## [VISVESVARAYA TECHNOLOGICAL UNIVERSITY](http://www.vtu.ac.in/)

##### “JNANA SANGAMA”, BELAGAVI-590018, KARNATAKA



**A Mini Project Report On**

S

***“TIC TAC TOE GAME”***

##### Submitted in the partial fulfillment of the requirement for the award of degree of

**BACHELOR OF ENGINEERING IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted By Vaibhav Mehta**

**(1VA14CS060)**

**Under the Guidance of**

|  |  |  |
| --- | --- | --- |
| **Prof. Sukruth Gowda M A Assistant Professor,**  **Dept., of CS&E** | **Prof. Chithra Apoorva D A Assistant Professor,**  **Dept., of CS&E** | **Prof. Mala B M Assistant Professor,**  **Dept., of IS&E** |



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## SAI VIDYA INSTITUTE OF TECHNOLOGY

**(Affiliated to Visvesvaraya Technological University, Belagavi | Recognized by Govt. of Karnataka | Approved by AICTE, New Delhi)**

##### RAJANUKUNTE, BENGALURU – 560 064

**2016-17**

**SAI VIDYA INSTITUTE OF TECHNOLOGY**

**(Affiliated to Visvesvaraya Technological University, Belagavi | Recognized by Govt. of Karnataka | Approved by AICTE, New Delhi)**

**Rajanukunte, Bengaluru- 560 064**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**CERTIFICATE**

Certified that the Mini Project work entitled *“****TIC TAC TOE GAME****”* carried out by **Mr. KUNAL KISHAN SEHRA (1VA14CS021),** a bonafide student of **SAI VIDYA INSTITUTE OF TECHNOLOGY**, Bengaluru, in partial fulfillment for the award of Bachelor of Engineering in Computer Science & Engineering of **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**, Belagavi during the year **2016-17.** It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for the said Degree.

|  |  |  |
| --- | --- | --- |
| **Prof. Sukruth Gowda M A**  Assistant Professor, Dept., of CS&E |  | **Prof. Chithra Apoorva D A**  Assistant Professor, Dept., of CS&E |
| **Dr. K G Mohan** Professor & Head, Dept., of CS&E |  | **Dr. H S Ramesh Babu**  Principal |

|  |  |
| --- | --- |
| **Name** | **Signature** |
| **1. \_\_\_ \_ \_ \_ \_ \_ \_ \_ \_** | **\_\_\_\_\_\_ \_ \_ \_ \_ \_ \_** |
| **2. \_\_\_ \_ \_ \_ \_ \_ \_ \_ \_** | **\_\_\_\_\_\_ \_ \_ \_ \_ \_ \_** |

# ABSTRACT

The aim of the mini project is to implement the Tic-Tac-Toe game. The Tic-Tac-Toe game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row first wins the game. Because of the simplicity of tic-tac-toe, it is often used as a pedagogical tool for teaching the concepts of good sportsmanship.

Tic-tac-toe is not a very challenging game for human beings. If you’re an enthusiast, you’ve probably moved from the basic game to some variant like threedimensional tic-tac-toe on a larger grid. If you sit down right now to play ordinary three-by-three tic-tac-toe with a friend, what will probably happen is that every game will come out a tie. Both you and your friend can probably play perfectly, never making a mistake that would allow your opponent to win. But can you describe how you know where to move each turn? Most of the time, you probably aren’t even aware of alternative possibilities; you just look at the board and instantly know where you want to move. That kind of instant knowledge is great for human beings, because it makes you a fast player. But it isn’t much help in writing a computer program

.For that, you have to know very explicitly what your strategy is.

Our project gives user the option to select between different Themes and also shows the current score of the players.

The completion of project brings with and sense of satisfaction, but it is never completed without thanking the persons who are all responsible for its successful completion. First and foremost I wish to express my deep sincere feelings of gratitude to our Institution, **Sai Vidya Institute of Technology**, for providing us an opportunity to do our education.

I would like to thank the **Management** and **Prof. M R Holla,** Director, Sai Vidya Institute of Technology for providing the facilities.

I extend my deep sense of sincere gratitude to **Dr. H S Ramesh Babu**, Principal, Sai Vidya Institute of Technology, Bengaluru, for having permitted to carry out the project work on Tic Tac Toe Game successfully**.**

I am thankful to **Prof. A M Padma Reddy,** Professor and Dean (Student affairs), Department of Computer Science and Engineering, Sai Vidya Institute of Technology, for his constant support and motivation.

I express my heartfelt sincere gratitude to **Dr. K G Mohan**, Professor and Head, Department of Computer Science and Engineering, Sai Vidya Institute of Technology, Bengaluru, for his valuable suggestions and support.

I express my special in-depth, heartfelt, sincere gratitude to **Prof. Sukruth Gowda M A,** Asst. Prof, Dept., of CS&E, **Prof. Chithra Apoorva D A,** Asst. Prof, Dept., of CS&E **Prof. Mala B M**, Asst. Prof, Dept., of IS&E, Sai Vidya Institute of Technology, Bengaluru, for their constant support in completing the project.

Finally, I would like to thank all the Teaching, Technical faculty and supporting staff members of Department of Computer Science and Engineering, Sai Vidya Institute of Technology, Bengaluru, for their support.

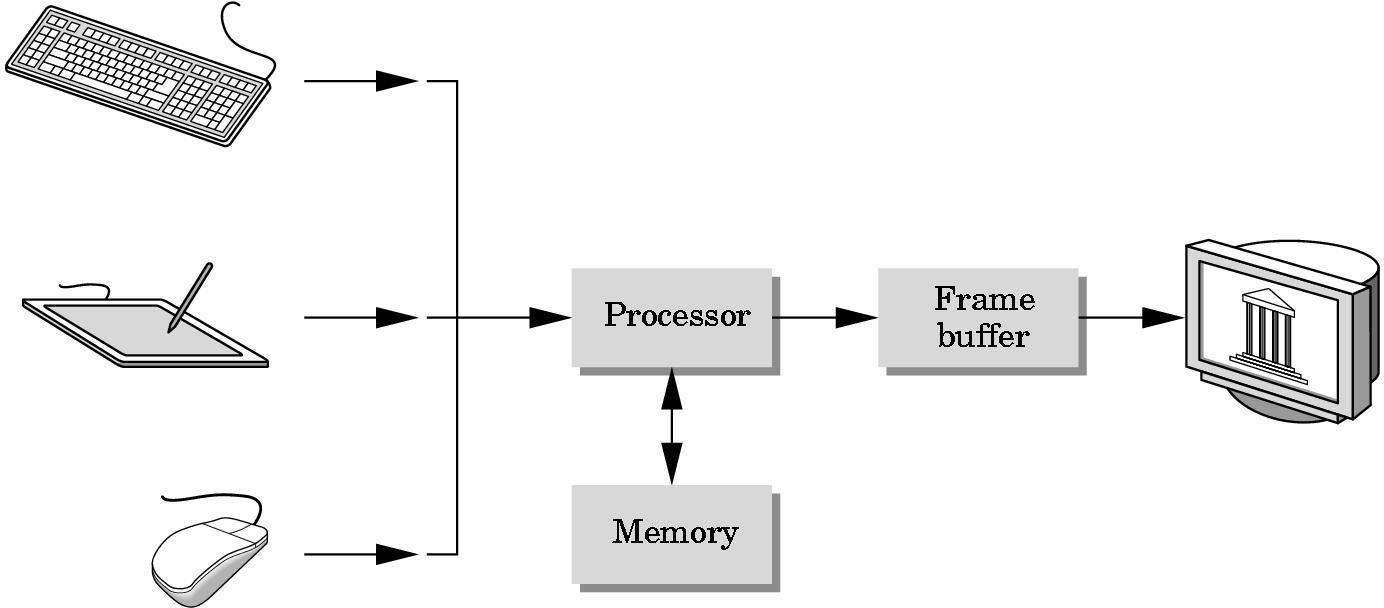
|  |  |
| --- | --- |
| Vaibhav Mehta | 1VA15CS060 |

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Chapter Title** | **Page No.** |
| **CHAPTER 1** | **INTRODUCTION** | 1-4 |
| 1.1 | Computer Graphics | 1 |
| 1.2 | Uses of Computer Graphics | 2 |
| 1.3 | OpenGL Technology | 2 |
| **CHAPTER 2** | **REQUIREMENTS AND SPECIFICATION** | 5 |
| 2.1 | Hardware Requirements | 5 |
| 2.2 | Software Requirements | 5 |
| **CHAPTER 3** | **FUNCTIONS USED** | 6-8 |
| **CHAPTER 4** | **SYSTEM DESIGN** | 9 |
| **CHAPTER 5** | **IMPLEMENTATION** | 10-14 |
| **CHAPTER 6** | **CONCLUTION** | 15 |
| **BIBLIOGRAPHY** | |  |
| **APPENDIX A** | **ABBREVIATIONS & ACRONYMS** |  |
| **APPENDIX B** | **SNAPSHOTS** |  |

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **CAPTION** | **PAGENO** |
| 1.1 | A Graphic System | 1 |
| 1.3.1  1.3.2 | Library Organization  OpenGL Graphics Architecture | 3  3 |
|  | Snapshot 1:Starting Window Snapshot 2:Different Themes Snapshot 3:Winner Window  Snapshot 4:Scoreboard Change |  |

## CHAPTER 1

**INTRODUCTION**

* 1. **Computer Graphics**
     + Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D Or 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly.
     + Computers have become a powerful medium for the rapid and economical production of pictures.
     + There is virtually no area in which graphical displays cannot be used to some advantage.
     + Graphics provide a so natural means of communicating with the computer that they have become widespread.
     + Interactive graphics is the most important means of producing pictures since the invention of photography and television.
     + We can make pictures of not only the real world objects but also of abstract objects such as mathematical surfaces on 4D and of data that have no inherent geometry.
     + A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.

##### Figure 1.1 A graphics system

***1***

## Uses of computer graphics:

#### User interface:

* + - * It is now a well-established fact that graphical interfaces provide an alternative and easy interaction between users and computers the built in graphics provided with user interfaces use the control items.
      * In industry, business government and education organization’s computer graphics is most commonly used to create 2D and 3D graphs of mathematical, physical and economic functions in the form of histograms, bars and pie charts which are very useful in decision making.

#### Computer aided drafting and design:

* + - * The computer aided drafting uses the graphics to components and systems. Electrical, mechanical and electronic devices such as automobile bodies, structure of airplane, ships, buildings.

#### Simulation and animation for scientific visualization and Env.:

* + - * Use of graphics in simulation makes mathematical models and mechanical systems more realistic and easy to study. The interactive graphics supported by animation

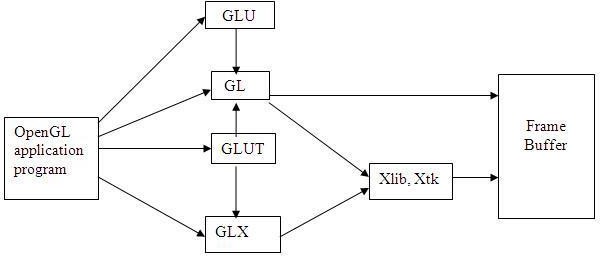
software proved their use in production of animated movies and cartoon films.

## 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.

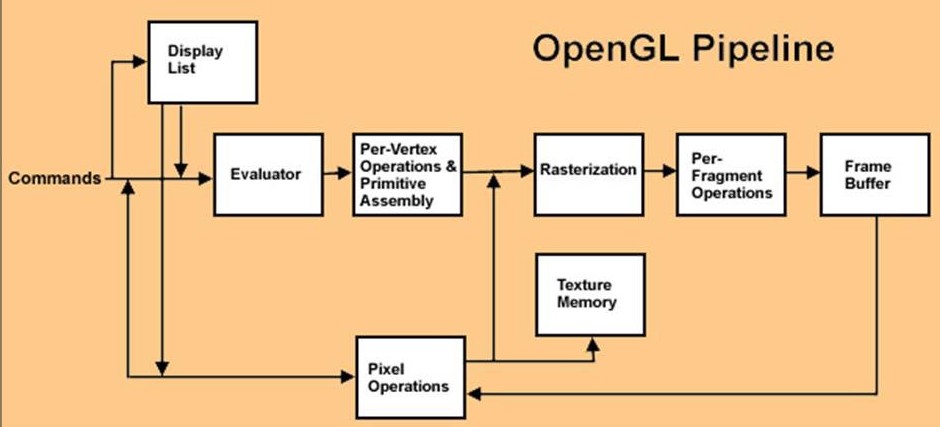
##### The OpenGL interface

Our application will be designed to access OpenGL directly through functions in three libraries namely: gl, glu, and glut.



##### Figure 1.3.1 Library Organization

* + 1. **OpenGL Graphics Architecture:**



**Figure 1.3.2 OpenGL Graphics Architecture**

* + - * **Display Lists:**

All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

#### Evaluators:

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions.

#### Per Vertex Operations:

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

#### Primitive Assembly:

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half space, defined by a plane.

#### Pixel Operation:

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the Rasterization step.

#### Rasterization:

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Color and depth values are assigned for each fragment square.

#### Fragment Operations:

Before values are actually stored into the frame buffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

# REQUIREMENTS AND SPECIFICATIONS

## Hardware Requirements

* + - The standard output device is assumed to be a **Colour Monitor**. It is quite essential for any graphics package to have this, as provision of colour 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 colour should be possible.

## Software Requirements

* + - The editor has been implemented on the OpenGL platform and mainly requires an appropriate version of Microsoft Visual Studio to be installed and functional in Windows.
    - 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:** Windows 10 version 1703
    - **Language Used In Coding:** C-language
    - **Tool Used In Coding:** Microsoft Visual Studio 2008

# 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**.

**GL\_MODELVIEW** Applies subsequent matrix operations to the modelview 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 and height 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 call glViewport(0,0,width,height) on the normal plane (and on the overlay if one exists).

If an overlay is established for the window, a single reshape callback is generated. It is the callback's responsibility to update both the normal plane and overlay for the window (changing the layer in use as necessary).

#### 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 return. It will call as necessary any callbacks that have been registered.

#### glutPostRedisplay ();

Mark the normal plane of *current window* as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

# SYSTEM DESIGN

START

WINDOWS FUNC

INTIAILAITION

|  |  |
| --- | --- |
| RESHAPE FUNC |  |
|  |
|  | |
| DISPLAY FUNC |  |
|  |
| KEYBOARD FUNC  MOUSE FUNC | |
| IDLE |  |
|  |

STOP

**4.1** Flow Chart for the program

LOOP BACK

# IMPLEMENTATION

## Source Code

##### Function to set the board for Tic Tac Toe Game

void Intialize()

{

turn=1;

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

board[i][j]=0;

}

##### Function to draw the game interface

void DrawLines()

{

glLineWidth(4.0); glBegin(GL\_LINES);

glColor3fv(v[ch][0]); glVertex2f(2,50); glVertex2f(2,380);

glVertex2f(100,50); glVertex2f(100,410); glVertex2f(200,50); glVertex2f(200,410); glVertex2f(299,50); glVertex2f(299,380); glVertex2f(50,350); glVertex2f(50,380); glVertex2f(150,350); glVertex2f(150,380); glVertex2f(250,350); glVertex2f(250,380);

glEnd();

glVertex2f(0,50); glVertex2f(300,50); glVertex2f(0,150); glVertex2f(300,150); glVertex2f(0,250); glVertex2f(300,250); glVertex2f(0,350); glVertex2f(300,350); glVertex2f(0,380); glVertex2f(300,380); glVertex2f(100,410); glVertex2f(200,410);

glLineWidth(1.f);

}

##### Function to check if there is any winner

bool CheckWinner()

{

int i, j;

for(i=0;i<3;i++)// horizontal check

{

for(j=1;j<3;j++)

{

if(board[i][0]!=0 && board[i][0]==board[i][j]) if(j==2) return true;

else break;

}

}

for(i=0;i<3;i++)// vertical check

{

for(j=1;j<3;j++)

if(board[0][i]!=0 && board[0][i]==board[j][i]) if(j==2) return true;

else break;

}

}

if( (board[0][0]!=0 && board[0][0]==board[1][1] && board[0][0]==board[2][2])

|| (board[2][0]!=0 && board[2][0]==board[1][1] && board[2][0]==board[0][2]) ) return true; //Diagonal check

return false;

}

##### Function to check if there is draw

bool CheckIfDraw()

{

int i, j; bool draw;

for(i=0;i<3;i++)

for(j=0;j<3;j++)

if(board[i][j]==0) return false;

return true;

}

##### Function to display up everything

void Display()

{

glClear(GL\_COLOR\_BUFFER\_BIT); if(turn == 1)

DrawString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, "X's Turn", 105, 20);

else

DrawString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, "O's Turn", 105, 40);

DrawLines(); DrawXO();

if(CheckWinner() == true)

{

if(turn == 1)

{

}

else

{

}

}

over = true; result = 2;

over = true; result = 1;

else if(CheckIfDraw() == true)

{

over = true; result = 0;

}

if(over == true)

{

if(result == 0)

DrawString(GLUT\_BITMAP\_HELVETICA\_18, "It's a draw", 107, 400);

if(result == 1)

DrawString(GLUT\_BITMAP\_HELVETICA\_18, "X wins", 125, 400);

if(result == 2)

DrawString(GLUT\_BITMAP\_HELVETICA\_18, "O wins", 125, 400);

DrawString(GLUT\_BITMAP\_HELVETICA\_18, "Press 'y' to Continue | 'n' to Exit ", 20, 430);

}

DrawString(GLUT\_BITMAP\_TIMES\_ROMAN\_24,"X",18,372); DrawString(GLUT\_BITMAP\_HELVETICA\_18,pint(p[1]),62,370); DrawString(GLUT\_BITMAP\_TIMES\_ROMAN\_24,"O",118,372); DrawString(GLUT\_BITMAP\_HELVETICA\_18,pint(p[2]),162,370); DrawString(GLUT\_BITMAP\_TIMES\_ROMAN\_24,"D",218,372); DrawString(GLUT\_BITMAP\_HELVETICA\_18,pint(p[0]),262,370);

glutSwapBuffers();

}

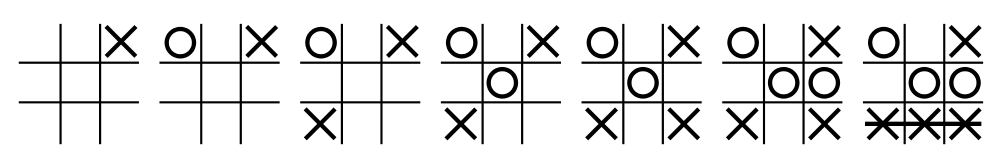
## CHAPTER 6

**CONCLUSION**

The code we have implemented for our project is working well to the best of our knowledge. This project is both informative and entertaining. This project provided an opportunity to learn the various concepts of the subject in detail and provided us a platform to express our creativity and imagination come true.

The full designing and creating of **Tic-Tac-Toe** has been executed under Windows operating system using MS Visual Studio, this platform provides a and satisfies the basic need of a good compiler. Using 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:



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 a row, play the third to block them.
3. **Fork**: Create an opportunity where you can win in two ways.

##### Block opponent's fork:

* + **Option 1**: Create two in a row to force the opponent into defending, as 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’s corner as well, "O" must not play a corner in order to win.
  + **Option 2**: If there is a configuration where the opponent can fork, block that fork.

1. **Centre**: Play the centre.
2. **Opposite corner**: If the opponent is in the corner, play the opposite corner.
3. **Empty corner**: Play in a corner square.
4. **Empty side**: Play in a middle square on any of the 4 sides.

***15***

# BIBLIOGRAPHY

### Reference Books

* “Interactive Computer Graphics – A Top-Down Approach using OpenGL” by EDWARD ANGEL

### Internet source

* [www.wikipedia.org](http://www.wikipedia.org/)
* [www.google.co.in](http://www.google.co.in/)
* [www.opengl.org](http://www.opengl.org/)

**APPENDIX A**

**ABBREVATION AND ACRONYMS**

**Short Form Expansion**

OpenGL Open Graphics Library

GLUT Graphics Library Utility Toolkit

GLU Graphics Language Utility

Env Environment

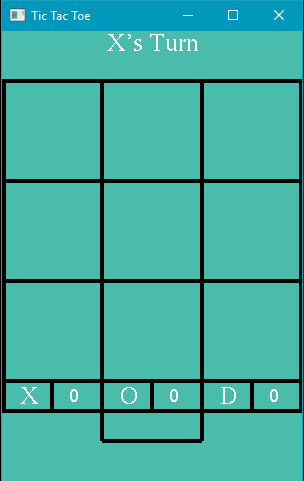
UI User Interface

API Application Programming Interface

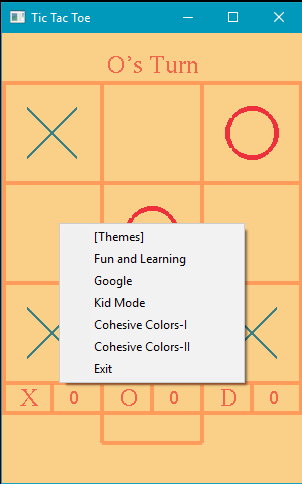
GL Graphics Library

## APPENDIX B

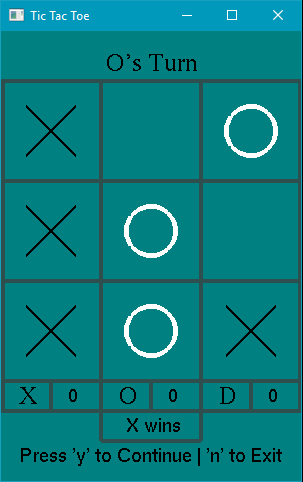
**SNAPSHOTS**



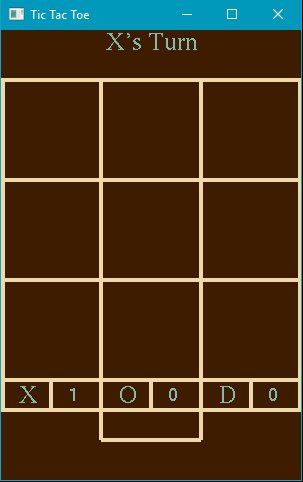
**Snapshot 1:** Starting Window of the game, here always player1 is X and player2 is O.



**Snapshot 2:** The Different Themes Available is shown in the menu



**Snapshot 3**:Here the player X wins the game as shown above, after that user are asked whether they want to continue using keyboard interface



**Snapshot 4:** After selecting pressing ‘y’ key from keyboard, the score of the previous game winner is incremented by one