Vivekanand Education Society's Institute of Technology

Department of Computer Engineering

2019-20

Subject: Computer Graphics Class:D7A/BATCH-B Semester: IV

Roll No: 27 46	Name: Harish Kumar Shikhar Niranjan		
	Title: Mini Project		
DOP:	04/04/20	DOS:	10/04/20
GRADE	LAB OUTCOMES:	SIGNATURE:	

INTRODUCTION

1.1 Introduction to the graphics:

Computer graphics is one of the most exciting and rapidly growing computer field and computer. It is also an extremely effective medium for communication between men.

The human can understand the information content of a displayed diagram or perceptive view much faster than it can understand a table of numbers.

There is a lot of development in hardware and software required to generate images, and now-a-days the cost of such hardware and software is also dropping rapidly. Due to this the interactive computer graphics is becoming available to more and more people.

Computer graphics today is largely interactive. The user controls the contents, structure and appearance of the objects and of their displayed images by using input devices, such as keyboard, mouse or touch sensitive panel on the screen. Because of the close relationship between the input devices and display, the handling of such devices is included in the study of computer graphics.

1.2 Uses of computer graphics:

User interface:

t is now a well-established fact that graphical interfaces provide an alternative and easy interaction between users and computers the built in graphics provided with user interfaces use the control items.

n industry, business government and education organization's computer graphics is most commonly used to create 2D and 3D graphs of mathematical, physical and economic functions in the form of histograms, bars and pie charts which are very useful in decision making.

Computer aided drafting and design:

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

Simulation and animation for scientific visualization and environment:

Use of graphics in simulation makes mathematical models and mechanical systems more realistic and easy to study. The interactive graphics supported by animation software proved their use in production of animated movies and cartoon films.

1.3 OpenGL

penGL (open graphic library) is a standard specification defining a cross language cross platform API for writing application that produces 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex 3D scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization and flight simulation. It is also used in video games, where it competes with direct 3D on Microsoft Windows Platforms, OpenGL is managed by the nonprofit technology consortium, the Khronos group Inc.

OpenGL serves two main purposes:

- To hide the complexities of interfacing with different 3D accelerators, by presenting programmer with a single. Uniform API
- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set.

1.4 About archery Game

Archery game is a precision sport where the competitors aim and shoot at the target using the arrow.

Every arrow is made of 3 parts:

Tip

Shaft

Nock

Tip of the arrow is used to hit the target. Tip is drawn using a triangle. Shaft of arrow is drawn using lines. Nock is drawn using quads.

The target used is a block which is drawn using point and enclosing the point within lone loop to create boundary for the target such that only the point hit by the player disappears the target. Disappearing is highlighted by drawing the target hit by the player using white color which appears as a hole. This game needs good vision and good concentration.

LITERATURE SURVEY

2.1 WWW.OPENGL.ORG:

OpenGL 3.0 adds many features to the core of OpenGL. It also brings with it a deprecation model that previous versions of OpenGL did not have. Before OpenGL 3.0, anything that became core had to remain in the specification permanently. The deprecation model allows OpenGL versions to announce that certain features may be removed from the core in later versions.

The OpenGL specification now is broken into two specifications: core and compatibility. Compatibility provides full backwards compatibility with GL 2.1 and below, while Core does not. A new context creation model exists; it is the only way to create core contexts of OpenGL 3.1 and above.

Part of this new API is a specification of exactly what version of OpenGL you want. So if you ask for a GL 3.1 context, you are telling the system that you expect that any entrypoints version 3.1 removed from earlier versions will not be available, and that any entrypoints 3.1 added to new versions will be available. The new API can fail if the implementation simply does not implement that version of OpenGL.

More detailed instructions for Creating an OpenGL Context are available.

OpenGL specifications:

OpenGL 4.1 Core Profile Specification
OpenGL Shading Language 4.1 Specification
OpenGL 3.3 Core Profile Specification
OpenGL Shading Language 3.3 Specification

How to make your first OpenGL Program:

The first thing to do is chose a programming language. It could be C, C++, C#, Visual Basic, Pascal, Perl, Java, Ada, x86 assembly, etc. As long as a language has an OpenGL binding for your chosen language, you may use it.

The second thing is to choose a compiler. It could be MS Visual C++, Code::Blocks, Delphi, Masm, etc. Remember that OpenGL is an API, so as long as you have the language bindings for your compiler, you can do OpenGL programming.

Typically, a compiler comes with the binding files. For example, if you have a C++ compiler, it will come with gl.h and opengl32.lib. It may even come with glu.h and glu32.lib, glut.h and glut32.lib.

If you don't have your binding files, you will need to figure out where to download them from. Microsoft releases their Windows Platform SDK which contains these files and most likely you don't need it because your compiler came with the files.

You might want to use SDL, GLUT, freeGLUT, or some other wrapper that takes care of creating a GL window for you and destroying for you. It makes it easier for someone who just wants to learn the OpenGL API syntax.

Assuming you know how to program in your language of choice, now all you need it to learn OpenGL. There are many online tutorials. Just search for opengl+tutorial in your favorite search engine or visit some of the tutorials listed here.

OpenGL Viewers:

These are programs that you install and run, and they give you information specific to the OpenGL API your system implements, like the version offered by your system, the vendor, the renderer, the extension list, supported viewport size, line size, point size, plus many other details. Some might include a benchmark. Some are standalone benchmarks.

GPU Caps Viewer (Windows XP, Vista 32)
OpenGL Extension Viewer (Windows, Windows x64 and MacOS X)
OpenGL ES benchmark tool (Linux, Symbian, Windows Mobile)
Fur rendering benchmark (Windows)
Futuremark's GL ES benchmark

2.2 GOOGLE SEARCH:

Google.com puts the Internet's wealth of information at your fingertips. The Google Search Appliance does the same for all your corporate information. The Google Search Appliance is an integrated hardware and software product designed to give businesses the productivity-enhancing power of Google search. It's a corporate search solution as simple, powerful and comprehensive as Google itself. The latest version adds key new features around search quality, access control and connectivity.

The Google Search Appliance makes the sea of lost and misplaced data on your web servers, file servers, content management systems, relational databases and business applications instantly available from a single familiar search box. Through an interface as simple and intuitive as Google.com, your employees will have instant, real-time secure access to all the information and knowledge across your entire enterprise – in more than 220 different file formats, and in over 109 different languages.

HARDWARE AND SOFTWARE REQUIREMENTS

3.1 Hardware requirements:

- Pentium or higher processor.
- 512 MB or more RAM
- A standard keyboard, compatible mouse and a VGA monitor

3.2 Software requirements:

This graphics package has been designed for UBUNTU Platform and uses ECLIPSE software

OS : Ubuntu 10.10

Development Tool : Eclipse

Language : C

SOFTWARE DESIGN

4.1 Proposed System

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

The advantages are:

- OpenGL is designed as a streamlined.
- It is a hardware independent interface, it can be implemented on many different hardware platforms.
- With OpenGL, we can draw a small set of geometric primitive such as points, lines and polygons etc.
- It provides double buffering which is vital in providing transformations.
- It is event driven software.
- It provides call back function.

4.2 Detailed Design

Transformation Functions

- Translation:

  Translation is done by adding the required amount of translation quantities to each of the points of the objects in the selected area. If P(x,y) be the a point and (tx,ty) translation quantities then the translated point is given by
  glTranlate(dx,dy,dz);

- Rotation:

  The rotation of an object by an angle 'a' is accomplished by rotating each of the points of the object. The rotated points can be obtained using the OpenGL functions
  glRotate (angle, vx,vy,vz);

- Scaling:

  The scaling operation on an object can be carried out for an object by multiplying each of the points (x,y,z) by the scaling factors sx, sy and sz.
  glScale(sx,sy,sz);

IMPLEMENTATION

5.1 ESCRIPTION:

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.

THIS PROJECT CONTAINS THE FOLLOWING KEY WORDS

glClearColor	: to screen cleared, use black.
glutCreateWindow()	: Tocreate window for oytput.
glVertex()	: To identify the vertices it maybe int,double,float.
Glut	:An introduction to the OpenGL utility ToolKit.
glutCreateWindow()	:Creates a top level window.
glutInitWindowPosition()	: set the initial window position and size.
glutInit	: initial the GLUT library.
glutMainLoop()	: enters the glut event processing loop.

ARCHERY game is single player game where the player hits the target. The target is a block with hole in center & the player should hit the target. There are 15 arrows, 10 blocks.

The arrow is made of three parts tip, shaft and nock. The block is covered by an elastic material which breaks when the tip of the arrow hits it exactly at the center. We have arrow count shown on the screen.

The player should aim the first arrow with lot of concentration as the speed is max initially and decreases as each arrow vanishes. We have used right button of the mouse to help the user know about the instructions.

The target can be hit by arrow by pressing ‘r’ key on the keyboard. If the user wishes to quit or exit from the game he can use the key ‘q’ on the keyboard.

Once the player begins the game the arrow starts moving to hit the target by following the given instruction. As the arrow is heading to the target and finally reaches the end of the screen the arrow count increases indication the no of arrows already used. Once the arrow count becomes 15 the game ends. If the player has hit all the targets then he wins the game, otherwise loses it.

The display of arrow is shown below

Fig 5.1

The target is shown below

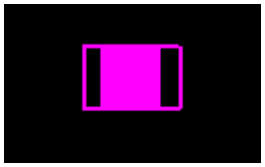


Fig 5.2

5.2 ALGORITHM

Step 1: Initialize the graphics windows and its size using GLUT functions.

Step 2: Register the keyboard and display call backs in main function.

Step 3: Game contains of **10** Blocks and **15** Arrows.

Step 4: Arrows starts moving upwards as soon as we enter the output screen.

Step 5: When the arrow starts moving the key **'r'** is pressed, which moves towards right in order to hit the block.

Step 6: If the key **'r'** is pressed at the correct position it hit the block or else it fails to hit the block.

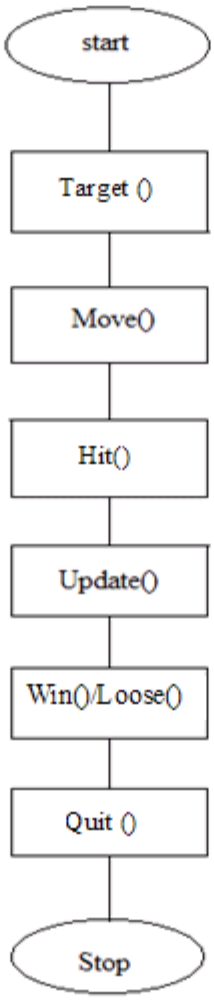
Step 7: If the player fails to hit the **'10'** blocks using **'15'** arrows then it will display a message as "**no arrows game over you lost**".

Step 8: else

Step 9: **Congratulation you won**.

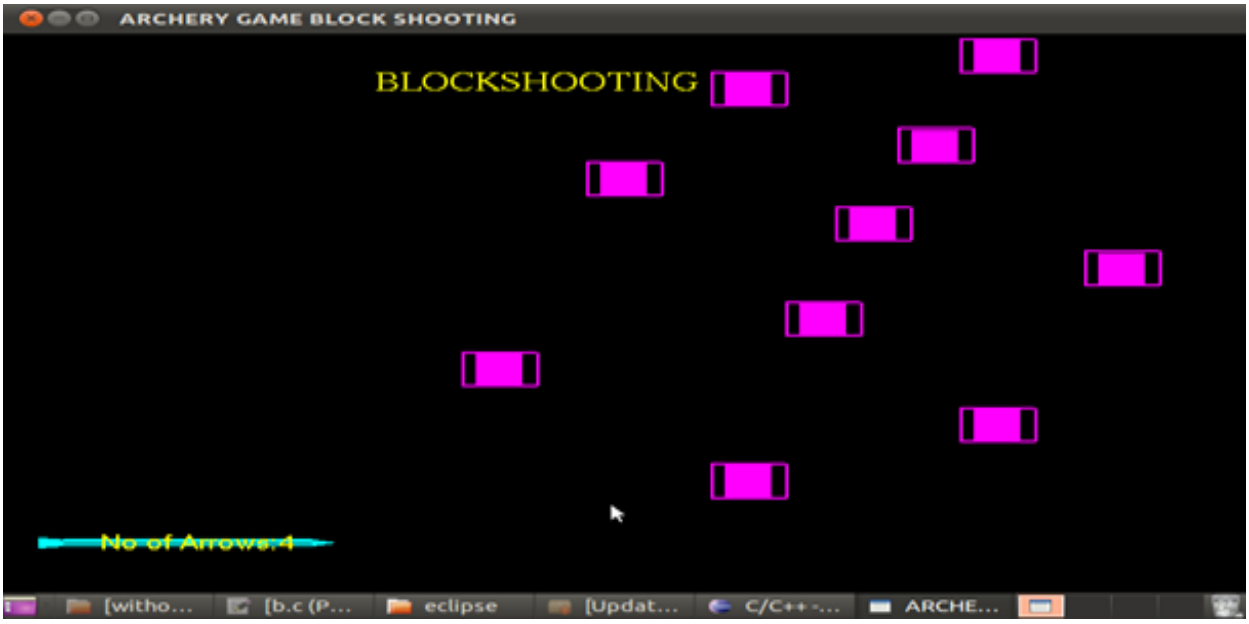
Step 10: By pressing a key **'q'** the player can quit/exit the game at any point of the stage.

5.3 DATA FLOW DIAGRAM



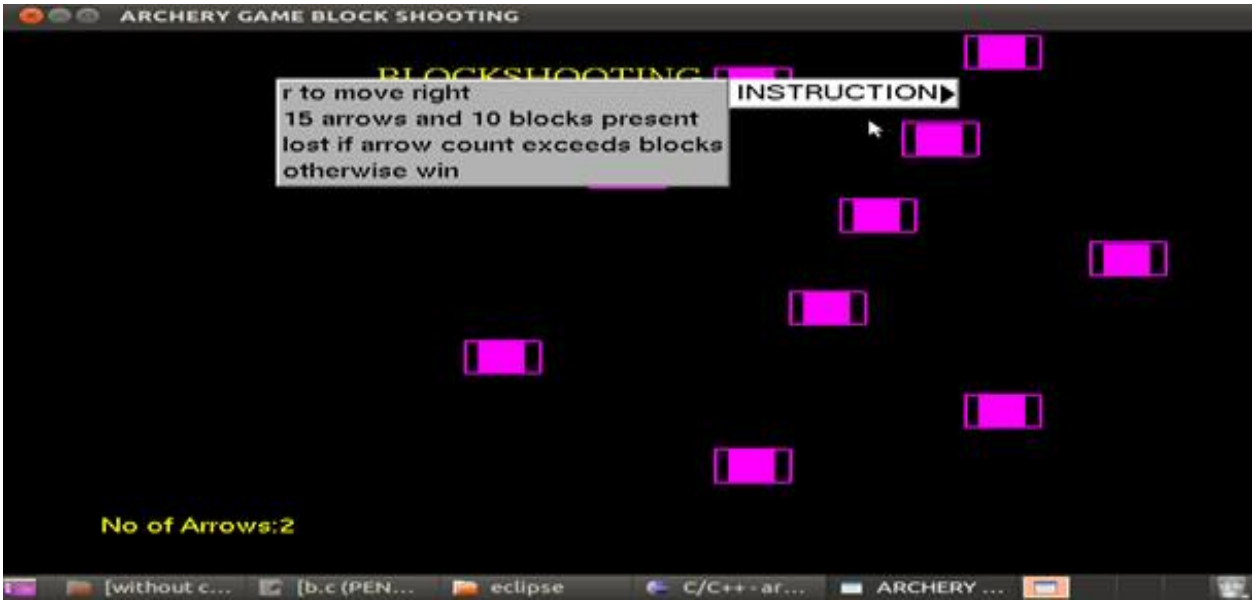
SNAPSHOTS

Figure 6.1:



Initial Position of the Archery Game

Figure: 6.2



Instruction about the game how to play

Figure 6.3:

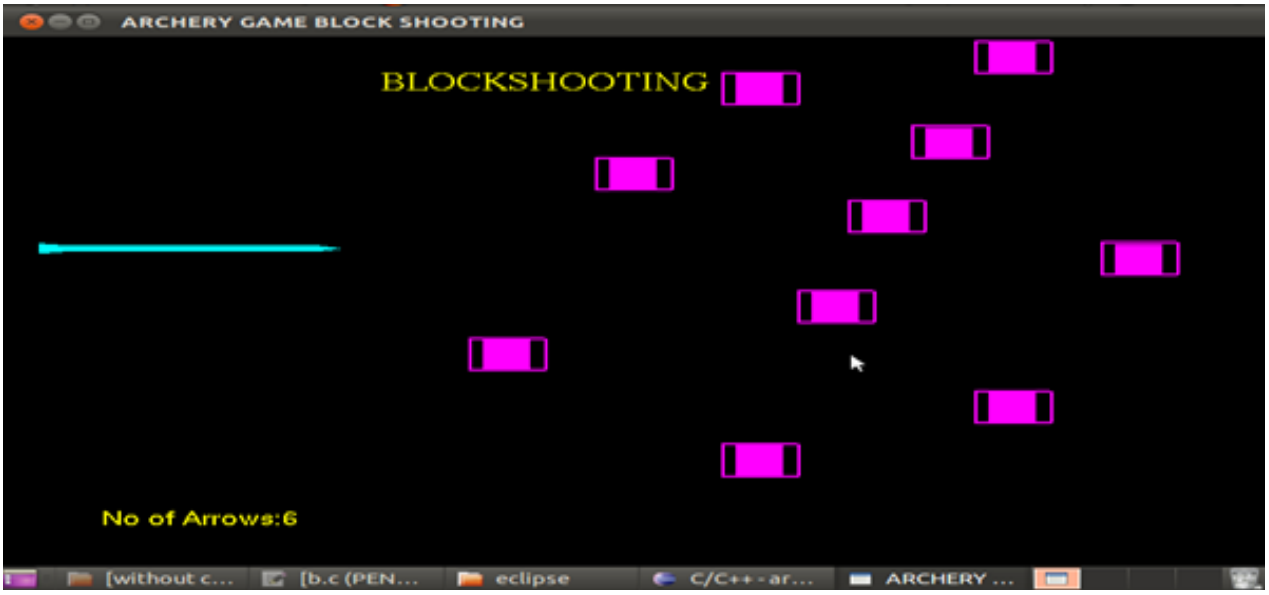


Figure 6.4:

Game Over

Figure 6.5:

Won the game

SOURCE CODE

```
#include<stdio.h>
#include<GL/glut.h>
#include<string.h>
int maxy=600;
int count=0;
int maxx=500;
int n=3;
int m=3;
int
count1=0,count2=0,count3=0,count4=0,count5=0,count6=0,count7=0,count8=0,count
9=0,count10=0;
int x=25, y=50;
char str [10];
void id1();
void id();
void draw_target();
void redraw();
```

/*to display bitmap char*/

```
void bitmap_output(int x, int y, char *string, void *font)
{
int len,i;
glRasterPos2f(x,y);
len=(int)strlen(string);
for(i=0;i<len;i++)
{

glutBitmapCharacter(font,string[i]);
}
return;
}
```

/*based on count display no of arrows and result of game*/

```
void counting()
{
    sprintf(str,"No of Arrows:%d",count);
    bitmap_output(40,40,str,GLUT_BITMAP_HELVETICA_18);

if(count1==1&&count2==1&&count3==1&&count4==1&&count5==1&&count6==1&&count7==
1&&count8==1&&count9==1&&count10==1)
{
    bitmap_output(5,300,"CONGRAGULATION U WON",GLUT_BITMAP_TIMES_ROMAN_24);
    glutIdleFunc(NULL);
}
else if(count>=15)
{
    sprintf(str,"No of Arrows:%d,NO ARROWS GAME OVER U LOST",count);
bitmap_output(5,300,str,GLUT_BITMAP_TIMES_ROMAN_24);
glutIdleFunc(NULL);
}
}
```

/*TO CHECK WHETHER ARROW HITS TARGET*/

```
void disa()
{

if((x+110==300)&&(y>=435&&y<=465)&&(!count1))
{
count1=1;

x=25;
y=0;
count++;
glutIdleFunc(id);
}
else if ((x+110==375)&&(y>=385&&y<=415)&&(!count2))
{
count2=1;

x=25;
y=0;
count++;
glutIdleFunc(id);
}
else if ((x+110==399) &&(y>=465&&y<=495) &&(!count3))
{
count3=1;
x=25;




y=0;
count++;
```

```
glutIdleFunc(id);
}


else if((x+110==249)&&(y>=355&&y<=385)&&(!count4))
{

count4=1;

x=25;
y=0;
count++;
glutIdleFunc(id);
}
else if((x+110==351)&&(y>=315&&y<=345)&&(!count5))
{
count5=1;

x=25;
y=0;
count++;
glutIdleFunc(id);
}
else if((x+110==450)&&(y>=275&&y<=305)&&(!count6))
{
count6=1;

x=25;
y=0;
count++;
glutIdleFunc(id);
}
else if((x+110==330)&&(y>=230&&y<=260)&&(!count7))
{
count7=1;


x=25;
y=0;
count++;
glutIdleFunc (id);
}
else if((x+110==201)&&(y>=185&&y<=215)&&(!count8))
{
count8=1;









x=25;
```

```
y=0;

count++;
glutIdleFunc(id);
}
else if((x+110==399)&&(y>=135&&y<=165)&&(!count9))
{
count9=1;

x=25;
y=0;


count++;
glutIdleFunc(id);
}
else if((x+110==300)&&(y>=85&&y<=115)&&(!count10))


{
count10=1;

x=25;
y=0;

count++;
glutIdleFunc(id);
}
}
```

/*to move arrow up*/

```
void id()
{
y+=n;
disa();
if(y>maxy)
    {
    y=0;
    count++;
    }
glutPostRedisplay();
}
```

```
/*to draw the arrow*/
```

```
void disp()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glColor3f(1,1,0);
bitmap_output(150,450,"BLOCKSHOOTING",GLUT_BITMAP_TIMES_ROMAN_24);
counting();
```

// Drawing of arrow

```
glColor3f(0,1,1);
glBegin(GL_LINES);
glVertex2d(x,y);
glVertex2d(x+100,y);
glEnd();
glLineWidth(2);
glBegin(GL_LINES);


glVertex2d(x,y+2);
glVertex2d(x+100,y+2);
glEnd();
glBegin(GL_LINES);
glVertex2d(x,y-2);
glVertex2d(x+100,y-2);
glEnd();
glBegin(GL_TRIANGLES);


glVertex2d(x+100,y+3);
glVertex2d(x+110,y);
glVertex2d(x+100,y-3);
glEnd();
glBegin(GL_QUADS);
glVertex2d(x,y+3);
glVertex2d(x,y-3);
glVertex2d(x-10,y-5);
glVertex2d(x-10,y+5);
glEnd();

draw_target();            // Drawing of target

glFlush();

glutSwapBuffers();
}
```

/*to clear screen & set projection mode*/

```
void init()
{
glClearColor(0,0,0,1);
glColor3f(1,0,0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

gluOrtho2D(0,500,0,500);
glMatrixMode(GL_MODELVIEW);
}
```

/*to draw the target inside line loop*/

```
void draw_target()
{


if(count1==0)
{
glColor3f(1,0,1);
glPointSize(30);
glBegin(GL_POINTS);
glVertex2d(300,450);


glEnd();
glBegin(GL_LINE_LOOP);

glVertex2d(285,465);
glVertex2d(315,465);
glVertex2d(315,435);
glVertex2d(285,435);
glEnd();
}
else




{
    glColor3f(1,1,1);
    glPointSize(20);
    glBegin(GL_POINTS);
    glVertex2d(300,450);
    glEnd();
}




if(count2==0)
{
```

```
    glColor3f(1,0,1);
    glPointSize(30);
    glBegin(GL_POINTS);
    glVertex2d(375,400);
    glEnd();
glBegin(GL_LINE_LOOP);


glVertex2d(360,415);
glVertex2d(390,415);
glVertex2d(390,385);
glVertex2d(360,385);
glEnd();
}
else

{
    glColor3f(1,1,1);
    glPointSize(20);
    glBegin(GL_POINTS);

    glVertex2d(375,400);
    glEnd();

}
if(count3==0)
{
    glColor3f(1,0,1);
    glPointSize(30);
    glBegin(GL_POINTS);
    glVertex2d(400,480);
    glEnd();
glBegin(GL_LINE_LOOP);
glVertex2d(385,495);
glVertex2d(415,495);


glVertex2d(415,465);
glVertex2d(385,465);
glEnd();
}else
{
    glColor3f(1,1,1);
        glPointSize(20);
        glBegin(GL_POINTS);
        glVertex2d(400,480);
        glEnd();


}




if(count4==0)
{
```

```
    glColor3f(1,0,1);
    glPointSize(30);
    glBegin(GL_POINTS);
    glVertex2d(250,370);
    glEnd();
glBegin(GL_LINE_LOOP);
glVertex2d(235,385);
glVertex2d(265,385);
glVertex2d(265,355);
glVertex2d(235,355);
glEnd();
}else
{
       glColor3f(1,1,1);
        glPointSize(20);
        glBegin(GL_POINTS);
        glVertex2d(250,370);
        glEnd();


}
if(count5==0)
{
    glColor3f(1,0,1);
    glPointSize(30);
    glBegin(GL_POINTS);
    glVertex2d(350,330);

  nd();
glBegin(GL_LINE_LOOP);
glVertex2d(335,345);
glVertex2d(365,345);
glVertex2d(365,315);
glVertex2d(335,315);
glEnd();
}else
{
        glColor3f(1,1,1);
        glPointSize(20);
        glBegin(GL_POINTS);


 glVertex2d(350,330);
        glEnd();

}









if(count6==0)
{
```

```
    glColor3f(1,0,1);
    glPointSize(30);



  glBegin(GL_POINTS);
    glVertex2d(450,290);
    glEnd();
glBegin(GL_LINE_LOOP);
glVertex2d(435,305);
glVertex2d(465,305);




glVertex2d(465,275);
glVertex2d(435,275);
glEnd();

}else
{
    glColor3f(1,1,1);
        glPointSize(20);
        glBegin(GL_POINTS);
        glVertex2d(450,290);
        glEnd();

}
if(count7==0)

{
    glColor3f(1,0,1);
    glPointSize(30);
    glBegin(GL_POINTS);
    glVertex2d(330,245);
    glEnd();
glBegin(GL_LINE_LOOP);
glVertex2d(315,260);
glVertex2d(345,260);


glVertex2d(345,230);
glVertex2d(315,230);
glEnd();
}
else
{
    glColor3f(1,1,1);
        glPointSize(20);
        glBegin(GL_POINTS);
        glVertex2d(330,245);



       glEnd();
```

```
}




if(count8==0)
{
    glColor3f(1,0,1);
    glPointSize(30);
    glBegin(GL_POINTS);
    glVertex2d(200,200);
    glEnd();
glBegin(GL_LINE_LOOP);
glVertex2d(185,215);
glVertex2d(215,215);
glVertex2d(215,185);
glVertex2d(185,185);
glEnd();
}
else {




    glColor3f(1,1,1);
        glPointSize(20);
        glBegin(GL_POINTS);
        glVertex2d(200,200);
        glEnd();

}
if(count9==0)
{
    glColor3f(1,0,1);

    glPointSize(30);

    glBegin(GL_POINTS);
    glVertex2d(400,150);
    glEnd();
glBegin(GL_LINE_LOOP);
glVertex2d(385,165);
glVertex2d(415,165);
glVertex2d(415,135);
glVertex2d(385,135);
glEnd();
}



else
{
```

```
    glColor3f(1,1,1);
        glPointSize(20);
        glBegin(GL_POINTS);
        glVertex2d(400,150);
        glEnd();

}




if(count10==0)
{
    glColor3f(1,0,1);
    glPointSize(30);


  glBegin(GL_POINTS);
    glVertex2d(300,100);
    glEnd();
glBegin(GL_LINE_LOOP);
glVertex2d(285,115);
glVertex2d(315,115);

glVertex2d(315,85);
glVertex2d(285,85);
glEnd();
}
else
{
    glColor3f(1,1,1);
        glPointSize(20);
        glBegin(GL_POINTS);
        glVertex2d(300,100);




       glEnd();

}

glFlush();
}
```

/* to move the arrow left wen 'r' pressed*/

```
void id1()
{
x+=m;
disa();
if(x+110>maxx)
{
x=25;
y=0;
count++;
glutIdleFunc(id);

}




glutPostRedisplay();
}
```

/*set key to perform desired operation*/

```
void keys(unsigned char k,int x,int y)
{
if(k=='r')
glutIdleFunc(id1);


if(k=='q')
    exit(0);
}
```

/*sub menu to display instructions*/

```
void demo_menu(int i)
{
switch(i)
{
case 5:
case 6:
case 7:
case 8:break;
}
}
```

/*sub menu to display designer names*/

```
void demo(int i)
{
switch(i)
{
case 9:
case 10:

case 11:break;
}
}




void game(int id)
{
    switch(id)
    {

    }
}


/*main to call display,keyboard and idle func*/

int main(int argc,char **argv)
{
int sub_menu;
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(900,900);
glutCreateWindow("ARCHERY GAME BLOCK SHOOTING");

sub_menu=glutCreateMenu(demo_menu);
glutAddMenuEntry("r to move right",5);
glutAddMenuEntry("15 arrows and 10 blocks present",6);



glutAddMenuEntry("lost if arrow count exceeds blocks",7);
glutAddMenuEntry("otherwise win",8);
glutCreateMenu(game);
glutAddSubMenu("INSTRUCTION",sub_menu);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutDisplayFunc(disp);
```

```
glutIdleFunc(id);
glutKeyboardFunc(keys);
init();
glEnable(GL_DEPTH_TEST);
glutMainLoop();
return 0;
}
```