Display2( ):

- It enters the Level 2 of the game. It initially sets the the red ball to a position given by ‘up’ variable, and using the Translatef() function it displaces only the y-coordinate of the ball upwards.
- Also, it sets the blue balls position and using the Translatef() function it displaces only the y-coordinate of the ball downwards.

- glutSolidSphere() renders a 3D sphere with radius, slices and stack as parameters.

- While the ‘f’ key is pressed, the function creates an arrow head, and associates a variable ‘pos’ with it, to translate the arrow towards the right in a single direction. This variable is incremented continually everytime and called with the Translatef() to redraw the arrow at new positions**.**

- If the condition for bounds satisfy, that means collision has occurred, and counter1 is incremented to register a hit. The flag ‘bang’ is set to 1, so that when encountered during the next iteration the following changes can take place: position of sphere is reset to 0, bang is reset to 0 (to prepare for next hit), position of arrow is reset.

- ‘up’ variable is continually incremented to keep the ball moving upwards for a large number of iterations by calling glutPostRedisplay() everytime. It marks the normal plane of current window as needing to be redisplayed with the same specifications. The counter2 value is checked for every iteration. Once it has reached a value of 3, myHit() is called to display that the player is a winner**.**

  MyHit( ):

  This function is primarily to display text on screen. glBlendFunc defines the operation of blending when it is enabled. We also set the antialiasing for lines and set the line width to prepare to draw our text as Stroke Characters.

  Drawhit( ):

  It draws text in Stroke Character font. A stroke font is a 3D font. As opposed to bitmap fonts these can be rotated, scaled, and translated.
  The GLUT_STROKE_ROMAN font is used here. The text can be placed anywhere on the screen using Translatef() and scaled to any size as needed using Scalef().

Instructions( ):

It creates a separate instruction page where the instructions are displayed on another window using StrokeCharacter font, and translated to appropriate postions on the screen.

Draw instruct( ):

It draws text in Stroke Character font. A stroke font is a 3D font. As opposed to bitmap fonts these can be rotated, scaled, and translated. The GLUT_STROKE_ROMAN font is used here. The text can be placed anywhere on the screen using Translatef() and scaled to any size as needed using Scalef().

Keyboard( ):

Keyboard is a keyboard callback function which is used to make our program interactive. It makes the shoot flag = 1 in our program everytime key 'f' is pressed, by recognizing it as an ASCII character.

Main():

The main function performs the required initializations and starts the event processing loop. All the functions in GLUT have the prefix glut, and those which perform some kind of initialization have the prefix glutInit.

- glutInit(int *argc, char **argv) - parameters are pointers to the unmodified argc and argv variables from the main function**.**

- We establish the window's position by using the function glutInitWindowPosition**.**

- We choose the window size using the function glutInitWindowSize.

- We define the display mode using the function glutInitDisplayMode. GLUT_RGB - selects a RGBA window. This is the default color mode. GLUT_SINGLE – selects a single buffer window.

- Each window can be created with glutCreateWindow. The return value of glutCreateWindow is the window identifier**.**

- glutDisplayFunc() passes the name of the function to be called when the window needs to be redrawn.

- glutKeyboardFunc- is notify the windows system which function(s) will perform the required processing when a key is pressed. This function is to register a callback for keyboard events that occur when you press a key.

- Creating a menu: glutCreateMenu creates a menu table on a default right click of mouse. glutAddMenuEntry adds a menu entry to this menu created.

- When we are ready to get in the application event processing loop we enter glutMainLoop. It gets the application in a never ending loop, always waiting for the next event to process.

Init( ):

Sets the background color for the game and enables light source to provide the following lighting effects:

- AMBIENT - light that comes from all directions equally and is scattered in all directions equally by the polygon**.**
- DIFFUSE - light that comes from a point source and hits surfaces with an intensity that depends on whether they face towards the light or away from it.
- SPECULAR - light is reflected more in the manner of a mirror where most of the light bounces off in a particular direction defined by the surface shape.
- EMISSION - the light is actually emitted by the polygon - equally in all directions It also sets the Material for the object and enables this option to provide different surface textures to our objects.

4.2 User implementation

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<GL/glut.h>
#include<time.h>
#include<math.h>
//#include<windows.h>
static GLfloat up=-0.2;
static GLfloat pos=-0.2;
int shoot=0,bang=0;
int counter1=0,counter2=0,count=0;
int game,instruct;
char tmp_str[40];
void display2();
void displost();
void init(void)
{
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shininess[] = { 50.0 };
GLfloat mat_diffuse[]={ 1.0,1.0,1.0,1.0};
GLfloat mat_ambient[]={0.0,0.0,0.0,1.0};
GLfloat light_position[] = { 1.0, 1.0, 0.0, 0.0 };
```

```
glClearColor (0.0, 0.0, 0.0, 0.0);
glShadeModel (GL_SMOOTH);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_DEPTH_TEST);
glEnable(GL_COLOR_MATERIAL);
}
void drawhit(const char * message, int x, int y)
{
glPushMatrix();
glScalef(0.3,0.2,0.15);
glTranslatef(x,y,0);
while (*message)
{
glutStrokeCharacter(GLUT_STROKE_ROMAN,*message++);
}


glPopMatrix();
}

void myHit()
{
glClear(GL_COLOR_BUFFER_BIT);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,200,0,200);
glMatrixMode(GL_MODELVIEW);
glClearColor(1.0,0.0,0.5,1.0);
glColor3f(0.0,0.8,0.80);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_BLEND);
glEnable(GL_LINE_SMOOTH);
glLineWidth(4.0);
drawhit("WINNER!!",70,550);
}
void draw_instruct(const char *message, int x, int y)
{
int j;
glPushMatrix();
glScalef(0.1,0.1,0.0);
glTranslatef(x,y,0);
while (*message)
{
glutStrokeCharacter(GLUT_STROKE_ROMAN,*message++);
```

```
}
for(j=0;j<10000;j++);
glPopMatrix();
}
void instructions()
{
glClear(GL_COLOR_BUFFER_BIT);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,200,0,200);
glMatrixMode(GL_MODELVIEW);
glClearColor(1.0,0.7,0.0,1.0);
glColor3f(1.0,0.5,0.1);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_BLEND);
glEnable(GL_LINE_SMOOTH);
glLineWidth(4.0);
draw_instruct("Instructions",600,1850); // change1
draw_instruct("Right click on mouse",300,1700);
draw_instruct("to play",300,1500);
draw_instruct("Press f to shoot",300,1300);


glFlush();
}

void Write(char *string)
{
glScalef(0.02,0.02,0.0);
while(*string)
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *string++);
}
void display1()
{
int i;
if(counter1==3)
{
display2();
glFlush();
}
else
{ int j;
for(j=0;j<10000;j++);
glClearColor(1.0,0.7,0.0,1.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glColor3f(1, 1, 0);
glRasterPos2f(-0.9, 0.9);
sprintf(tmp_str, "Arrow count: %d", count);
Write(tmp_str);
glPopMatrix();
```

```
if(count>=30)
glutDisplayFunc(displost);
glPushMatrix();
glColor3f(1, 1, 0);
glRasterPos2f(-0.2, 0.9);
sprintf(tmp_str, "Score: %d", counter1);
Write(tmp_str);
glPopMatrix();
glPushMatrix();
glColor3f(1.0,0.0,0.0);
glLoadIdentity();
glTranslatef(0.8,-0.869+up,0.0);
glutSolidSphere(0.15,20,16);
if(shoot==1)
{
glPushMatrix();
glLoadIdentity();
glTranslatef(-0.8+pos,0.0,0.0);
glColor3f(0.0,0.0,0.0);
glLineWidth(2.0);


glBegin(GL_LINES);
glVertex3f(-0.2,0.0,0.0);
glVertex3f(0.1,0.0,0.0);
glVertex3f(0.1,0.0,0.0);
glVertex3f(0.03,0.05,0.0);
glVertex3f(0.1,0.0,0.0);
glVertex3f(0.03,-0.05,0.0);
glEnd();
glPopMatrix();
}
if(bang==1)
{
bang=0;pos=-0.2;
glPushMatrix();
glLoadIdentity();
up=0;
glColor3f(1.0,0.0,0.0);
glutSolidSphere(1,20,16);
glPopMatrix();
}
glPopMatrix();
for( i=0;i<200;i=i+20)
{
if(pos>=1.74 && up>0.825 && up<0.975)
//collision detection
{
counter1 ++;
for(j=0;j<10000;j++);
shoot=0;
pos=-0.2;
bang=1;
}
if(counter1==3)
count=0;
up=(up+0.005);
if(up>2)
up=0;
if(shoot==1)
{
pos=pos+0.009;
if(pos>2)
{
pos=-0.2;
shoot=0;
}
}
glutPostRedisplay();
}


glFlush();
}
}
void display2()
{int i;
if(counter2==3)
{ myHit();
glFlush();
}
else
{int j;
for(j=0;j<10000;j++);
glClearColor(1.0,0.7,0.0,1.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glPushMatrix();
glColor3f(1, 1, 0);
glRasterPos2f(-0.9, 0.9);
sprintf(tmp_str, "Arrow count: %d", count);
Write(tmp_str);
glPopMatrix();
if(count>=20)
glutDisplayFunc(displost);
glPushMatrix();
glColor3f(1, 1, 0);
glRasterPos2f(-0.2, 0.9);
sprintf(tmp_str, "Score: %d", counter2);
Write(tmp_str);
glPopMatrix();
glPushMatrix();
```

```
glColor3f(1.0,0.0,0.0);
glLoadIdentity();
glTranslatef(0.8,-0.769+up,0.0);
glutSolidSphere(0.10,20,16);
glColor3f(0.0,0.0,1.0);
glPushMatrix();
glColor3f(0.0,0.0,1.0);
glLoadIdentity();
glTranslatef(0.4,0.769-up,0.0);
glutSolidSphere(0.10,20,16);
glColor3f(0.0,0.0,1.0);
if(shoot==1)
{
glPushMatrix();
glLoadIdentity();
glTranslatef(-0.8+pos,0.0,0.0);
glColor3f(0.0,0.0,0.0);
glLineWidth(2.0);


glBegin(GL_LINES);
glVertex3f(-0.2,0.0,0.0);
glVertex3f(0.1,0.0,0.0);
glVertex3f(0.1,0.0,0.0);
glVertex3f(0.03,0.05,0.0);
glVertex3f(0.1,0.0,0.0);
glVertex3f(0.03,-0.05,0.0);
glEnd();
glPopMatrix();
}
if(bang==1)
{
bang=0;pos=-0.2;
glPushMatrix();
glLoadIdentity();
up=0;
glColor3f(1.0,0.0,0.0);
glutSolidSphere(1,20,16);
glPopMatrix();
}
glPopMatrix();
for( i=0;i<200;i=i+20)
{
if(pos>=1.75 && up>0.825 && up<0.975)
{
counter2 ++;
for(j=0;j<10000;j++);
shoot=0;
pos=-0.2;
bang=1;
}
up=(up+0.005);
if(up>2)
up=0;
if(shoot==1)
{
pos=pos+0.009;
if(pos>2)
{
pos=-0.2;
shoot=0;
}
}
glutPostRedisplay();
}
}




glFlush();
}
void display()
{
glClearColor(1.0,0.7,0.0,1.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glFlush();
}
void displost()
{
glClear(GL_COLOR_BUFFER_BIT);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,200,0,200);
glMatrixMode(GL_MODELVIEW);
glClearColor(1.0,0.0,0.5,1.0);
glColor3f(0.0,0.8,0.80);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_BLEND);
glEnable(GL_LINE_SMOOTH);
glLineWidth(4.0);
drawhit("you lost!!",70,550);
glFlush();
}
void indisplay()
{
glClearColor(1.0,0.7,0.0,1.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
instructions();
glFlush();
}
void keyboard(unsigned char key,int x,int y)
```

```
{
if (key=='f')
{
shoot=1;
count++;
}
}
void choose(int i)
{
switch(i)
{ case 1: exit(0);
case 2: glutDisplayFunc(display1);
break;
case 3: glutDisplayFunc(display2);
break;



default:exit(0);
}
}
int main(int argc,char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DEPTH|GLUT_RGB);
glutInitWindowSize(1500,1500);
glutInitWindowPosition(0,0);
instruct=glutCreateWindow("Instructions");
init();
glutDisplayFunc(indisplay);
glutInitDisplayMode(GLUT_DEPTH|GLUT_RGB);
glutInitWindowSize(1000,1000);
glutInitWindowPosition(0,0);
game=glutCreateWindow("Proj");
init();
glutDisplayFunc(display);
glutKeyboardFunc(keyboard);
glutCreateMenu(choose);
glutAddMenuEntry("Quit",1);
glutAddMenuEntry("PlayLevel1",2);
glutAddMenuEntry("PlayLevel2",3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop();
return 0;
}
```