space and primary memory available for proper working of the package to execute the program. Pentium III or higher processor, 16MB or more RAM. A functional display card.

Minimum Requirements expected are cursor movement, creating objects like lines, squares, rectangles, polygons, etc. Transformations on objects/selected area should be possible. Filling of area with the specified color should be possible.

## 2.2 Software Requirements

The editor has been implemented on the OpenGL platform and mainly requires an appropriate version of eclipse compiler to be installed and functional in ubuntu. Though it is implemented in OpenGL, it is very much performed and independent with the restriction, that there is support for the execution of C and C++ files. Text Modes is recommended.

Developed Platform

<u>Ubantu 16.1.04</u>

Language Used In Coding

<u>C-language</u>

## 2.3 Functional requirements

A functional requirement defines a function of a system and its components. A function is described as a set of inputs, the behavior, and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. The functional requirement used in the project is:

• Keyboard function

## 2.4 Non-functional requirements

A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. Non-functional requirements define how a system is supposed to be. It essentially specifies how the system should behave and that it is a constraint upon the systems behavior. Reliability describes the ability of a system or component to function under stated conditions for a specified period of time. Reliability is defined as the probability of success as the frequency of failures. Maintainability is the ease with which a product can be maintained in order to isolate defects, correct defects or their cause, maximize a product's useful life, maximize efficiency, reliability, and safety. Availability is the probability that a system, at a point in time, will be operational and able to deliver the requested services. It is typically measured as a factor of its reliability - as reliability increases, so does availability.

# CHAPTER 3

# DESIGN PHASE

## 3.1 DESIGN OF THE GAME

The full designing and creating of Tic-Tac-Toe has been executed under ubantu operating system. This platform provides and satisfies the basic need of a good compiler. Using GL/glut.h library and built in functions make it easy to design good graphics package such as this simple game.

The following example game is won by the first player, X:

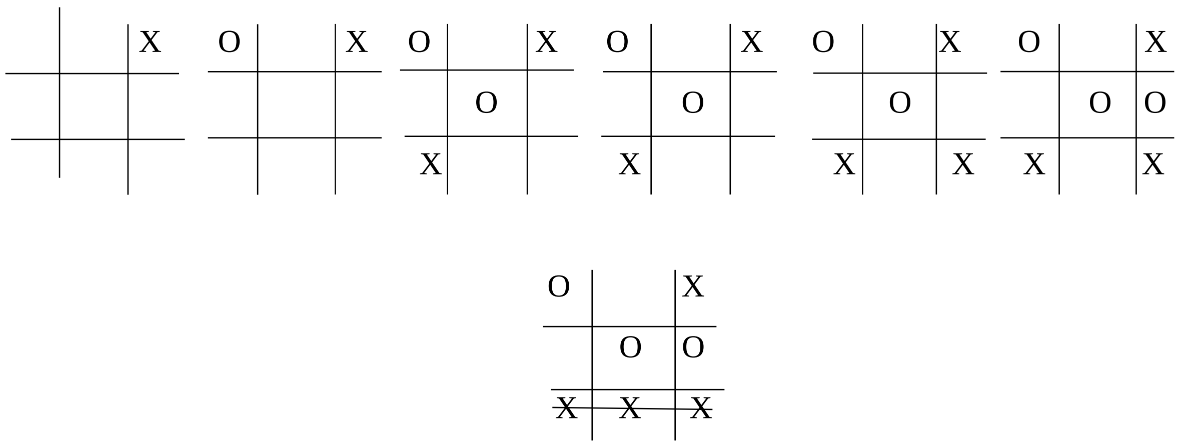


Fig 3.1:Player winning the game

A player can play perfect tic-tac-toe(win or draw) given they move according to the highest possible moves:

1. Win: If the player has two in a row, play the third to get three in a row.
2. Block: If the opponent has two in arrow, play the third to block them.
3. Fork; Create an opportunity where you can win in two days.
4. Block opponent's fork:
   - Option 1: Create two in a row to force the opponent into defending ad long as it doesn't result in them creating a fork or winning. For example, if 'X' has a corner,'O' has the centre and 'X' has the opposite corner as well,'O' must not play a corner in order to win.(Playing a corner in this scenario creates a fork for 'X' to win.)

- Option2: If there is a configuration where the opponent can fork,block that fork.

5. Centre:Play the centre.
6. Opposite corner:If the opponenf is in the corner,play the opposite corner.
7. Empty corner:Play in a corner square.
8. Empty side: Play in a middle square on any of the 4 sides.

## 3.2FLOW CHART

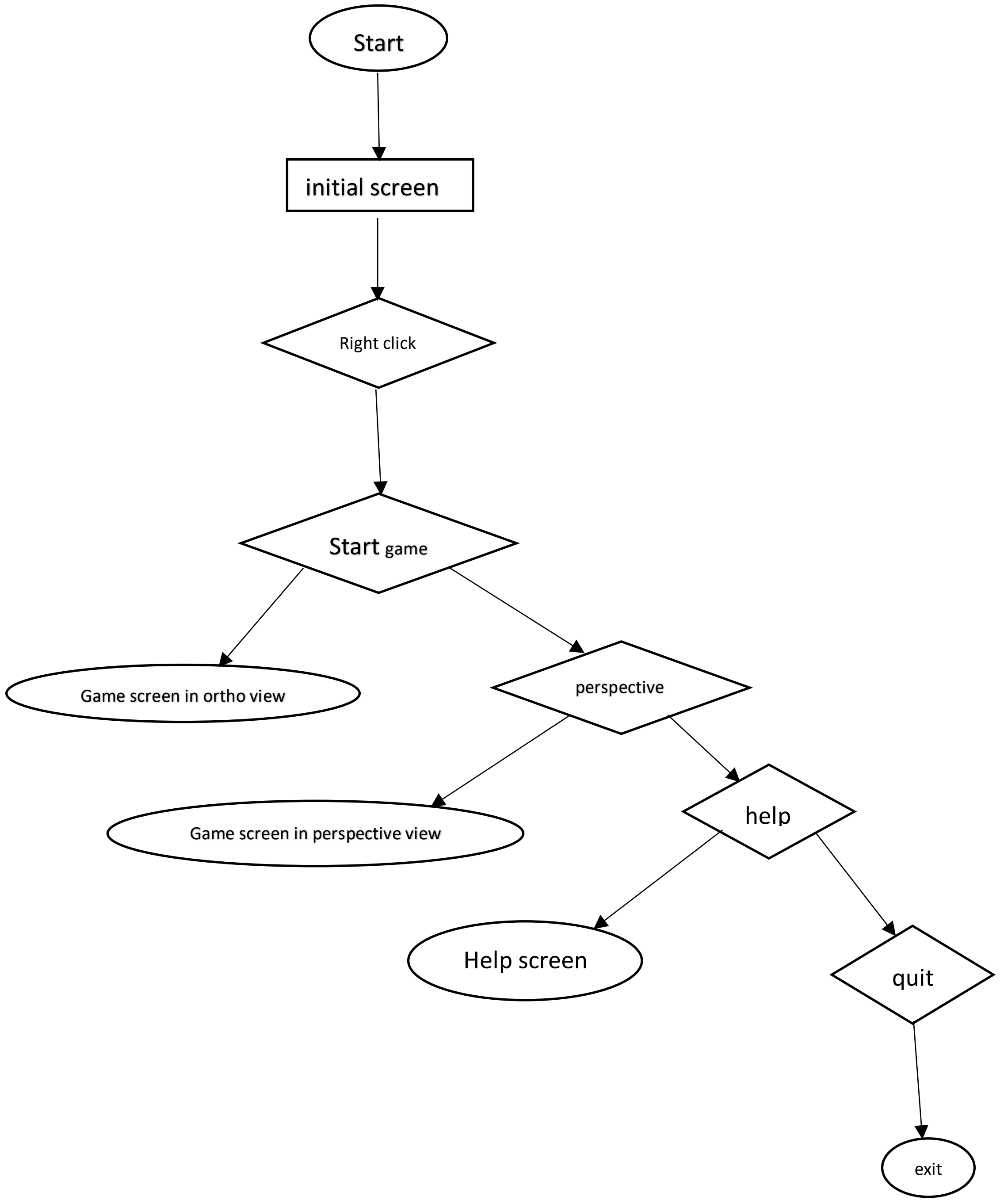


Fig 3.2:Flow chart of Tic-Tac-Toe game

## 3.3 SYSTEM DESIGN

### EXISTING SYSTEM

Existing system for a graphics is the TC++ . This system will support only the 2D graphics. 2D graphics package being designed should be easy to use and understand. It should provide various option such as free hand drawing , line drawing , polygon drawing , filled polygons, flood fill, translation , rotation , scaling , clipping etc. Even though these properties were supported, it was difficult to render 2D graphics cannot be . Very difficult to get a 3Dimensional object. Even the effects like lighting , shading cannot be provided. So we go for Eclipse software.

### PROPOSED SYSTEM :

To achieve three dimensional effects, OpenGL software is proposed . It is software which provides a graphical interface. It is a interface between application program and graphics hardware. the advantages are:

1.OpenGL is designed as a streamlined.

2. It is a hardware independent interface i.e. it can be implemented on many different hardware platforms.
3. With OpenGL, we can draw a small set of geometric primitives such as points, lines and polygons etc.
4. Its provides double buffering which is vital in providing transformations.
5. It is event driven software.
6. It provides call back function.

## 3.4 DETAILED DESIGN

TRANSFORMATION FUNCTIONS

Matrices allow arbitrary linear transformations to be represented in a consistent format, suitable for computation. This also allows transformations to be concatenated easily (by multiplying their matrices).

Linear transformations are not the only ones that can be represented by matrices. Using homogenous coordinates,both affine transformation and perspective projection on Rn can be represented as linear transformations on RPn+1 (that is, n+1dimensional real projective space). For this reason, 4x4 transformation matrices are widely used in 3D computer graphics.

3-by-3 or 4-by-4 transformation matrices containing homogeneous coordinates are often called, somewhat improperly, "homogeneous transformation matrices". However, the transformations they represent are, in most cases, definitely non-homogeneous and nonlinear (like translation, roto-translation or perspective projection). And even the matrices themselves look rather heterogeneous, i.e. composed of different kinds of elements (see below). Because they are multi-purpose transformation matrices, capable of representing both affine and projective transformations, they might be called "general transformation matrices", or, depending on the application, "affine transformation" or "perspective projection" matrices. Moreover, since the homogeneous coordinates describe a projective vector space, they can also be called "projective space transformation matrices".

FINDING THE MATRIX OF A TRANSFORMATION

If one has a linear transformation T(x) in functional form, it is easy to determine the transformation matrix **A** by simply transforming each of the vectors of the standard basis by T and then inserting the results into the columns of a matrix. In other words,

$$\mathbf{A} = [T(\vec{e_1}) \quad T(\vec{e_2}) \quad \dots \quad T(\vec{e_n})$$

For example, the function T(x) = 5x is a linear transformation. Applying the above process (suppose that n = 2 in this case) reveals that

$$T(x) = 5x = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$$

## *Examples in 2D graphics*

Most common geometric transformations that keep the origin fixed are linear, including rotation, scaling, shearing, reflection, and orthogonal projection; if an affine transformation is not a pure translation it keeps some point fixed, and that point can be chosen as origin to make the transformation linear. In two dimensions, linear transformations can be represented using a 2×2 transformation matrix.

### **Rotation**

For rotation by an angle θ anticlockwise about the origin, the functional form is $x' = x\cos\theta - y\sin\theta$ and $y' = x\sin\theta + y\cos\theta$. Written in matrix form, this becomes:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Similarly, for a rotation clockwise about the origin, the functional form is x' = xcosθ + ysinθ and y' = − xsinθ + ycosθ and the matrix form is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} \longrightarrow \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

**Scaling**

For scaling (that is, enlarging or shrinking), we have $x' = Sx.X$ and $y' = Sy.Y$. The matrix form is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} \quad \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

When $SxSy=1$ , then the matrix is a squeeze mapping and preserves areas in the plane.

# CHAPTER 4

# IMPLEMENTATION

## 4.1 FUNCTIONS USED

Void glColor3f(float red, float green, float blue);

This function is used to mention the color in which the pixel should appear. The number 3 specifies the number of arguments that the function would take. 'f ' gives the type that is float. The arguments are in the order RGB(Red, Green, Blue). The color of the pixel can be specified as the combination of these 3 primary colors.

Void glClearColor(int red, int green, int blue, int alpha);

This function is used to clear the color of the screen. The 4 values that are passed as arguments for this function are (RED, GREEN, BLUE, ALPHA) where the red green and blue components are taken to set the background color and alpha is a value that specifies depth of the window. It is used for 3D images.

Void glutKeyboardFunc(); void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));

where func is the new keyboard callback function. glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

Void glFlush();

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

Void glMatrixMode(GLenum  mode);

where  mode Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted:  GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE.  The initial value is GL_MODELVIEW.

glMatrixMode sets the current matrix mode.  mode can assume one of three values:

GL_MODELVIEW: Applies subsequent matrix operations to the model view matrix stack.

GL_PROJECTION :  Applies subsequent matrix operations to the projection matrix stack.

void glViewport(GLint x, GLint y, GLsizei width, GLsizei height)

where  x, y Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0, 0). width, height Specify the width and height of the viewport. When a GL context is first attached to a surface (e.g. window), width and height are set to the dimensions of that surface. glViewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Let (xnd, ynd) be normalized device coordinates. Then the window coordinates (xw, yw) are computed as follows:

xw = ( xnd + 1 ) width/2 + x yw = ( ynd + 1 ) height/2 + y

Viewport width and height are silently clamped to a range that depends on the implementation. To query this range, call  glGetInteger  with argument GL_MAX_VIEWPORT_DIMS.

void glutInit(int *argc, char **argv);

glutInit

will initialize the GLUT library and negotiate a session with the window system.

During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options.glutInit also processes command line options, but the specific options parse are window system dependent.

void glutReshapeFunc(void (*func)(int width, int height));

glutReshapeFunc sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width andheight parameters of the callback specify the new window size in pixels. Before the callback, the current window is set to the window that has been reshaped.

If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply

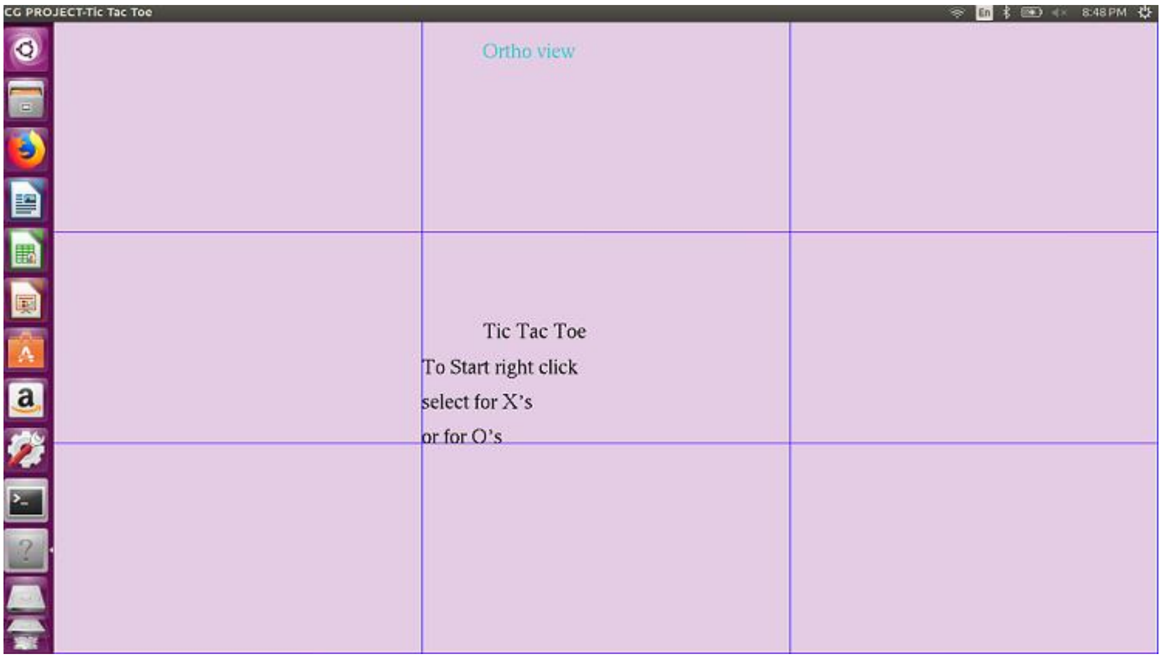
void glutMainLoop(void);

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never

glutPostRedisplay(): glutPostRedisplay, glutPostWindowRedisplay — marks the current or specified window as needing to be redisplayed.

# CHAPTER 5

## SNAPSHOTS



Tic Tac Toe
To Start right click
select for X's
or for O's

**Fig 5.1 : INSTRUCTION BEFORE STARTING THE GAME**

**(ortho view)**

The following picture shows the starting screen,before the game is started. The instruction about how to start the game is given to the player. There are 2 views available for the player to view the game:ortho and perspective. In this figure the view is ortho.

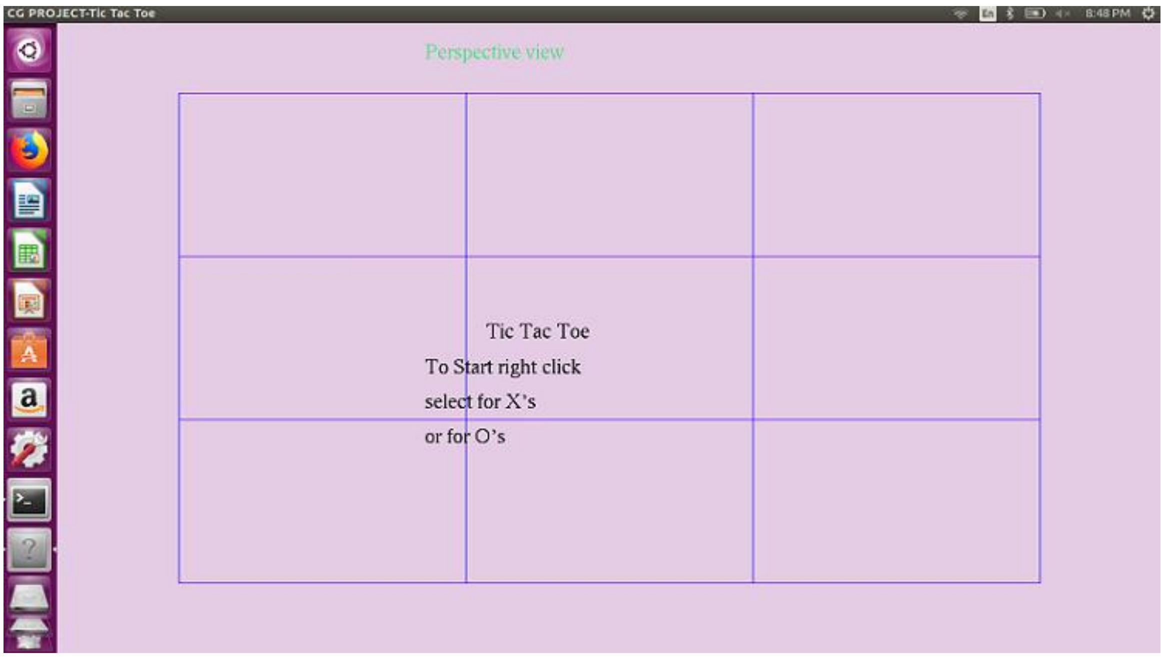


**Fig 5.2: Perspective view**

In the following figure, the view is perspective. The player can view the game in perspective view. The ortho and perspective view can be changed by using the 'w' key.The player can choose the view according to their interest.

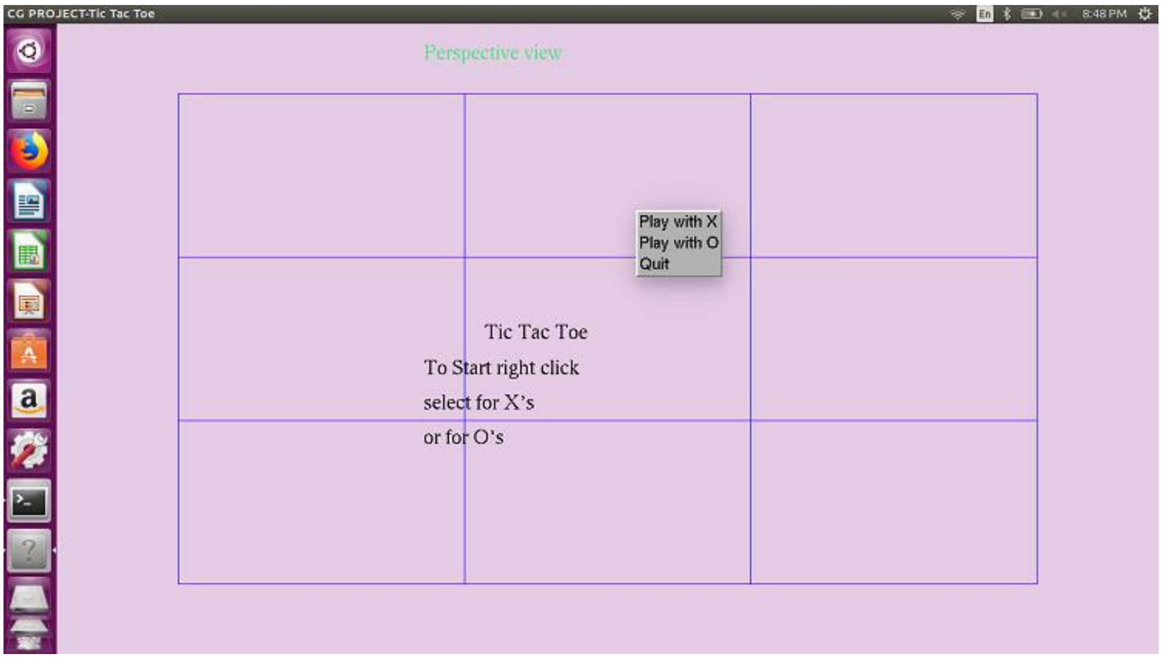


**Fig 5.3: PLAYER's SELECTION**

The figure shows the options provided to the player before starting the game. Menu containing the options are provided. The menu options are : play with X, play with O and quit. The player can choose to either play as X or O or quit the game.

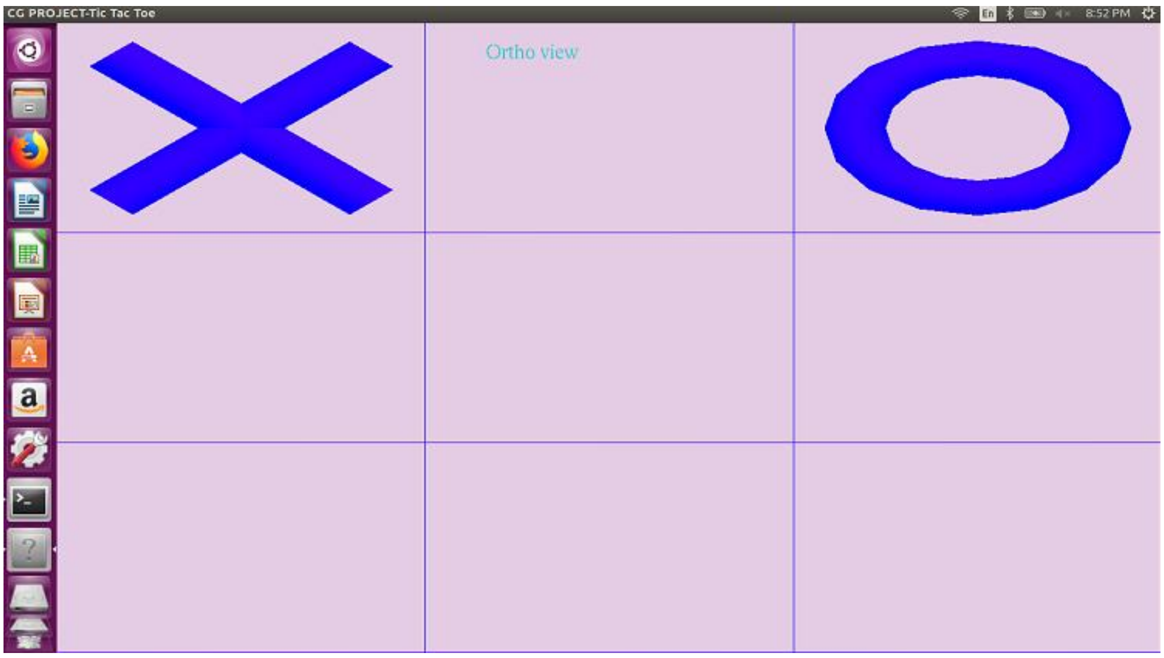
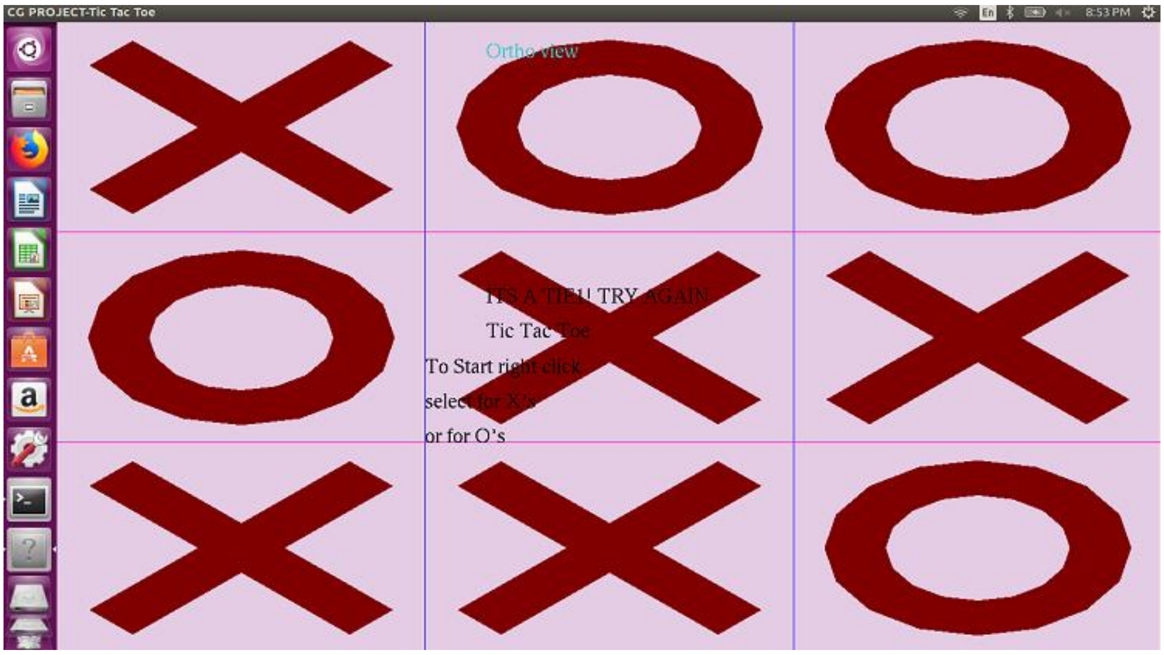


**Fig 5.4: Starting the game**

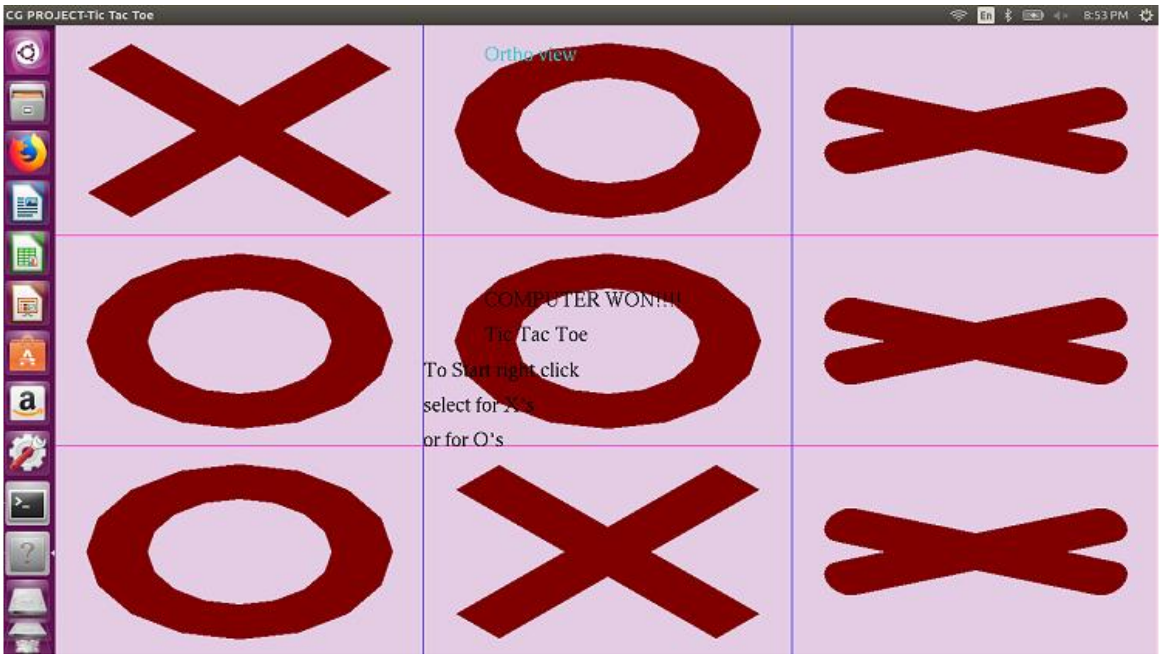
The player starts the game by either choosing to play as X or O. The computer automatically plays as the opponent by choosing the opposite symbol. The player can choose to play the game in 2 colours : blue or brown.



ITS A TIE!! TRY AGAIN
Tic Tac Toe
To Start right click
select for X's
or for O's

**Fig 5.5 : TIE CONDITION**

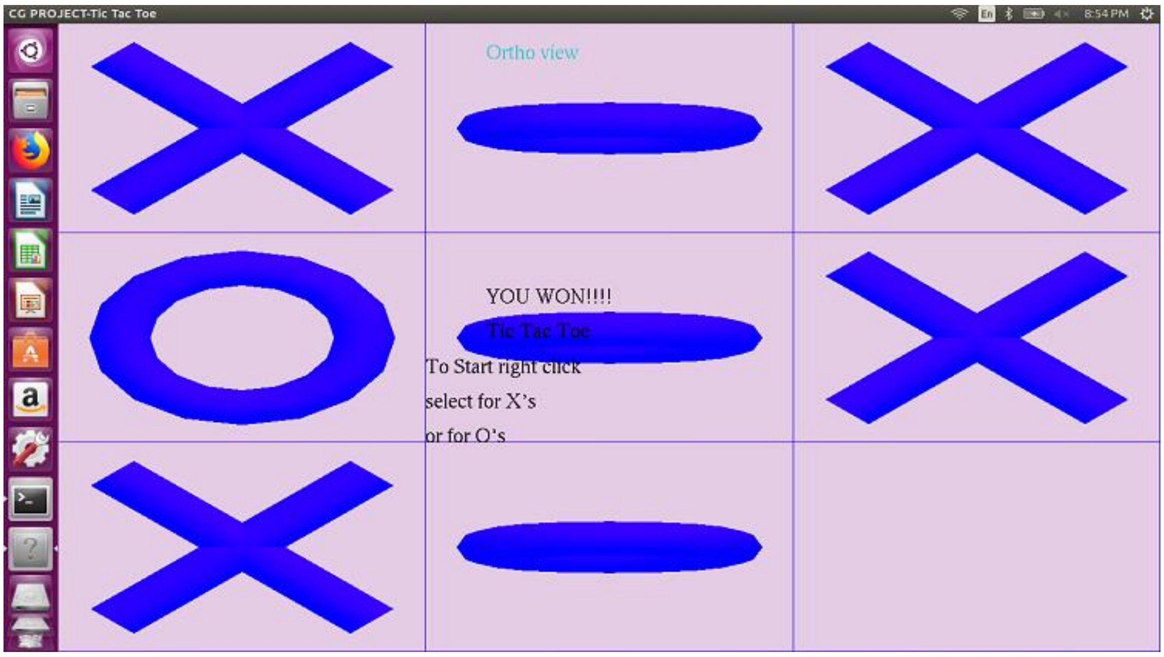
The player or the computer can win the game in 3 ways: by joining the corresponding symbols horizontally, vertically or diagonally. When that does not happen, neither the computer nor the player can win the game and the game ends up in a tie condition between the player and the computer.



COMPUTER WON!!!!
Tic Tac Toe
To Start right click
select for X's
or for O's

**Fig 5.6 : COMPUTER WINS**

In the above figure player chooses to play as O, hence computer plays as X. Computer wins by arranging the Xs vertically.



YOU WON!!!!
Tic Tac Toe
To Start right click
select for X's
or for O's

**Fig 5.7 : PLAYER's WIN**

The player chooses to play as O and the computer automatically plays as X. The player wins the game against the computer by arranging Os vertically. Hence O rotates.

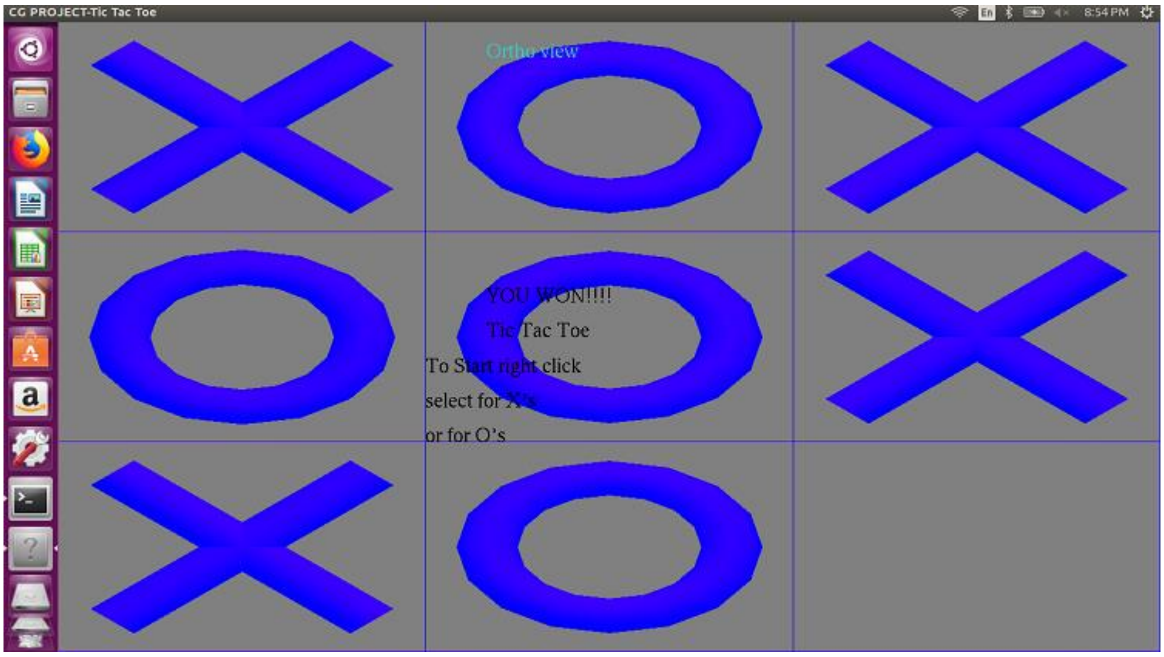


**Fig 5.8 : NIGHT MODE**

The player is provided with 2 background options: night and day view. Night view has a dimmer background. Player can choose the night view by choosing the ‘n’ button.

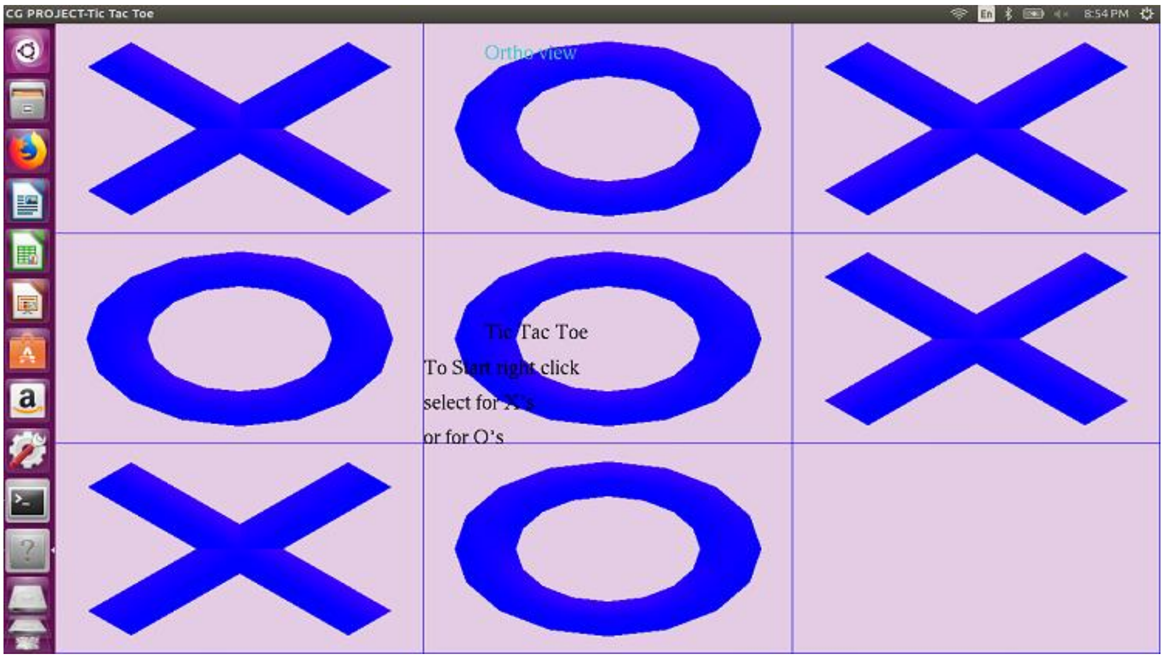


**Fig 5.9 : DAY MODE**

Day view has a brighter background. The player can choose this background by clicking the ‘d’ button.