VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JnanaSangama, Belagavi-590018, Karnataka, INDIA

PROJECT REPORT
on

"Snake And Ladder Game "

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

A BHARATH REDDY	**1PE14CS001**
BHARGAV N	**1PE14CS031**

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 entitle **“Snake And Ladder Game”** is a bonafide work carried out by **BHARGAV N** and **A BHARATH REDDY** bearing USN **1PE14CS031** and **1PE14CS001*** 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, Belgaum during the year 2016-2017.*

Signature of the guide

Ksv Srikanth
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 **KSV Srikanth, Assosciate Professor and Bidisha Goswamy, 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.

A Bharath Reddy

Bhargav N

ABSTRACT

The user(s) need to press a key to roll a dice and dice gives a random number and the pawn goes to respective numbered tile .There are ladders and snakes visible on the board and the user will go back to the tile pointed to by the tail of the snake.If pawn is on the tile pointed to by the beginning of the ladder then it will reach to the top of the tile pointed to by the ladder.The main aim of the game is to reach the 100 th tile whoever reaches first is declared winner.The two players can visually see the movement of pawns across the tiles, ladders and snakes and sound effects are also added for enhanced view and gaming experience

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 *glutInit(&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 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.

3.5 glutPostWindowRedisplay

Description

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

3.6 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.

3.7 **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.

3.8 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).

3.9 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.

Font Rendering Functions.

3.10 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.

3.11 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

```
#include"snlv0.h"
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include<string.h>
#include<time.h>
#include<math.h>
using namespace std;
int check(int fg);
int p=1;
int m=1;
int ik=1,ch=0;
int q=0,lol=0;
int flg=1;
int vehicleX = x0+30, vehicleY = y2+30;
int vehicleX1 = x0+30, vehicleY1 = y2+30;
int vehicleY100,vehicleX100;
int vehicleY101,vehicleX101;
int getpos(int ji);

int divx = 250, divy = 4, movd;
int f=0;
void pawn();
void pawn2();
int drawsnakeandladder();
int cnt;
//Code to display  character string on the display
char mess[4];

void glutBitmapString(const char *str)
{
        int i=0;
        while(str[i]!='\0')
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,str[i++]);
}
int width,height;
/*---------------code for dice---------------*/

GLfloat vertices [][3] = {{1150.0,550.0,-50.0}, {1250.0,550.0,-50.0}, {1250.0,650.0,-50.0},
{1150.0,650.0,-50.0}, {1150.0,550.0,50.0}, {1250.0,550.0,50.0}, {1250.0,650.0,50.0},
{1150.0,650.0,50.0}};
GLfloat colors [][3] = {{0.0,0.0,0.0}, {1.0,-1.0,-1.0}, {1.0,1.0,0.0}, {0.0,1.0,0.0},
{0.0,0.0,1.0}, {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};
void polygon(int a,int b,int c,int d,int e)
{
        glBegin(GL_POLYGON);
                glColor3f(1.0,0.0,0.0);
        glVertex3fv(vertices[a]);
        glVertex3fv(vertices[b]);
        glVertex3fv(vertices[c]);
                glVertex3fv(vertices[d]);
        glEnd();
        glLineWidth(1.0);
        glBegin(GL_LINE_LOOP);
                glColor3f(0.9,0.0,0.0);
        glVertex3fv(vertices[a]);
        glVertex3fv(vertices[b]);
        glVertex3fv(vertices[c]);
                glVertex3fv(vertices[d]);
        glEnd();
        glLineWidth(2.0);
        glColor3f(1.0,1.0,1.0);
        glPointSize(3.0);
        glBegin(GL_POINTS);
    switch(e)
    {
        case 1 :
                glVertex3f(1200.0,600.0,-50.0);
                break;
        case 2 :
                glVertex3f(1175.0,651.0,25.0);
                glVertex3f(1225.0,651.0,-25.0);
                break;
        case 3 :    glVertex3f(1149.0,575.0,25.0);
                glVertex3f(1149.0,600.0,0.0);
                glVertex3f(1149.0,625.0,-25.0);
                break;
        case 4 :    glVertex3f(1251.0,575.0,25.0);
                glVertex3f(1251.0,575.0,-25.0);
                glVertex3f(1251.0,625.0,25.0);
                glVertex3f(1251.0,625.0,-25.0);
                break;
        case 5 :    glVertex3f(1175.0,549.0,25.0);
                glVertex3f(1175.0,549.0,-25.0);
                glVertex3f(1200.0,549.0,0.0);
                glVertex3f(1225.0,549.0,25.0);
                glVertex3f(1225.0,549.0,-25.0);
                break;
        case 6 :    glVertex3f(1175.0,575.0,51.0);
                glVertex3f(1175.0,600.0,51.0);
                glVertex3f(1175.0,625.0,51.0);
                glVertex3f(1225.0,575.0,51.0);
                glVertex3f(1225.0,600.0,51.0);
                glVertex3f(1225.0,625.0,51.0);
                break;
```

```
      }
         glEnd();
}
void colorcube(void)
{
   polygon(0,3,2,1,1);
   polygon(2,3,7,6,2);
   polygon(0,4,7,3,3);
   polygon(1,2,6,5,4);
   polygon(4,5,6,7,5);
   polygon(0,1,5,4,6);
}
static GLfloat theta[]={0.0,0.0,0.0};
/*--------------code for the board---------*/

void rect()
{
        int i,j;
        x[-1]=x0;
        y[-1]=y2;
        for( i=0;i<=maxx;i++)
                x[i]=x0+i*dx;
        for( j=0;j<=maxy;j++)
                y[j]=y2+j*dy;
        glColor3f(0.88,0.88,0.88);
        glBegin(GL_QUADS);
        glVertex3f(x0,y2,10);
        glVertex3f(x[maxx],y2,10);
        glVertex3f(x[maxx],y[maxy],10);
        glVertex3f(x0,y[maxy],10);
        glEnd();
        int count=1;
        int temp=1;
        i=0;
        for(j=0;j<maxy;j++)
        {
                while(1)
                {
                        if(temp<0)
                        {
                                if(i<0)
                                        break;
                        }
                        else
                        {
                                if(i>maxx-1)
                                        break;
                        }
                        glColor3f(0.4,0.2,0.0);
                        glBegin(GL_LINE_LOOP);
                        glVertex3f(x[i],y[j],11);
                        glVertex3f(x[i],y[j+1],11);
                        glVertex3f(x[i+temp],y[j+1],11);
                        glVertex3f(x[i+temp],y[j],11);
                        glEnd();
                        glColor3f(0.50/(i+j/2.0),1.0/(i+j),1.0/(i-j));
                        glRasterPos3f(x[i]+10,y[j]+10,20);
                        int k=0;
                        sprintf(mess,"%d",count);
                        if(count==100)
                                strcpy(mess,"END");
                        glutBitmapString(mess);
                        count++;
                        i+=temp;
                }
                if(temp>0)
                {
                        temp=-1;
                }
                else
                {
                        temp=1;
                }
                i+=temp;
        }
}
void display3()
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glClearColor(1.0,1.0,0.8,1.0);
        glColor3f(1.0,0.0,0.0);
   glRasterPos3f(400,700,10);
   glutBitmapString("PLAYER 2 HAS GAME OVER");
        glFlush();
   glutSwapBuffers();
}
void display4()
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glClearColor(1.0,1.0,0.8,1.0);
        glColor3f(1.0,0.0,0.0);
   glRasterPos3f(400,700,10);
   glutBitmapString("PALYER 1 HAS WON GAME OVER");
        glFlush();
   glutSwapBuffers();
}
/*--------------code for the starting page------------------*/
void display1()
{
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
   glClearColor(1.0,1.0,0.8,1.0);
   int i;
   glColor3f(1.0,0.0,0.0);
   glRasterPos3f(400,700,10);
   glutBitmapString("PES COLLEGE OF ENGINEERING");
   glRasterPos3f(370,650,10);
   glutBitmapString("DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING");
   glRasterPos3f(600,600,10);
   glutBitmapString("SNAKE 'N' LADDER");
   glRasterPos3f(200,400,10);
   glutBitmapString("USN : 1PE14CS031");
   glRasterPos3f(200,350,10);
   glutBitmapString("USN : 1PE14CS001");
   glRasterPos3f(850,400,10);
   glutBitmapString("NAME : BHARGAV N");
   glRasterPos3f(850,350,10);
   glutBitmapString("NAME : A BHARATH REDDY");
   glRasterPos3f(200,200,10);
   glutBitmapString("UNDER THE GUIDANCE OF : Mr.SRIKANTH AND
MS.BIDISHA");
   glColor3f(0.0,0.0,0.0);
   glBegin(GL_LINE_LOOP);
      glVertex3f(615.0,100.0,10.0);
      glVertex3f(615.0,140.0,10.0);
      glVertex3f(710.0,140.0,10.0);
      glVertex3f(710.0,100.0,10.0);
   glEnd();
   glColor3f(0.82,0.82,0.82);
   glBegin(GL_QUADS);
      glVertex3f(615.0,100.0,9.0);
      glVertex3f(615.0,140.0,9.0);
      glVertex3f(710.0,140.0,9.0);
      glVertex3f(710.0,100.0,9.0);
   glEnd();
   glColor3f(1.0,0.0,0.0);
   glRasterPos3f(630,110,10.0);
   glutBitmapString("NEXT");
   glFlush();
   glutSwapBuffers();
}
/*--------------Code for the gaming page-------------------*/
void display2()
{
         glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

         glClearColor(1.0,1.0,0.8,1.0);
         glPushMatrix();
                  glTranslatef(1200.0,600.0,0.0);
                  glRotatef(theta[0],1.0,1.0,0.0);
         glRotatef(theta[1],0.0,1.0,0.0);
                  glRotatef(theta[2],0.0,0.0,1.0);
         glTranslatef(-1200.0,-600.0,0.0);
                  colorcube();
         glPopMatrix();
         glFlush();
         rect();
         drawsnakeandladder();
         if(lol==0)
         {

                  pawn();

                  pawn2();

                  lol=1;
         }
         if(lol==1)
         {
                  pawn();

                  pawn2();

                  lol=0;

         }
         glColor3f(1.0,(128.0/255.0),0.0);
         int i;
         glRasterPos3f(50.0,600.0,20.0);
         glutBitmapString("PLAYER 1 : ");
         glRasterPos3f(50.0,550.0,20.0);
         glutBitmapString("PLAYER 2 : ");
         glColor3f(1.0,1.0,0.0);
         glBegin(GL_QUADS);
                  glVertex3f(200.0,600.0,20.0);
         glVertex3f(210.0,610.0,20.0);
         glVertex3f(200.0,620.0,20.0);
         glVertex3f(190.0,610.0,20.0);
         glEnd();
         glColor3f(0.5,1.0,0.0);
         glBegin(GL_QUADS);
         glVertex3f(200.0,550.0,20.0);
         glVertex3f(210.0,560.0,20.0);
         glVertex3f(200.0,570.0,20.0);
         glVertex3f(190.0,560.0,20.0);
         glEnd();
         glPushMatrix();
                  glTranslatef(1200.0,300.0,0.0);
                  //glColor3f((51.0/255.0),(51.0/255.0),1.0);
                  glColor3f(1,1,0);
```

```
                glBegin(GL_POLYGON);
                for(i=0;i<360;i++)
        {
                        float rad = i*(180/3.14);
                        glVertex3f(cos(rad)*60,sin(rad)*40,18.0);
        }
                glEnd();
        glPopMatrix();
        glColor3f(1.0,1.0,(201.0/255.0));
        glRasterPos3f(1163,306,20);
        glutBitmapString("CLICK");
        glRasterPos3f(1163,276,20);
        glutBitmapString(" HERE");
        glColor3f(0.4,0.2,0.0);
        glBegin(GL_QUADS);
     glVertex3f(x0+20,y[maxy]+20,0);
     glVertex3f(x0,y[maxy],10);
     glVertex3f(x[maxx],y[maxy],10);
     glVertex3f(x[maxx]+20,y[maxy]+20,0);
        glEnd();
        glBegin(GL_QUADS);
     glVertex3f(x[maxx],y[maxy],10);
     glVertex3f(x[maxx]+20,y[maxy]+20,0);
     glVertex3f(x[maxx]+20,y2+20,0);
     glVertex3f(x[maxx],y2,10);

        glEnd();


        glutSwapBuffers();
        //glutPostRedisplay();
}
void pawn(){
        //glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1,1,1);
        //drawRoad();
        //rawDivider();
  glPointSize(10.0);


   glBegin(GL_QUADS);
     glColor3f(1,0,0);//middle body
     glVertex3f(vehicleX100 - 20, vehicleY100 + 20,20);
     glVertex3f(vehicleX100 - 20, vehicleY100 - 20,20);
     glVertex3f(vehicleX100 + 20, vehicleY100 - 20,20);
     glVertex3f(vehicleX100 + 20, vehicleY100 + 20,20);
   glEnd();
```

```
                //vehicleX1=x0+30;
                ///vehicleY1=y2+30;






  // glFlush();

}
void pawn2(){
        //glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1,1,1);
        //drawRoad();
        //rawDivider();
  glPointSize(10.0);


   glBegin(GL_QUADS);
      glColor3f(1,1,0);//middle body
      glVertex3f(vehicleX101 - 20, vehicleY101 + 20,20);
      glVertex3f(vehicleX101 - 20, vehicleY101 - 20,20);
      glVertex3f(vehicleX101 + 20, vehicleY101 - 20,20);
      glVertex3f(vehicleX101 + 20, vehicleY101 + 20,20);
   glEnd();
   //vehicleX=x0+30;
                //vehicleY=y2+30;






  // glFlush();

}

int drawsnakeandladder()
{       int vehicleY1001,vehicleX1001,vehicleX1002,vehicleY1002;
        int arry[]={98,95,92,83,73,69,64,59,55,55,48,46,44,8,21,43,50,54,62,66,80};
        int arry2[]={28,24,51,19,1,33,36,17,7,11,9,5,22,26,82,77,91,93,96,87,100};
        for(int i=0;i<21;i++)
        {
                if(arry[i]<arry2[i])
                {
                        int kim,jhon;
                        int ko,jo,po;
                        jo=switchcase(arry[i]);
                        ko=(jo/10);
```

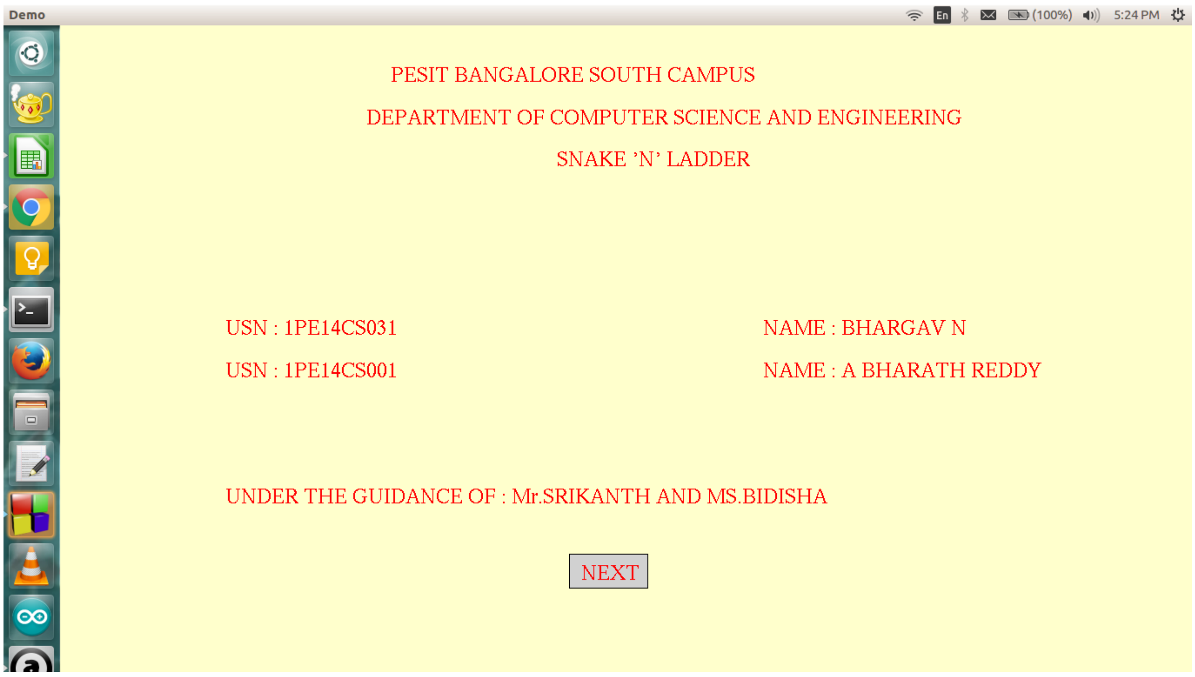
```
                    vehicleY1001=vehicleY+(ko*70);
                    po=(jo%10);
                    vehicleX1001=vehicleX+(po*70);
                    jo=switchcase(arry2[i]);
                    ko=(jo/10);

                    vehicleY1002=vehicleY+(ko*70);
                    po=(jo%10);
                    vehicleX1002=vehicleX+(po*70);
                    glColor3f(0.0,1.0,0.0);
                    glBegin(GL_LINES);
                    glVertex3f(vehicleX1001,vehicleY1001,20.0);
        glVertex3f(vehicleX1002,vehicleY1002,20.0);
        glEnd();




                }else
                {
                    int kim,jhon;
                    int ko,jo,po;
                    jo=switchcase(arry[i]);
                    ko=(jo/10);

                    vehicleY1001=vehicleY+(ko*70);
                    po=(jo%10);
                    vehicleX1001=vehicleX+(po*70);
                    jo=switchcase(arry2[i]);
                    ko=(jo/10);

                    vehicleY1002=vehicleY+(ko*70);
                    po=(jo%10);
                    vehicleX1002=vehicleX+(po*70);
                    glColor3f(1.0,0.0,0.0);
                    glBegin(GL_LINES);
                    glVertex3f(vehicleX1001,vehicleY1001,20.0);
        glVertex3f(vehicleX1002,vehicleY1002,20.0);
                    glEnd();

                }
        }
}
int getpos(int ji)
{
        int kim,jhon;
        int ko,jo,po;
        jo=switchcase(ji);
        ko=(jo/10);
```

```
            vehicleY100=vehicleY+(ko*70);
            po=(jo%10);
            vehicleX100=vehicleX+(po*70);
            //          vehicleX=vehicleX+70;


}
int getpos2(int ji)
{
            int ko,jo,po;
            jo=switchcase(ji);
            ko=(jo/10);
            vehicleY101=vehicleY1+(ko*70);
            po=(jo%10);
            vehicleX101=vehicleX1+(po*70);
            //          vehicleX=vehicleX+70;


}
int check(int fg)
{
    if(p==100)
    {
        cout<<"\n player 1 has won";
        cout<<"\ngame over";
                            glutDisplayFunc(display4);
            glutPostRedisplay();
        //exit(0);

    }
    if(m==100)
    {   cout<<"\nplayer 2 has won";
        cout<<"\ngame over";
        glutDisplayFunc(display3);
        glutPostRedisplay();
        //exit(0);

    }
    if(q==0)
    {q=1;
        int np=p;
        p=p+fg;
        switch(p)
        {
            case 98 :p=28;break;
            case 95 :p=24;break;
            case 92 :p=51;break;
            case 83 :p=19;break;
            case 73 :p=1;break;
            case 69 :p=33;break;
            case 64 :p=36;break;
            case 59 :p=17;break;
            case 55 :p=7;break;
            case 52 :p=11;break;
            case 48 :p=9;break;
            case 46 :p=5;break;
            case 44 :p=22;break;
            case 8 :p=26;break;
            case 21 :p=82;break;
            case 43 :p=77;break;
            case 50 :p=91;break;
            case 54 :p=93;break;
            case 62 :p=96;break;
            case 66 :p=87;break;
            case 80 :p=100;
        }
        if(p>100)
        {
            p=np;
        }
        cout<<"\n you have moved to player 1\t"<<p;
        getpos(p);
        return 0;
    }
    else
    {
        q=0;
        int nm=m;
        m=m+fg;
        switch(m)
        {
            case 98 :m=28;break;
            case 95 :m=24;break;
            case 92 :m=51;break;
            case 83 :m=19;break;
            case 73 :m=1;break;
            case 69 :m=33;break;
            case 64 :m=36;break;
            case 59 :m=17;break;
            case 55 :m=7;break;
            case 52 :m=11;break;
            case 48 :m=9;break;
            case 46 :m=5;break;
            case 44 :m=22;break;
            case 8 :m=26;break;
            case 21 :m=82;break;
            case 43 :m=77;break;
            case 50 :m=91;break;
            case 54 :m=93;break;
            case 62 :m=96;break;
            case 66 :m=87;break;
            case 80 :m=100;
        }
        if (m>100)
        {
            m=nm;
        }
        cout<<"\n you have moved to player 2\t"<<m;
        getpos2(m);
        return 0;
    }

}

/*------------Code for spinning the cube----------------*/
int spincube()
{

            theta[0]+=12;
    if(theta[0]>360.0)
            {
                        cnt++;
        theta[0]-=360;
            }
    display2();
            if(cnt<2)
        spincube();
    else
    {
        if(flg)
                        {

            srand(time(NULL));
           //vehicleX=vehicleX+70;
           //getpos(20);

           //for(ik=1;ik<100;ik++)
           //{
           //getpos(70);
                                //}
                                int s=(rand()%6)+1;
                                if(ch==0)
                                {
                                            check(s);
                                            ch==1;

                                }
                                if(ch==1)
                                {
                                        check(s);
                                        ch==0;
                                }

        switch(s)
        {
           case 1 :    theta[1]+=90;
                       theta[2]-=45;
                       break;
           case 2 :    theta[2]+=45;
                       break;
           case 3 :    theta[2]-=45;
                       break;
           case 4 :    theta[2]+=135;
                       break;
           case 5 :    theta[2]+=225;
                       break;
           case 6 :    theta[1]-=90;
                       theta[2]-=45;
                       break;
                                }
                                flg=0;
                                spincube();
                        }
        }
        return 0;
}
/*-------------- code for mouse call back --------------------*/
void mouse(int btn,int state,int x,int y)
{
        if(!f)
        {
                if(x>=((610*width)/1367) && x<=((710*width)/1367) && y<=(((767-
100)*height)/767) && y>=(((767-140)*height)/767))
        {
                                if(btn==GLUT_LEFT_BUTTON&&state==GLUT_DOWN)
                                {
                                glutDisplayFunc(display2);
                                glutPostRedisplay();
                                f=1;
                                }
        }
        }
        else
        {
                if(x>=((1140*width)/1367) && x<=((1260*width)/1367) && y<=(((767-
260)*height)/767) && y>=(((767-340)*height)/767))
                {
                                if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
                {
```

```
                              cnt=0;flg=1;theta[1]=90;theta[2]=45;
                              spincube();
                              //getpos(20);
                              system("audacious rename.mp3 ");


                    }
                    }
          }
}

void myReshape(int w,int h)
{
      width=w;height=h;
         glViewport(0,0,w,h);
         glMatrixMode(GL_PROJECTION);
         glLoadIdentity();
      glOrtho(0.0,1367.0,0.0,767.0, -180.0,180.0);
      glMatrixMode(GL_MODELVIEW);
      glClearColor(1.0,1.0,0.8,1.0);
}
int main(int argc, char **argv)
{
         glutInit(&argc,argv);
         glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB| GLUT_DEPTH);
         glutInitWindowPosition(0,0);
         glutInitWindowSize(1366,768);
         glutCreateWindow("Demo");
         glEnable(GL_DEPTH_TEST);

         //glDepthFunc(GL_NEVER);
         glutDisplayFunc(display1);
         glutMouseFunc(mouse);
         glutReshapeFunc(myReshape);
         glutMainLoop();
}
```

5. SAMPLE OUTPUT

Fig 5.1 The title page displaying the project details.



PESIT BANGALORE SOUTH CAMPUS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SNAKE 'N' LADDER

USN : 1PE14CS031 NAME : BHARGAV N

USN : 1PE14CS001 NAME : A BHARATH REDDY

UNDER THE GUIDANCE OF : Mr.SRIKANTH AND MS.BIDISHA

NEXT

Fig 5.2 The board of the game along with Snakes and Ladders

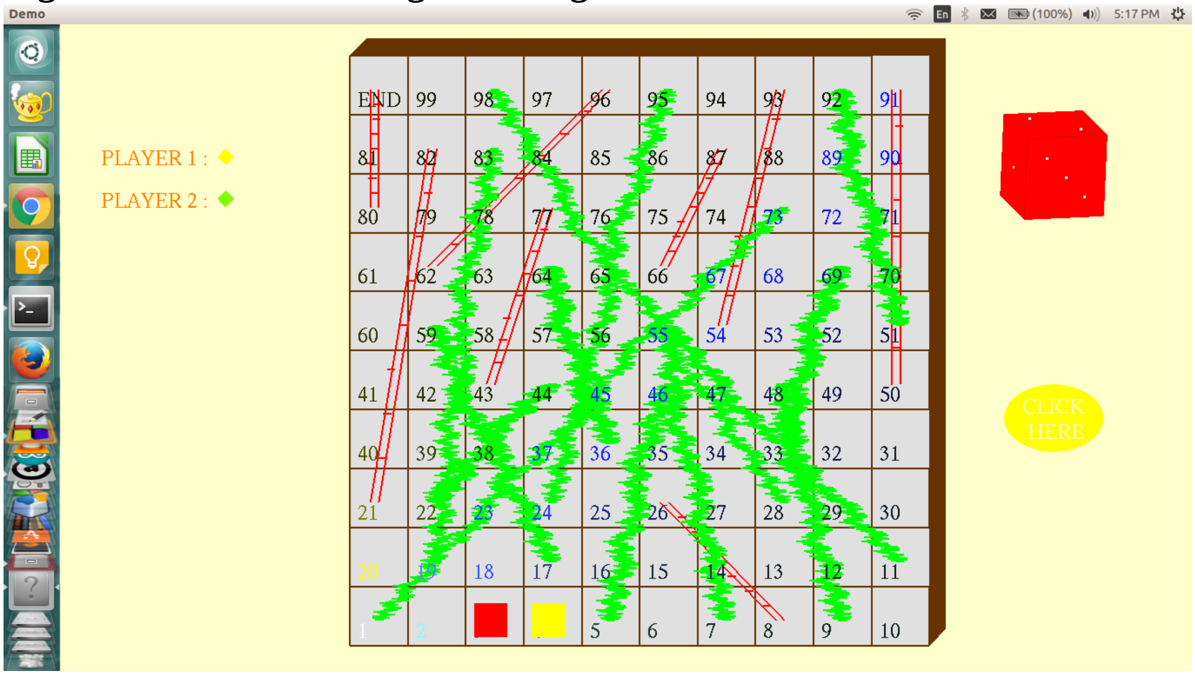
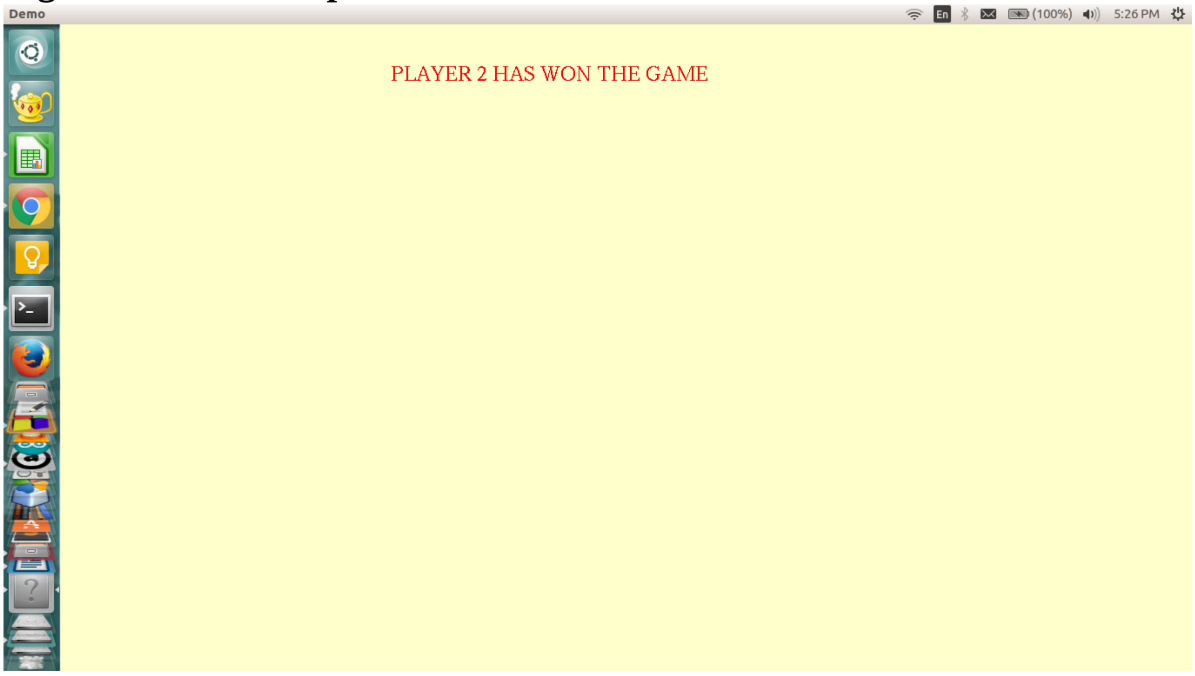


Fig 5.3 The final output of the Game :-



```
PLAYER 2 HAS WON THE GAME
```

CONCLUSION

This game lets both the users play their game alternately by pressing their keys .we have mainly demonstrates the use of OpenGl library in building simulation of movement of pawns , dice and construction of the board, ladders and snakes It is an interactive program that takes keyboard input from the user and accordingly display the simulation sequentially.

The basic minimal features such as built-in functions and apis have been included in this graphics program.

A sound library called AUDACIOUS have been included in this project that automatically plays ambient sounds in a loopback manner in the background whenever there is motion of pawn and whenever it gets on a ladder or snake.We have given a different sound when the user finally wins and whenever the dice is rolled.

This project serves as a better understanding of the graphics and OPENGL along with the game.

We also are looking forward to connect two different laptops and let them play the game simultaneously.

BIBLIOGRAPHY

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

2. https://en.wikipedia.org/wiki/Snake_Andladder_gamecode

3. https://www.opengl.org/resources/libraries/glut/

4.. http://micro.magnet.fsu.edu/primer/java/games/Snakaandladdergame

5.https://stackoverflow.com/questions/4171784/how-to-draw-a-snake-in-opengl