# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JnanaSangama, Belagavi-590018, Karnataka, INDIA

**PROJECT REPORT**
**on**

## *"An Abstract Strategy Game of Connect 4 "*

Submitted in partial fulfillment of the requirements for the VI Semester

## Computer Graphics and Visualization Lab(10CSL67)

**Bachelor of Engineering**
**IN**
**COMPUTER SCIENCE AND ENGINEERING**

**For the Academic year**
**2016-2017**

**BY**


| | |
|---|---|
| **KARUNYA ATLURI** | **1PE14CS051** |
| **KAUSHIK N.** | **1PE14CS053** |


PES INSTITUTIONS


**Department of Computer Science and Engineering**
**PESIT BANGALORE SOUTH CAMPUS**
**Hosur Road, Bengaluru -560100**

# PESIT BANGALORE SOUTH CAMPUS

**Hosur Road, Bengaluru-560100**

## Department of Computer Science and Engineering


# *CERTIFICATE*

*This is to certify that the project entitles* ***"An Abstract Strategy Game of Connect 4"*** *is a bona fide work carried out by* ***KARUNYA ATLURI.*** *and* ***KAUSHIK N.*** *bearing USN* ***1PE14CS051*** *and* ***1PE14CS053*** respectively, *in Computer Graphics and Visualization Lab(10CSL67) for the 6th Semester in partial fulfillment for the award of Degree of Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belagavi during the year 2016-2017.*

---

***Signature of the guide***

**Dr. Sarasvathi V**
Associate Professor
PESIT BSC, Bengaluru.

---

***Signature of the HOD***

**Prof. Sandesh B J**
HOD, Dept. of CSE
PESIT BSC, Bengaluru.

# ACKNOWLEDGEMENT

We gratefully acknowledge the help lent out by all Staff Members of Computer Science and Engineering Department of PESIT-BSC at all difficult times.

We are indebted to our internal guide **Dr. Sarasvathi V, Associate Professor and Mrs. Shubha Raj K B, Assistant Professor** who has not only coordinated our work but also given suggestions from time to time and also seen to it, that all of us are doing well in the project work.

We are grateful to **Prof. Sandesh B J, Associate Professor** and **HOD** of Computer Science and Engineering Department who has seen to it, that all of us are doing well in the project work.

We take the opportunity to thank our beloved **Dr. J. Surya Prasad, Director/Principal** for all the help and encouragement throughout the project.

We express our sincere thanks to our loving friends, who have helped us directly or indirectly to make this project work successful.

**Karunya Atluri**

**Kaushik N.**

# ABSTRACT

This project basically deals with the implementation of an abstract strategy game called Connect-4 which is a two-player connection game where the players first choose a color and then take turns dropping colored discs from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the next available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs. The main part of this project's core will be implementing Connect 4 in C++, reading in moves and printing the board with the correct move inserted into the board. The program will also be given a basic graphical user interface (GUI) to help with the visuals, written in OpenGL. It involves two main paths: *AI* and *Graphics*. For AI (artificial intelligence), we have incorporated the logic of a computer opponent in Connect 4. For Graphics, we have used OpenGL and expanded on the already given GUI to create a unique look and feel for the game. We have added a sound library that plays a background wav music in the background in a loopback manner.

# 1. Introduction to OpenGL

## 1.1 OpenGL

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

OpenGL bases on the state variables. There are many values, for example the color, that remain after being specified. That means, you can specify a color once and draw several polygons, lines or whatever with this color then. There are no classes like in DirectX. However, it is logically structured. Before we come to the commands themselves, here is another thing:

To be hardware independent, OpenGL provides its own data types. They all begin with "GL". For example GLfloat, GLint and so on. There are also many symbolic constants, they all begin with "GL_", like GL_POINTS, GL_POLYGON. Finally the commands have the prefix "gl" like *glVertex3f ()*. There is a utility library called GLU, here the prefixes are "GLU_" and "glu". GLUT commands begin with "glut", it is the same for every library

## 1.2 Glut

GLUT is a complete API written by Mark Kilgard which facilitates creation of windows and handling messages. It exists for several platforms, that means that a program which uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.

GLUT provides some routines for the initialization and creating the window (or fullscreen mode). Those functions are called first in a GLUT application:
In the first line always g*lutInit(&argc, argv) is written* . After this, GLUT must be told which display mode is required – single or double buffering, color index mode or RGB and so on. This is done by calling *glutInitDisplayMode()*. The symbolic constants are connected by a logical OR, so you could use *glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE)*.

# 2. PROJECT DESCRIPTION

## 2.1 SOFTWARE AND HARDWARE SPECIFICATION.

### 2.1.1 HARDWARE

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 move the car left or right in the game. A **keyboard** is used for controlling and inputting data in the form of characters, numbers i.e.. 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.

### 2.1.2 SOFTWARE

**IDE :** CodeBlocks
**LIBRARIES :** FreeGlut
**OS :** Ubuntu

## 2.2 SYSTEM DESIGN

### glutDisplayFunc(display);

**glutDisplayFunc(void(*func)(void))** is the first and most important event callback function. Whenever GLUT determine the contents of the window need to be redisplayed, the callback function registered by glutDisplayFunc is executed.

## glutReshapeFunc(Reshape);

**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, the default reshape callback is used. This default callback will simply call glViewport(0,0,width,height) on the normal plane.

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

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 state of modifier keys such as Shift cannot be determined directly; their
only effect will be on the returned ASCII data. 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.

## glutBitmapCharacter(font,*c)

**glutBitmapCharacter** renders a bitmap character using OpenGL.

Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font. The available fonts are:

void glutBitmapCharacter(void *font, int character);
GLUT_BITMAP_8_BY_13
GLUT_BITMAP_9_BY_15
GLUT_BITMAP_TIMES_ROMAN_10
GLUT_BITMAP_TIMES_ROMAN_24
GLUT_BITMAP_HELVETICA_10
GLUT_BITMAP_HELVETICA_12
GLUT_BITMAP_HELVETICA_18

## glTranslatef(LGfloat x,GLfloat y,GLfloat z);

**glTranslate :** multiply the current matrix by a translation matrix

Use glPushMatrix and glPopMatrix to save and restore the untranslated coordinate system.

## glRasterpos3f(x,y,z);

void **glRasterPos3f**(GLfloat *x*,GLfloat *y*,GLfloat *z*);

Specify the x, y, z, and w object coordinates (if present) for the raster position.

**glRasterPos :**specify the raster position for pixel operations

The GL maintains a 3D position in window coordinates. This position, called the raster position, is used to position pixel and bitmap write operations. It is maintained with subpixel accuracy.

## glShadeModel

void glShadeModel(GLenun *mode*)

**glShadeModel:** select flat or smooth shading.

GL primitives can have either flat or smooth shading. Smooth shading, the default, causes the computed colors of vertices to be interpolated as the primitive is rasterized, typically assigning different colors to each resulting pixel fragment. Flat shading selects the computed color of just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive.

# 3. APIs Used

## Initialization Functions

### 3.1 glutInit

### 3.2 glutInitWindowPosition, glutInitWindowSize

The glutInitWindowPosition and glutInitWindowSize functions specify a desired position and size for windows that *freeglut* will create in the future.

**Usage**

void glutInitWindowPosition ( int x, int y );
void glutInitWindowSize ( int width, int height );

## Description

The glutInitWindowPosition and glutInitWindowSize functions specify a desired position and size for windows that *freeglut* will create in the future. The position is measured in pixels from the upper left hand corner of the screen, with "x" increasing to the right and "y" increasing towards the bottom of the screen. The size is measured in pixels. *Freeglut* does not promise to follow these specifications in creating its windows, but it certainly makes an attempt to.

## Event Processing Functions

### 3.2 glutMainLoop

The glutMainLoop function enters the event loop.

**Usage**

void glutMainLoop ( void );

**Description**

The glutMainLoop function causes the program to enter the window event loop. An application should call this function at most once. It will call any application callback functions as required to process mouse clicks, mouse motion, key presses, and so on.

## Window Functions

### 3.3 glutCreateWindow

## Display Functions

### 3.4 glClear(GLbitfieldmask) :

glClear takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.

**Mask :**

Bitwise OR of masks that indicate the buffer to be cleared. The four masks are GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_ACCUM_BUFFER_BIT, and GL_STENCIL_BUFFER_BIT.

### 3.5 glColor3f(GLfloat red,GLfloat green,GLfloat blue):

glColor sets a new three valued RGB color. Current color values are stored in floating-point format. Unsigned integer color components are linearly mapped to floating-point values such that the largest representable value maps to 1.0 (full intensity), and 0 maps to 0.0 (zero intensity).

*red:* The new red value for the current color.

*green:* The new green value for the current color.

*blue:* The new blue value for the current color.

### 3.6 glClearColor(GLfloat red,GLfloat green,GLfloat blue) :

glClearColor specifics the red, green, blue and alpha values used by glClear to clear the color buffers. Values specified by gIClearColor are clamped to the range [0,1].

**red , green , blue , alpha :** Specify the red, green, blue and alpha values used when the color buffers are cleared. The initial values are all 0.

### 3.7 glVertex

specify a vertex

**Parameters** x, y, z, w

Specify x, y, z, and w coordinates of a vertex.

Not all parameters are present in all forms of the command.

id glVertex3f(GLfloat x, GLfloat y, GLfloat z);

**Description**

glVertex commands are used within glBegin/glEnd pairs to specify point, line, and polygon vertices. The current color, normal, texture coordinates, and fog coordinate are associated with the vertex when glVertex is called. When only x and y are specified, z defaults to 0 and w defaults to 1. When x,y, and z are specified, w defaults to 1.

### 3.8 glutPostRedisplay

glutPostRedisplay marks the *current window* as needing to be redisplayed.

**Usage**

void glutPostRedisplay(void);

**Description**

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.

### 3.9 glutPostWindowRedisplay

**Description**

Similar to glutPostRedisplay(), except that instead of affecting the current window , this function affects an arbitrary window, indicated by the windowID parameter.

### 3.10 glutSwapBuffers

glutSwapBuffers swaps the buffers of the *current window* if double buffered.

**Usage**

void glutSwapBuffers(void);

**Description**

Performs a buffer swap on the *layer in use* for the *current window*. Specifically, glutSwapBuffers promotes the contents of the back buffer of the *layer in use* of the *current window* to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffers is called.

An implicit glFlush is done by glutSwapBuffers before it returns. Subsequent OpenGL commands can be issued immediately after calling glutSwapBuffers, but are not executed until the buffer exchange is completed.

If the *layer in use* is not double buffered, glutSwapBuffers has no effect.

## Global Callback Registration Functions.

### 3.11 glutTimerFunc

glutTimerFunc registers a timer callback to be triggered in a specified number of milliseconds.

**Usage**

void glutTimerFunc(unsigned int msecs,
void (*func)(int value), value);

**Description**

glutTimerFunc registers the timer callback func to be triggered in at least msecs milliseconds. The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc. Multiple timer callbacks at same or differing times may be registered simultaneously.

The number of milliseconds is a lower bound on the time before the callback is generated. GLUT attempts to deliver the timer callback as soon as possible after the expiration of the callback's time interval.

There is no support for canceling a registered callback. Instead, ignore a callback based on its value parameter when it is triggered.

## Window-Specific Callback Registration Functions.

### 3.12 glutDisplayFunc

glutDisplayFunc sets the display callback for the *current window*.

**Usage**

void glutDisplayFunc(void (*func)(void));
func
The new display callback function.

**Description**

glutDisplayFunc sets the display callback for the *current window*. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the *current window* is set to the window needing to be redisplayed and (if no overlay display callback is registered) the *layer in use* is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback (this includes ancillary buffers if your program depends on their state).

### 3.13 glutKeyboardFunc

glutKeyboardFunc sets the keyboard callback for the *current window*.

**Usage**

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

func

The new keyboard callback function.

**Description**

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 state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. 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.

### 3.14 glutMouseFunc

`glutMouseFunc` sets the mouse callback for the *current window*.

**Usage**

```
void glutMouseFunc(void (*func)(int button, int state,
```

```
                                    int x, int y));
```

`func`

The new mouse callback function.

**Description**

`glutMouseFunc` sets the mouse callback for the *current window*. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The `button` parameter is one of `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON`, or `GLUT_RIGHT_BUTTON`.

### 3.15 glutEntryFunc

glutEntryFunc sets the mouse enter/leave callback for the *current window*.

**Usage**

void glutEntryFunc(void (*func)(int state));
func
The new entry callback function.

**Description**

glutEntryFunc sets the mouse enter/leave callback for the *current window*. The state callback parameter is either GLUT_LEFT or GLUT_ENTERED depending on if the mouse pointer has last left or entered the window.
Passing NULL to glutEntryFunc disables the generation of the mouse enter/leave callback.
Some window systems may not generate accurate enter/leave callbacks.

### 3.16 glutMotionFunc, glutPassiveMotionFunc

glutMotionFunc and glutPassiveMotionFunc set the motion and passive motion callbacks respectively for the *current window*.

**Usage**

void glutMotionFunc(void (*func)(int x, int y));

void glutPassiveMotionFunc(void (*func)(int x, int y));

func

The new motion or passive motion callback function.

**Description**

glutMotionFunc and glutPassiveMotionFunc set the motion and passive motion callback respectively for the *current window*. The motion callback for a window is called when the mouse moves within the window while one or more mouse buttons are pressed. The passive motion callback for a window is called when the mouse moves within the window while *no* mouse buttons are pressed.

The x and y callback parameters indicate the mouse location in window relative coordinates.

### 3.17 glMatrixMode (GLenum mode):

It switches matrix mode between the two matrices-

- MODEL_VIEW (GL_MODELVIEW)

### 3.18 glViewport (GLint x, GLint y, GLsizei width, GLsizei height):

It specifies that the viewport will have lower left corner (x,y) in screen coordinates and will be width pixels wide and height pixels high.

### 3.19 glOrtho

multiply the current matrix with an orthographic matrixC Specificationvoid glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal);

**Parameters**

left, right

Specify the coordinates for the left and right vertical clipping planes.

bottom, top

Specify the coordinates for the bottom and top horizontal clipping planes.

nearVal, farVal

Specify the distances to the nearer and farther depth clipping planes.

These values are negative if the plane is to be behind the viewer.

**Description**

glOrtho describes a transformation that produces a parallel projection.

## Font Rendering Functions.

### 3.20 glutBitmapCharacter

The glutBitmapCharacter function renders a single bitmapped character in the *current window* using the specified font.

**Usage**

void glutBitmapCharacter ( void *font, int character );

font The bitmapped font to use in rendering the character
character The ASCII code of the character to be rendered

**Description**

The glutBitmapCharacter function renders the given character in the specified bitmap font. *Freeglut* automatically sets the necessary pixel unpack storage modes and restores the existing modes when it has finished. Before the first call to glutBitMapCharacter the application program should call glRasterPos* to set the position of the character in the window. The glutBitmapCharacter function advances the cursor position as part of its call to glBitmap and so the application does not need to call glRasterPos* again for successive characters on the same line.

### 3.21 glutBitmapString

The glutBitmapString function renders a string of bitmapped characters in the *current window* using the specified font.

**Usage**

void glutBitmapString ( void *font, char *string );

font The bitmapped font to use in rendering the character string
string String of characters to be rendered

**Description**

The glutBitmapString function renders the given character string in the specified bitmap font. *Freeglut* automatically sets the necessary pixel unpack storage modes and restores the existing modes when it has finished. Before calling glutBitMapString the application program should call glRasterPos* to set the position of the string in the window. The glutBitmapString function handles carriage returns. Nonexistent characters are rendered as asterisks.

# 4. SOURCE CODE

**/*main.cpp*/**

```
#include "Game.h"
#include "Drawing.h"
#include "GL/glut.h"
#include <string>
#include <iostream>
using namespace std;
#include "RandomStrategy.h"
#include <stdio.h>
#include <irrKlang.h>
using namespace irrklang;


ISoundEngine *SoundEngine = createIrrKlangDevice();

#define UPDATE_INTERVAL_MS 20
#define CPU_THINK_INTERVAL 500
#define BOARD_WIDTH 7
#define BOARD_HEIGHT 6

#define HORIZONTAL_MARGIN 20
#define VERTICAL_MARGIN 20
#define TOP_MARGIN 160

#define MINVALUE(a, b) ((a) < (b) ? (a) : (b))

int gWindowHeight, gWindowWidth;
int gPlayerMouseColumn;int flag=0;

void myInit();
void drawScene();
void resize(int w, int h);
void keyInput(unsigned char key, int x, int y);
void mouseInput(int button, int state, int x, int y);
void mouseMove(int x, int y);
```

```
void mouseEnter(int state);
void timerFunction(int arg);
void cover();


Game game(BOARD_WIDTH, BOARD_HEIGHT);

void drawInfo();
void drawGrid();
void drawPlayerMove();
void drawGridMarks();

vector<Player*> players;
unsigned playerOne = 0, playerTwo = 0;
bool cpuPlaying = false;
float cpuThinkTime;

int main(int argc, char** argv)
{
   cout << "Program Interaction:\n";
   cout << "\tPress ESC anytime to restart game board.\n";
   cout << "\tUse keys 1 and 2 to change the players 1 and 2 controllers.\n";
   cout << "\tUse left mouse button to make a play (for human players).\n";

        glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);

        glutInitWindowSize (500, 600);
        glutInitWindowPosition (500, 200);
        glutCreateWindow ("Connect Four");
        glutDisplayFunc(drawScene);
        glutReshapeFunc(resize);
        glutKeyboardFunc(keyInput);
        glutPassiveMotionFunc(mouseMove);
```

```
        glutMouseFunc(mouseInput);
        glutEntryFunc(mouseEnter);
        glutMotionFunc(mouseMove);
        glutTimerFunc(UPDATE_INTERVAL_MS, timerFunction,
UPDATE_INTERVAL_MS);

        myInit();
        players.push_back(0);
   players.push_back(new RandomStrategy);
        game.setPlayers(0, 0);
        game.startNewGame();
        glutMainLoop();

        return 0;
}

void myInit()
{
   SoundEngine-
>play2D("C:\\Users\\kaush\\OneDrive\\Documents\\Original\\kids.ogg",true);

        glClearColor(1.f, 1.f, 1.f, 1.f);
}
void cover()
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        glClearColor(1.0,1.0,1.0,1.0);
      glColor3f(1.0, 0.08, 0.08);
      Draw::text("An OPENGL PROJECT on", 130, gWindowHeight -28 );
      glColor3f(0.08, 0.08, 1.0);
      Draw::text("CONNECT 4", 180, gWindowHeight -50);
      glColor3f(0.5, 0.0, 0.9);
      Draw::text("Submitted by:", 10, gWindowHeight -160);
```

```
glColor3f(0.5, 0.0, 0.9);
glBegin(GL_LINES);
glVertex3f(10.0,gWindowHeight -170,0.0);
glVertex3f(120.0,gWindowHeight -170,0.0);
glEnd();
glColor3f(0.08, 0.08, 1.0);
Draw::text("KAUSHIK N.", 20, gWindowHeight -250);
Draw::text("USN:1PE14CS053", 10, gWindowHeight -370);

glColor3f(1.0, 0.08, 0.08);
Draw::text("KARUNYA ATLURI.", 340, gWindowHeight -250);
Draw::text("USN:1PE14CS051",330, gWindowHeight -370);
glColor3f(0.08, 0.08, 1.0);
Draw::circle2d(70.0,300.0, 40.0, 30);
glColor3f(1.0, 0.08, 0.08);
Draw::circle2d(400.0,300.0, 40.0, 30);
Draw::text("On Behalf of", 190, gWindowHeight -430 );
glColor3f(0.08, 0.08, 1.0);
Draw::text("PES", 60, gWindowHeight -460 );
glColor3f(1.0, 0.08, 0.08);
Draw::text("IT", 96, gWindowHeight -460 );
Draw::text("B", 140, gWindowHeight -460 );
glColor3f(0.08, 0.08, 1.0);
Draw::text("ANGALORE", 153, gWindowHeight -460 );
glColor3f(1.0, 0.08, 0.08);
Draw::text("S", 280, gWindowHeight -460 );
glColor3f(0.08, 0.08, 1.0);
Draw::text("OUTH", 293, gWindowHeight -460 );
glColor3f(1.0, 0.08, 0.08);
Draw::text("C", 370, gWindowHeight -460 );
glColor3f(0.08, 0.08, 1.0);
Draw::text("AMPUS", 384, gWindowHeight -460 );
glColor3f(0.3, 0.5, 0.8);
Draw::text("Press", 150, gWindowHeight -580 );
```

```
        glColor3f(1.0, 0.08, 0.08);
        Draw::text("Enter", 200, gWindowHeight -580 );
        glColor3f(0.3, 0.5, 0.8);
        Draw::text("to Continue", 250, gWindowHeight -580 );
        glBegin(GL_QUADS);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0.0,gWindowHeight -70,0.0);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(600.0,gWindowHeight -70,0.0);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(600.0,gWindowHeight -120,0.0);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0.0,gWindowHeight -120,0.0);
        glEnd();
        glBegin(GL_QUADS);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0.0,gWindowHeight -490,0.0);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(600.0,gWindowHeight -490,0.0);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(600.0,gWindowHeight -540,0.0);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(0.0,gWindowHeight -540,0.0);
        glEnd();
            glFlush();                  //Finish rendering
            glutSwapBuffers();
}

void drawScene()
{

       if(flag==0)
          cover();
       if(flag==1)
```

```
    {
            glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(1.0, 1.0, 1.0, 1.0);
            glLoadIdentity();
            drawPlayerMove();
            drawInfo();
            drawGrid();
            drawGridMarks();
            glutSwapBuffers();
    }
}
void resize(int w, int h)
{
        gWindowHeight = h;
        gWindowWidth = w;
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0, w, 0, h, 0, 1);
        glMatrixMode(GL_MODELVIEW);
}

void keyInput(unsigned char key, int x, int y)
{
        switch(key)
 {
  case 13:     if(flag==0) //Ascii of 'enter' key is 13
              flag=1;
                break;
  case 27:      game.startNewGame();
                break;
  case '1':     playerOne = (playerOne + 1) % players.size();
              game.setPlayers(players[playerOne], players[playerTwo]);
                        cpuThinkTime = -CPU_THINK_INTERVAL;
```

```
                    break;

        case '2':       playerTwo = (playerTwo + 1) % players.size();
                    game.setPlayers(players[playerOne], players[playerTwo]);
                    cpuThinkTime = -CPU_THINK_INTERVAL;

 }
   drawScene();
}

void mouseInput(int button, int state, int x, int y)
{
        if (button != GLUT_LEFT_BUTTON || state != GLUT_DOWN)
                return;

        if (gPlayerMouseColumn >= 0 && game.isHumanPlaying())
                game.play(gPlayerMouseColumn);
}

void mouseMove(int x, int y)
{
        float boardWidth = (gWindowWidth - HORIZONTAL_MARGIN);
        x -= HORIZONTAL_MARGIN / 2.f;

        if (x < 0 || x > boardWidth)
                gPlayerMouseColumn = -1;
        else
        {
                gPlayerMouseColumn = (float)x / boardWidth * BOARD_WIDTH;

                if (gPlayerMouseColumn >= BOARD_WIDTH)
                        gPlayerMouseColumn = -1;
        }
}
```

```
void mouseEnter(int state)
{
	if (state == GLUT_LEFT)
		gPlayerMouseColumn = -1;
}

void timerFunction(int arg)
{

	glutTimerFunc(UPDATE_INTERVAL_MS, timerFunction, 0);

	if (game.isHumanPlaying())
		cpuPlaying = false;
	else
	{
		if (!cpuPlaying)
		{
			cpuPlaying = true;
			cpuThinkTime = 0.f;
		}
		else
			cpuThinkTime += UPDATE_INTERVAL_MS;
	}

	glutPostRedisplay();

	if (cpuPlaying && cpuThinkTime >= CPU_THINK_INTERVAL)
	{
		game.update(UPDATE_INTERVAL_MS / 1000);
		cpuThinkTime = 0.f;
	}
}
```

```
void drawGrid()
{
	float boardWidth = (gWindowWidth - HORIZONTAL_MARGIN);
	float boardHeight = (gWindowHeight - VERTICAL_MARGIN - TOP_MARGIN);

	glColor3f(0.f, 0.f, 0.f);
	glBegin(GL_LINES);

	for (unsigned i = 0; i <= BOARD_HEIGHT; i++)
	{
		// Horizontal Line
		glVertex3f(HORIZONTAL_MARGIN / 2.f, VERTICAL_MARGIN / 2.f +
(float)i / BOARD_HEIGHT * boardHeight, 0);
		glVertex3f(HORIZONTAL_MARGIN / 2.f + boardWidth,
VERTICAL_MARGIN / 2.f + (float)i / BOARD_HEIGHT * boardHeight, 0);
	}

	for (unsigned i = 0; i <= BOARD_WIDTH; i++)
	{
		// Vertical lines
		glVertex3f(HORIZONTAL_MARGIN / 2.f + (float)i / BOARD_WIDTH *
boardWidth, VERTICAL_MARGIN / 2.f, 0);
		glVertex3f(HORIZONTAL_MARGIN / 2.f + (float)i / BOARD_WIDTH *
boardWidth, VERTICAL_MARGIN / 2.f + boardHeight, 0);
	}

	glEnd();
}

void drawInfo()
{
	std::string text;

	if (game.hasTied())
```

```
        {
                text = "The game has tied.";
                glColor3f(0.f, 0.f, 0.f);
        }
        else if (game.getWinner() == 1)
        {
                text = "Player 1 won the game";
                glColor3f(1.f, 0.08f, 0.08f);
        }
        else if (game.getWinner() == -1)
        {
                text = "Player 2 won the game";
                glColor3f(0.08f, 0.08f, 1.f);
        }
        else if (game.getPlayerTurn() == 1)
        {
                text = "Its player 1 turn to move";
                glColor3f(1.f, 0.08f, 0.08f);
        }
        else if (game.getPlayerTurn() == -1)
        {
                text = "Its player 2 turn to move";
                glColor3f(0.08f, 0.08f, 1.f);
        }

        Draw::text(text, 10, gWindowHeight - 28);

        glColor3f(1.f, 0.08f, 0.08f);
        Draw::text("Player 1: " + game.getPlayerName(1)+ " (press '1' to change player)",
10, gWindowHeight - 88);
        glColor3f(0.08f, 0.08f, 1.f);
        Draw::text("Player 2: " + game.getPlayerName(2)+ " (press '2' to change player)",
10, gWindowHeight - 118);
}
```

```
void drawPlayerMove()
{
	float boardWidth = (gWindowWidth - HORIZONTAL_MARGIN);
	float boardHeight = (gWindowHeight - VERTICAL_MARGIN - TOP_MARGIN);
	float cellHalfWidth = boardWidth / BOARD_WIDTH / 2.f;
	float circleRadius = 18.f;

	if (gPlayerMouseColumn >= 0 && game.isHumanPlaying() && !game.hasWinner()
&& !game.hasTied())
	{
		if (game.getPlayerTurn() == 1)
			glColor3f(1.f, 0.08f, 0.08f);
		else if (game.getPlayerTurn() == -1)
			glColor3f(0.08f, 0.08f, 1.f);

		Draw::circle2d(HORIZONTAL_MARGIN / 2.f +
(float)gPlayerMouseColumn / BOARD_WIDTH * boardWidth + cellHalfWidth,
			boardHeight + VERTICAL_MARGIN / 2.f + circleRadius + 5.f,
circleRadius, 30);
	}
}

void drawGridMarks()
{
	float boardWidth = (gWindowWidth - HORIZONTAL_MARGIN);
	float boardHeight = (gWindowHeight - VERTICAL_MARGIN - TOP_MARGIN);
	float cellHalfHeight = boardHeight / BOARD_HEIGHT / 2.f;
	float cellHalfWidth = boardWidth / BOARD_WIDTH / 2.f;
	float radius =  MINVALUE(cellHalfHeight, cellHalfWidth) - 5.f;

	bool vic;
	int p;
```

```
	for (unsigned row = 0; row < BOARD_HEIGHT; row++)
	{
		for (unsigned col = 0; col < BOARD_WIDTH; col++)
		{
			vic = false;
			p = game.getPlayerFromPosition(BOARD_HEIGHT - row - 1, col,
&vic);

	if(vic)
	{
	  glColor3f(0.08f, 1.f, 0.08f);
	  Draw::circle2d(HORIZONTAL_MARGIN / 2.f + (float)col / BOARD_WIDTH *
boardWidth + cellHalfWidth,
				VERTICAL_MARGIN / 2.f + (float)row / BOARD_HEIGHT
* boardHeight + cellHalfHeight, radius, 30);
	}
			if (p == 1)
				glColor3f(1.f, 0.08f, 0.08f);
			else if (p == -1)
				glColor3f(0.08f, 0.08f, 1.f);
			else
				continue;

			Draw::circle2d(HORIZONTAL_MARGIN / 2.f + (float)col /
BOARD_WIDTH * boardWidth + cellHalfWidth,
				VERTICAL_MARGIN / 2.f + (float)row / BOARD_HEIGHT
* boardHeight + cellHalfHeight, vic ? radius - 11.f : radius, 30);

		}
	}
}
```

**/*Drawing.cpp*/**

```
#include "Drawing.h"
#include <cmath>
#include "Gl/glut.h"

void Draw::text(std::string text, int x, int y)
{
	glRasterPos3f(x, y, 0);

	for (unsigned i = 0; i < text.size(); i++)
		glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);
}

void Draw::circle2d(float x, float y, float radius, unsigned points)
{
	float angle = 0.f, delta = 2 * PI / points;
	glBegin(GL_TRIANGLE_FAN);
	glVertex3f(x, y, 0.f); glVertex3f(x + radius, y, 0.f);
	while (points-- >= 2)
	{
		angle += delta;
		glVertex3f(x + radius * std::cos(angle), y - radius * std::sin(angle), 0.f);
	}
	glVertex3f(x + radius, y, 0.f);
	glEnd();
}

void Draw::wirecircle2d(float x, float y, float radius, unsigned points)
{
	float angle = 0.f, delta = 2 * PI / (points - 1);
	glBegin(GL_LINE_STRIP);
	glVertex3f(x + radius, y, 0.f);
	while (points-- >= 2)
```

```
        {
                angle += delta;
                glVertex3f(x + radius * std::cos(angle), y - radius * std::sin(angle), 0.f);
        }
        glVertex3f(x + radius, y, 0.f);
        glEnd();   }
```

**/*Game.cpp*/**

```
#include "Game.h"
#include <cstdlib>
#include <cstdio>
#include <iostream>

Game::BoardSlot::BoardSlot() // One position of the board constructor
{
        owner_player = 0;
        victory = false;
}

Game::Game(unsigned width, unsigned height)
{
        board_width = width;
        board_height = height;
        player1 = player2 = 0;
}

std::string Game::getPlayerName(int i)
{
        if (i == 1)
                return player1 ? player1->getName() : "Human Player";
        else if (i == 2)
                return player2 ? player2->getName() : "Human Player";

        return "?";
```

```
}

unsigned Game::getBoardWidth()
{
        return board_width;
}

unsigned Game::getBoardHeight()
{
        return board_height;
}

void Game::startNewGame()
{
        board.clear();
        board.resize(board_height, std::vector<Game::BoardSlot>(board_width));
        player_turn = 1;
        winner = 0;
        tied = false;
        last_col_played = -1;
}

void Game::setPlayers(Player* p1, Player* p2)
{
        player1 = p1;
        player2 = p2;

        if (p1) p1->setId(1);
        if (p2) p2->setId(-1);
}

int Game::getPlayerTurn()
{
        return player_turn;
```

```
}

int Game::getWinner()
{
	return winner;
}

bool Game::hasWinner()
{
	return winner != 0;
}

bool Game::hasTied()
{
	return tied;
}

bool Game::isHumanPlaying()
{
	return (player_turn == 1 && player1 == NULL) ||
		(player_turn == -1 && player2 == NULL);
}

std::vector<Game::BoardSlot> Game::operator[](unsigned index)
{
	// If an invalid index is given, return an empty vector
	if (index >= board.size())
		return std::vector<BoardSlot>();

	return board[index];
}

int Game::getPlayerFromPosition(unsigned row, unsigned col, bool* isVictory)
{
```

```
        if (row >= board_height || col >= board_width)
                return 0;

        if (isVictory)
                *isVictory = board[row][col].victory;

        return board[row][col].owner_player;
}

int Game::checkWinner(unsigned row, unsigned col)
{
        int player = board[row][col].owner_player;

#define CHECKLINE(r, c) if (checkLine(player, row, col, r, c) >= 4) {markLine(player,
row, col, r, c); return player;}

        CHECKLINE(1, 0);
        CHECKLINE(0, 1);
        CHECKLINE(1, 1);
        CHECKLINE(1, -1);

        return 0;
}

bool Game::canPlay(int player, unsigned col)
{
        if (col >= board_width || winner != 0)
                return false;

        for (unsigned i = 0; i < board_height; i++)
                if (board[board_height - 1 - i][col].owner_player == 0)
                        return player == player_turn;

        return false;
```

```
}

void Game::play(unsigned col)
{
        if (hasWinner() || hasTied() || col >= board_width)
                return;

        for (unsigned i = 0; i < board_height; i++)
        {
                if (board[board_height - 1 - i][col].owner_player == 0)
                {
                        board[board_height - 1 - i][col].owner_player = player_turn;

                        if (checkWinner(board_height - 1 - i, col))
                                winner = player_turn;
                        else
                                checkTie();

                        break;
                }
        }

        setNextPlayerTurn();
        last_col_played = (int)col;
}

void Game::unplay(unsigned col)
{
   if (col >= board_width)
      return;

        winner = 0;
   tied = false;
   setNextPlayerTurn();
```

```
	for (unsigned i = 0; i < board_height; i++)
	{
		if (board[i][col].owner_player == player_turn)
		{
			board[i][col].owner_player = 0;
			return;
		}
	}

	setNextPlayerTurn();
}

unsigned Game::getRandomPlay()
{
	std::vector<unsigned> cols;

	for (unsigned i = 0; i < board_width; i++)
		if (canPlay(player_turn, i))
			cols.push_back(i);

	if (cols.empty())
		return 0;

	return cols[rand() % cols.size()];
}

void Game::update(float inteval)
{
	if (!isHumanPlaying())
	{
		unsigned nextMove = 0xFFFFFFFF;

		if (player_turn == 1)
```

```
                {
                        player1->setId(1);
                        nextMove = player1->getNextPlay(*this);
                }
                else if (player_turn == -1)
                {
                        player2->setId(-1);
                        nextMove = player2->getNextPlay(*this);
                }

                if (!canPlay(player_turn, nextMove))
                        nextMove = getRandomPlay();

                play(nextMove);
        }
}

void Game::print(std::string extra, std::string after)
{
        std::cout << extra;
        std::cout << " -------\n";
        for (unsigned row = 0; row < board_height; row++)
        {
                std::cout << "|";
                for (unsigned col = 0; col < board_width; col++)
                {
                        std::cout << (board[row][col].owner_player == 0 ? '0' :
(board[row][col].owner_player == 1 ? '1' : '2'));
                }
                std::cout << "|\n";
        }
        std::cout << " -------\n";
        std::cout << after;
}
```

```
void Game::export_as_file(std::string filepath, std::string mode, std::string extra)
{
	FILE* f = fopen(filepath.c_str(), mode.c_str());

	if (!f)
		return;

	fprintf(f, "%s\n", extra.c_str());

	for (unsigned row = 0; row < board_height; row++)
	{
		for (unsigned col = 0; col < board_width; col++)
		{
			fprintf(f, "%c", board[row][col].owner_player == 0 ? ' ' :
(board[row][col].owner_player == 1 ? '1' : '2'));
		}
		fprintf(f, "\n");
	}
	fprintf(f, "----------------------------\n");
	fclose(f);
}

void Game::import_from_file(std::string filepath)
{
	FILE* f = fopen(filepath.c_str(), "r");

	if (!f)
		return;

	startNewGame();
   char c;

	for (unsigned row = 0; row < board_height; row++)
```

```
        {
                for (unsigned col = 0; col < board_width; col++)
                {
                        fscanf(f, "%c", &c);

                        if (c == '2')
                                board[row][col].owner_player = -1;
                        else if (c == '1')
                                board[row][col].owner_player = 1;
                        else
                                board[row][col].owner_player = 0;
                }
                fscanf(f, "\n");
        }
        fclose(f);
}

void Game::import_from_str(std::string str)
{
        startNewGame();
        unsigned pos = 0;
        for (unsigned row = 0; row < board_height; row++)
        {
                for (unsigned col = 0; col < board_width; col++)
                {
                        if (str.size() <= pos)
                                return;

                        board[row][col].owner_player = str[pos++];

                        if (board[row][col].owner_player == '2')
                                board[row][col].owner_player = -1;
                        else if (board[row][col].owner_player == '1')
                                board[row][col].owner_player = 1;
```

```
                    else
                        board[row][col].owner_player = 0;
                }
        }
}

int Game::checkLine(int player, unsigned row, unsigned col, int x, int y)
{
        int connected = 1;

        for (unsigned i = 1; i < 4; i++)
        {
                if (getPlayerFromPosition(row + x * i, col + y * i) != player || connected >=
4)
                        break;
                else
                        connected++;
        }

        for (unsigned i = 1; i < 4; i++)
        {
                if (getPlayerFromPosition(row - x * i, col - y * i) != player || connected >= 4)
                        break;
                else
                        connected++;
        }

        return connected;
}

void Game::markLine(int player, unsigned row, unsigned col, int x, int y)
{
        if (!player)
                return;
```

```
        if (getPlayerFromPosition(row, col) == player)
                board[row][col].victory = true;

        for (unsigned i = 1; i < 4; i++)
        {
            if (getPlayerFromPosition(row + x * i, col + y * i) == player)
                board[row + x * i][col + y * i].victory = true;

            if (getPlayerFromPosition(row - x * i, col - y * i) == player)
                board[row - x * i][ col - y * i].victory = true;
        }
}

void Game::setNextPlayerTurn()
{
        if (player_turn == 0)
            player_turn = 1;
        else
            player_turn *= -1;
}

void Game::checkTie()
{
        if (hasWinner())
            return;

        bool full = true;

        for (unsigned i = 0; i < board_width; i++)
        {
            if (board[0][i].owner_player == 0)
            {
                full = false;
```

```
                        break;
                }
        }

        if (full)
                tied = true;
}
```

**/*RandomStrategy.cpp*/**

```
#include "RandomStrategy.h"
#include <climits>
#include <iostream>

unsigned RandomStrategy::getNextPlay(Game current)
{
        return current.getRandomPlay();
}

std::string RandomStrategy::getName()
{
        return "Random CPU";
}
```

**INCLUDE FILES:**

**/*Drawing.h*/**

```
#ifndef _DRAWING_H
#define _DRAWING_H

#include <string>

#define PI 3.1415f
```

```
namespace Draw
{
	void text(std::string text, int x, int y);
	void circle2d(float x, float y, float radius, unsigned points);
	void wirecircle2d(float x, float y, float radius, unsigned points);
};

#endif
```

**/*Game.h*/**

```
#ifndef _GAME_H
#define _GAME_H

class Game;

#include <vector>
#include <string>
#include "Player.h"

class Game
{
public:

	struct BoardSlot {
		BoardSlot();
		bool victory;
		int owner_player;
	};

	Game(unsigned width = 7, unsigned height = 6);
	std::string getPlayerName(int i);
	unsigned getBoardWidth();
```

```
        unsigned getBoardHeight();
        void startNewGame();
        void setPlayers(Player* p1 = 0, Player* p2 = 0);
        int getPlayerTurn();
        int getWinner();
        bool hasWinner();
        bool hasTied();
        bool isHumanPlaying();
        std::vector<BoardSlot> operator [] (unsigned index);
        int getPlayerFromPosition(unsigned row, unsigned col, bool* isVictory = 0);
        int checkWinner(unsigned row, unsigned col);
        bool canPlay(int player, unsigned col);
        void play(unsigned col);
        void unplay(unsigned col);
        unsigned getRandomPlay();

        void update(float inteval);
        void print(std::string extra = "", std::string after = "");
        void export_as_file(std::string filepath, std::string mode = "w", std::string extra ="");
        void import_from_file(std::string filepath);
        void import_from_str(std::string str);

        void setNextPlayerTurn();
private:

        int checkLine(int player, unsigned row, unsigned col, int x, int y);
        void markLine(int player, unsigned row, unsigned col, int x, int y);
        void checkTie();

        std::vector< std::vector<BoardSlot> > board;
        unsigned board_width, board_height;
        char last_col_played;
        int player_turn;
        int winner;
```

```
        bool tied;

        Player *player1, *player2;

};

#endif
```

**/*Player.h*/**

```
#ifndef _PLAYER_H
#define _PLAYER_H

class Player;

#include "Game.h"
#include <string>

class Player
{
public:
        virtual unsigned getNextPlay(Game current) = 0;
        virtual std::string getName() {return "?";}
        void setId(int id) {player_id = id;}
protected:
        int player_id;
};

#endif
```

**/*RandomStrategy.h*/**

```
#ifndef _RANDOM_ST_H
#define _RANDOM_ST_H
```

```
#include "Player.h"

class RandomStrategy : public Player
{
public:
        unsigned getNextPlay(Game current);
        std::string getName();
};

#endif
```

# 5. SAMPLE OUTPUT

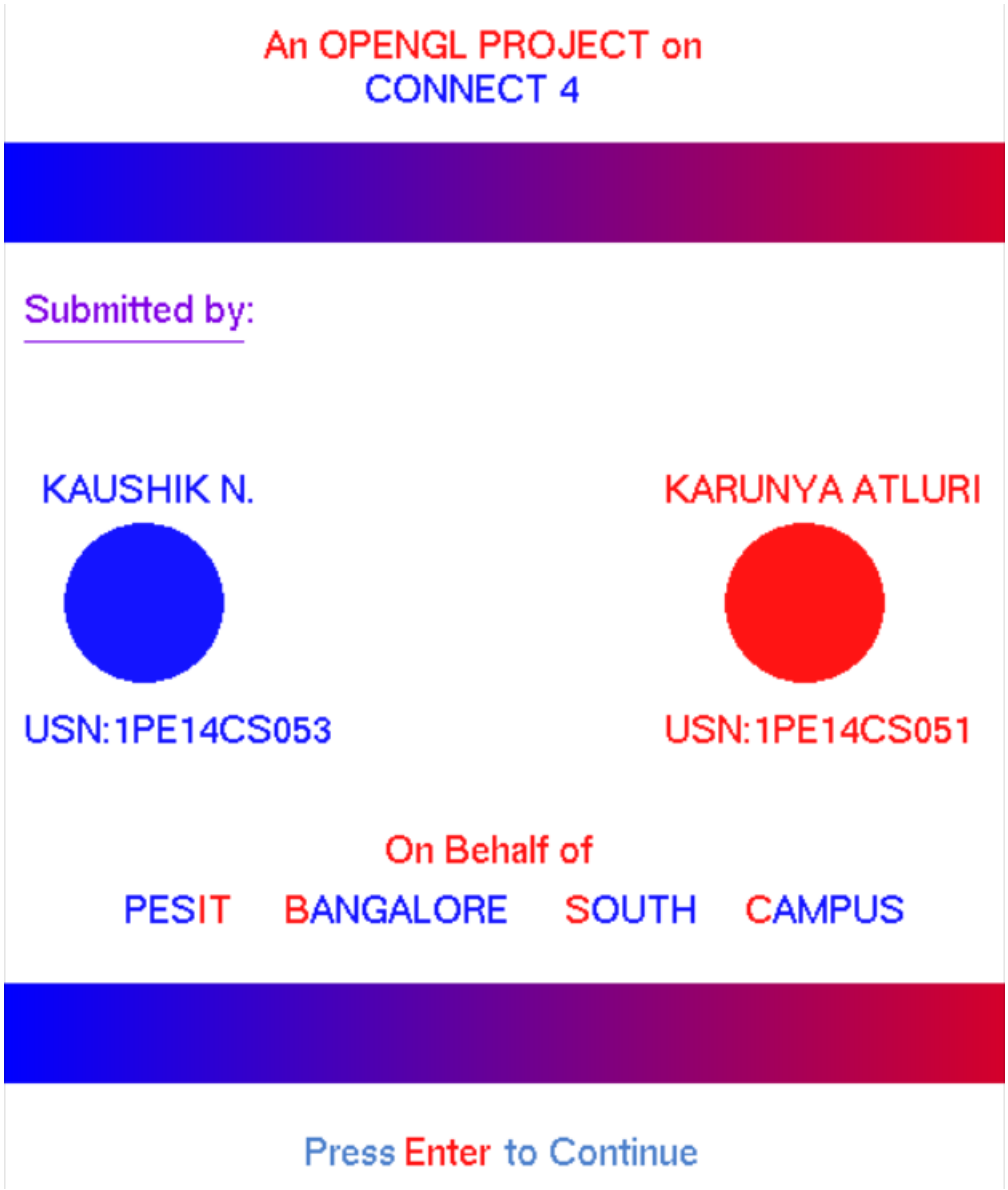


**Fig 5.1 : The Title page displaying the project details.**

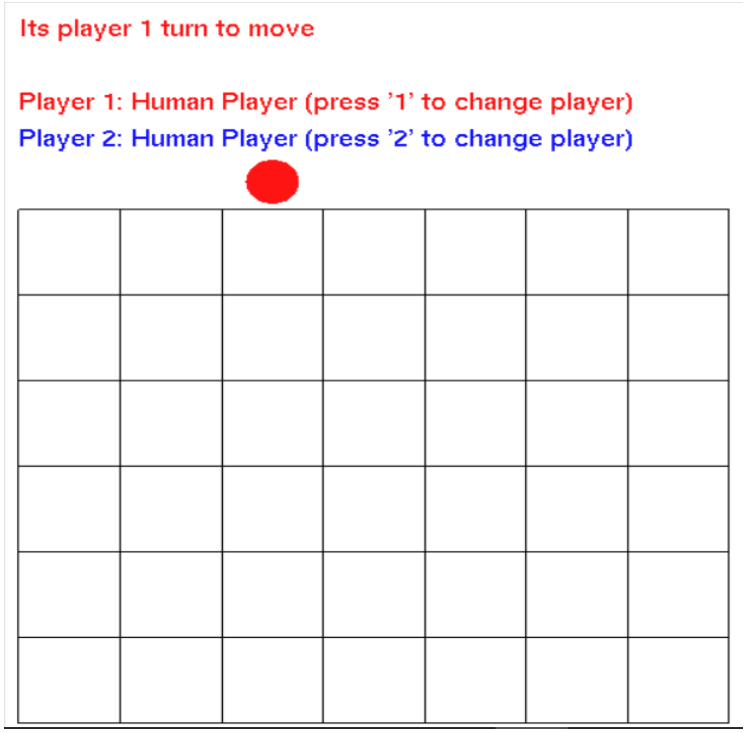


**Fig 5.2: In-game display where the Player 1 tries to make the first move**

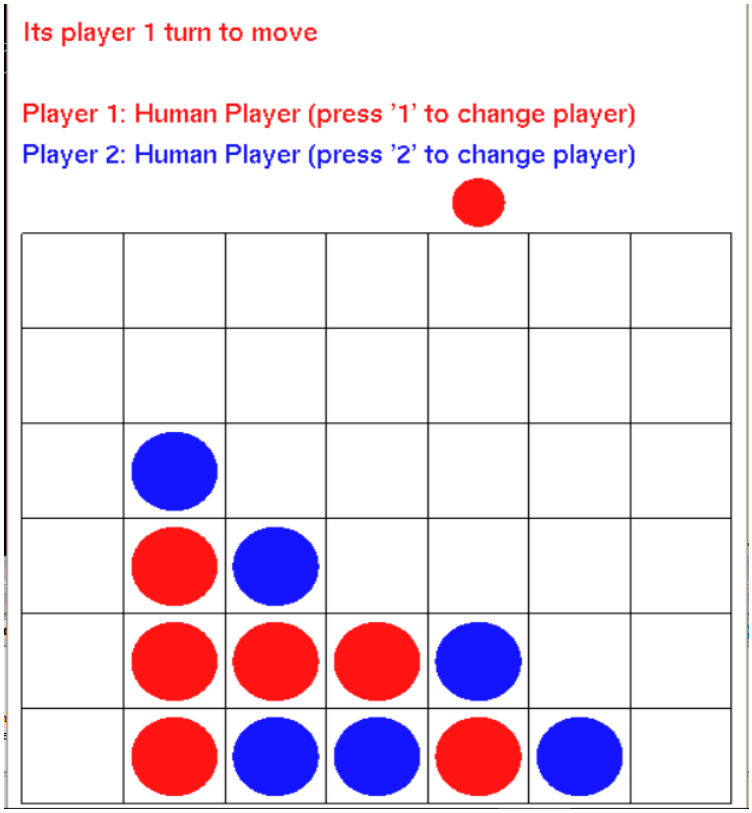


**Fig 5.3 : Showing a gameplay between two human players.**

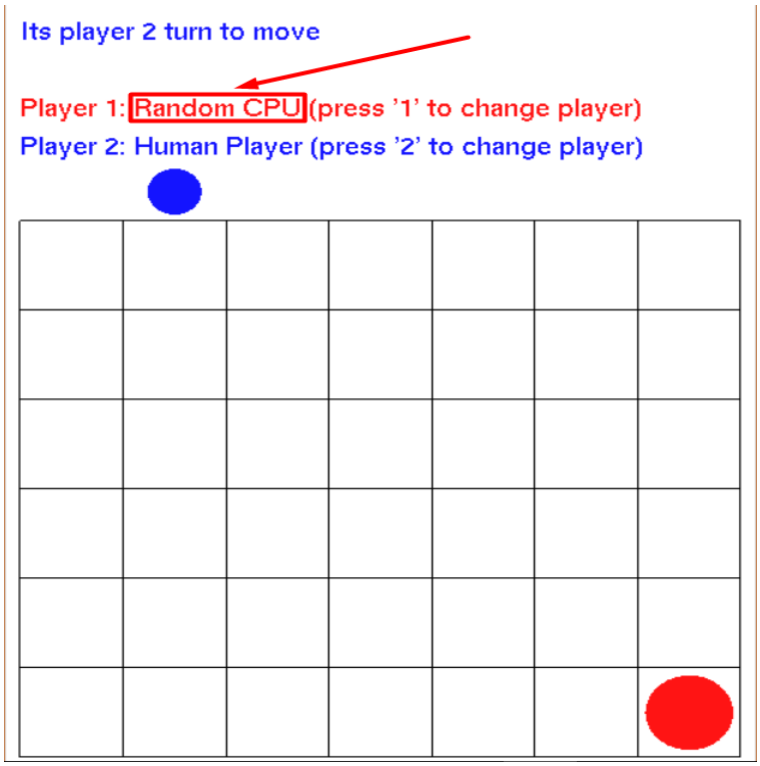


**Fig 5.4 : Showing a gameplay between a Human and CPU player.**


**Fig 5.5 : Outcome of match= Player 1 wins the game**

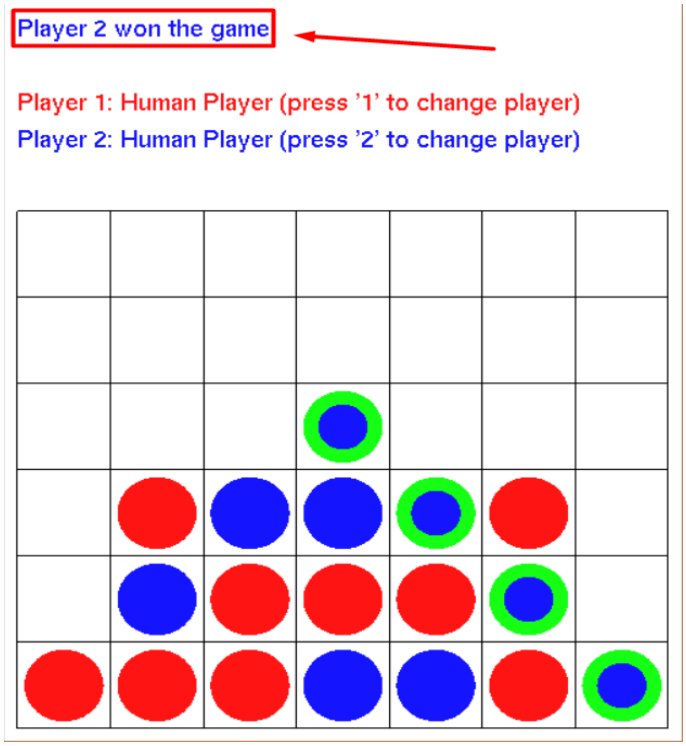


**Fig 5.6 : Outcome of match= Player 2 wins the game**

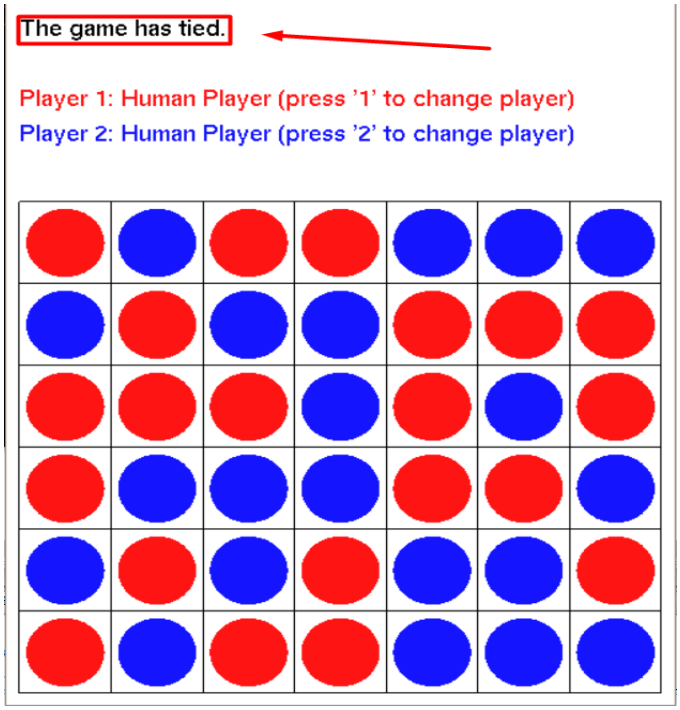


**Fig 5.7 : Outcome of match= A Tie Match**

# CONCLUSION

The goal of the project is to implement a fully-functioning game of Connect 4. The game will be played in a new graphics window with two players manually entering their moves. The program will check for a win or draw after every move.
This project serves as a simulation of the Connect-Four strategy game that benefits the players as follows:

- Builds skills such as **problem solving, basic math**
- Encourages players to **plan ahead** – looking out for opportunities to connect 4 discs
- Provides opportunity to **detect patterns**
- Instigates **prediction of the outcome of alternative moves** – for your own play as well as your opponent's. Keeping an eye on their moves keeps you one step ahead!
- **Learn from experience** – if you can ever get in a position to win by 2 options in one move, then you will never forget it!

A sound library called IrrKlang has been included in this project that automatically plays a wav ambient sound in a loopback manner in the background.

# BIBLIOGRAPHY

1. Edward Angel: Interactive Computer Graphics: A Top-Down Approach with OpenGL.

2. http://en.wikipedia.org/wiki/Connect_Four - A Wikipedia Entry on Connect-Four

3. https://www.opengl.org/resources/libraries/glut/ - A Repository of Glut Library files

4. http://www.pomakis.com/c4/ - A Masters Thesis on N-in a row game concept

5. http://nehe.gamedev.net/ - Reference articles and lessons on OpenGL