**Chapter 1**

**INTRODUCTION**

**1.1** **Introduction to the Computer 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.

The Graphics system is a computer system which consists of almost all the components of general purpose computer. There are five major components.

They are:

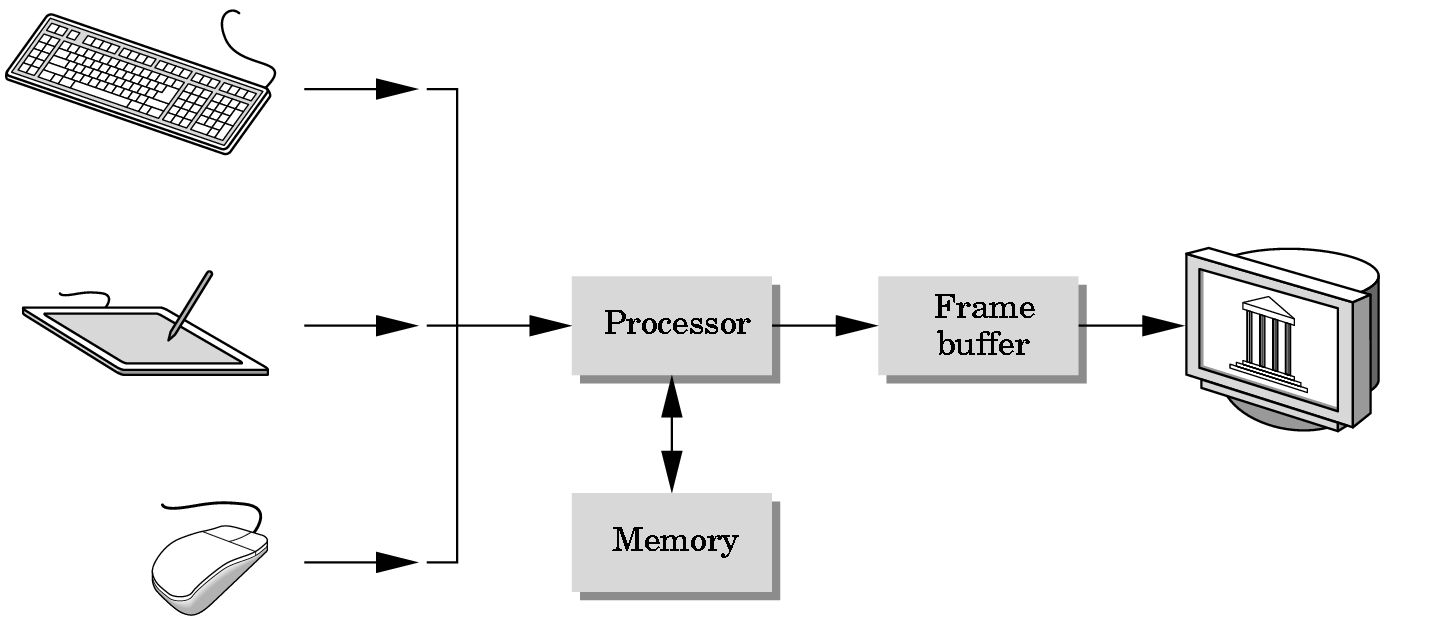
**Input devices:** Graphics systems provide a keyboard and another one input device. Most common is the mouse, the joy stick and the data tablet. This provides the positional information.

**Processor:** The main function of the graphical processor is to take specification of the graphical primitives. Generated by the application programs and to assign values to the pixels in the frame buffer that best represents these entities .

**Frames Buffer:** The frame buffers are the core elements of the graphics system. Its resolution determines the detail of the image we can view.

**Memory:** Frame buffer is usually implemented with special type of memories chips that enable fast redisplays on the contents of the frame buffer.

**Output Devices:** The dominant type of the display or monitor has been cathode ray tube Flat panel technologies are now more popular.



**Fig 1.1 Graphics System**

**1.2 OpenGL**

OpenGL (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.

**Features of OpenGL**

The main features of OpenGL are:

* It provides 3D geometric objects such as lines, polygons, triangle, meshes, spheres, cubes, quadric surface, curved surfaces.
* It provides 3D modelling transformations and viewing functions to create views of 3D scenes using the idea of a virtual camera.
* It supports high-quality rendering of scenes, including hidden-surface removal, multiple light sources, material types, transparency, textures, blending, fog.
* It provides display lists for creating graphics caches and hierarchical models. It also supports the interactive “picking” of objects.
* It supports the manipulation of images as pixels, enabling frame-buffer effects such as antialiasing, motion blur, depth of field and soft shadows.

A key feature of the design of OpenGL is the separation of interaction from rendering. OpenGL itself is concerned only with graphics rendering.

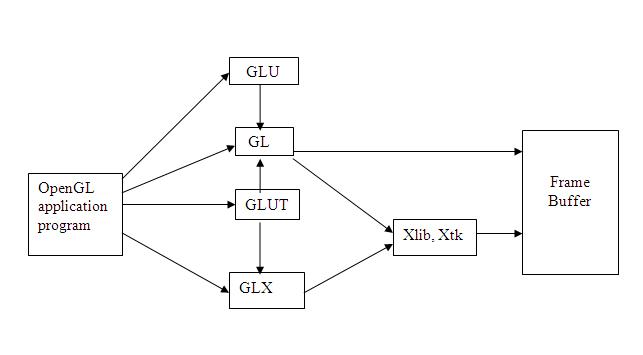
**Support Libraries**

OpenGL has three support libraries like GL, GLU, GLUT. The development of utility libraries have greatly extend the low-level functionality of OpenGL.

GL provides lower level functions for drawing points, lines and polygons. All GL function name start with “gl”.

GLU provides functions for drawing more complex primitives of OpenGL such as curves and surfaces. It also functions to help specify 3D views of scenes. All GLU function start with “glu”.

GLUT provides the facilities for interaction that OpenGL lacks. It provides functions for managing windows on the display screen and handling input events from the mouse and keyboard. It provides some rudimentary tools for creating Graphical User Interface(GUIs).



**Fig 1.2 Supporting Libraries**

**1.3 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.

**Chapter 2**

**LITERATURE SURVEY**

**2.1** [**WWW.OPENGL.ORG:**](http://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 entry points version 3.1 removed from earlier versions will not be available, and that any entry points 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.

**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.

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.

**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 INBUILT FUNCTIONS IN COMPUTER GRAPHICS**

**1. glClearColor()** - void glClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a)- Sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating – point numbers between 0.0 and 1.0.

**2. glPointSize() -** void glPointSize(GLfloat size) - Sets the point size attribute in pixels.

**3. glFlush()** - void glFlush() - Forces any buffered OpenGL commands to execute.

**4. glutInit()** - void glutInit(int \*argc, char \*\*argv) - Initializes GLUT. The arguments from main are passed in and can be used by the application.

**5. glutCreateWindow()** - int glutCreateWindow(char \*title) - Creates a window on the display. The string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

**6. glutInitDisplayMode()** - void glutInitDisplayMode(unsigned int mode) - Requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT\_RGB, GLUT\_INDEX) and buffering (GLUT\_SINGLE, GLUT\_DOUBLE).

**7. glutInitWindowSize()** - void glutInitWindowSize(int width, int height) - Specifies the initial height and width of the window in pixels.

**8. glViewport()** - void glViewport(int x, int y, GLsizei width, GLsizei height) - Specifies a width \* height viewport in pixels whose lower – left corner is at (x,y) measured from the origin of the window.

**9. glutMainLoop()** - void glutMainLoop() - Cause the program to enter an event – processing loop. It should be the last statement in main.

**10. glutDisplayFunc()** - void glutDisplayFunc(void (\*func) (void)) - Registers the display function func that is executed when the window needs to be redrawn.

**11. glutPostRedisplay()** - void glutPostRedisplay() - Requests that the display callback be executed after the current callback returns.

**12. glutSwapBuffers()** - void glutSwapBuffers() - Swaps the front and back buffers.

**13. glutMouseFunc()** - void glutMouseFunc(void \*f(int button, int state, int x, int y) - Registers the mouse callback function f. The callback function returns the button (GLUT\_LEFT\_BUTTON, GLUT\_MIDDLE\_BUTTON, GLUT\_RIGHT\_BUTTON), the state of the button after the event (GLUT\_UP, GLUT\_DOWN), and the position of the mouse relative to the top – left corner of the window.

**14. glutReshapeFunc()** - void glutReshapeFunc(void \*f(int width, int height)) - Registers the reshape callback function f. The callback function returns the height and width of the new window. The reshape callback invokes a display callback.

**15.glutKeyboardFunc() -** void glutKeyboardFunc(void \*f(char key,int width,int height))-Registers the display callback function f.The callback function returns the ASCII code of the key pressed and the position of the mouse.

**16. glutIdleFunc()** - void glutIdleFunc(void (\*f) (void)) - Registers the display callback function f that is executed whenever there are no other events to be handled.

**17. glEnable() -** void glEnable(GLenum feature) - Enables an OpenGL feature. Features that can be enabled include GL\_DEPTH\_TEST, GL\_LIGHTING, GL\_LIGHTi, GL\_TEXTURE\_1D, GL\_TEXTURE\_2D, GL\_TEXTURE\_3D, GL\_LINE\_SMOOTH, GL\_POLYGON\_SMOOTH, GL\_POINT\_SMOOTH, GL\_BLEND, GL\_LINE\_STIPPLE, GL\_POLYGON\_STIPPLE, GL\_FOG, GL\_NORMALIZE.

**18. glMatrixMode()** - void glMatrixMode(GLenum mode) - Specifies which matrix will be affected by subsequent transformations. Mode can be GL\_MODELVIEW, GL\_PROJECTION, or GL\_TEXTURE.

**19. glLoadIdentity()** - void glLoadIdentity() - Sets the current transformation matrix to an identity matrix.

**20. glTranslate()** - void glTranslate[fd] ( TYPE x, TYPE y, TYPE z) - Alters the current matrix by a displacement of (x,y,z). TYPE is either GLfloat or GLdouble.

**21. glRotate()** - void glRotate[fd](TYPE angle, TYPE dx, TYPE dy, TYPE dz) - Alters the current matrix by a rotation of angle degrees about the axis (dx,dy,dz). TYPE is either GLfloat or GLdouble.

**Chapter 3**

**REQUIREMENTS**

A **Software Requirements Specification** (**SRS**) is a complete description of the behavior of the system to be developed. It includes a set of [use cases](http://en.wikipedia.org/wiki/Use_case) that describe all the interactions the users will have with the software. Use cases are also known as [functional requirements](http://en.wikipedia.org/wiki/Functional_requirements). In addition to use cases, the SRS also contains nonfunctional (or supplementary) requirements. [Non-functional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements) are requirements which impose constraints on the design or implementation .

**3.1 REQUIREMENT SPECIFICATION**

**3.1.1 Hardware Requirements:**

        The hardware requirement is minimal and the software can run with minimal requirements. The basic requirements are as enlisted below:

1. Processor:  Intel 486/ Pentium processor or a processor with higher specifications
2. Processor speed: 500MHz or above.
3. RAM : 64MB or above
4. Storage space : 2MB or above
5. Monitor resolution: A color monitor with a minimum resolution of 640 \* 480.

**3.1.2** **Software Requirements:**

1. An MS-DOS based operating system like Windows 98, Windows 2000 or Windows XP is the platform required to develop the 3D simulation.

2. A C/C++ (integrated with OPEN GL) compiler like Eclipse is required for compiling the source code to make the executable file which can then be directly executed.

1. A built in graphics library; glut.h is required for drawing the layout of the game.
2. glut32.dll for running the application.

**Chapter 4**

**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:

1. OpenGL is designed as a streamlined.

2. It is a hardware independent interface, it can be implemented on many different hardware platforms.

3. With OpenGL, we can draw a small set of geometric primitive such as points, lines and polygons etc.

4. It provides double buffering which is vital in providing transformations.

5. It is event driven software.

6. 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);

**Chapter 5**

**IMPLEMENTATION**

**5.1** **DESCRIPTION:**

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.

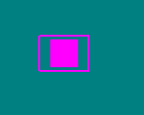
Once the player begins the game the arrow starts moving to hit the target by following the given instruction. As the arrow is heading to the target and finally reaches the end of the screen the arrow count increases indication the no of arrows already used. Once the arrow count becomes 15 the game ends. 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 Arrow**

**The target is shown below**



**Fig 5.2 TARGET**

**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 inorder 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 theblock.

**Step 7**: If the player fails to hit the **‘10’** blocks using **‘15’** arrows then it will display a messageas “**no arrows game over you lost**”.

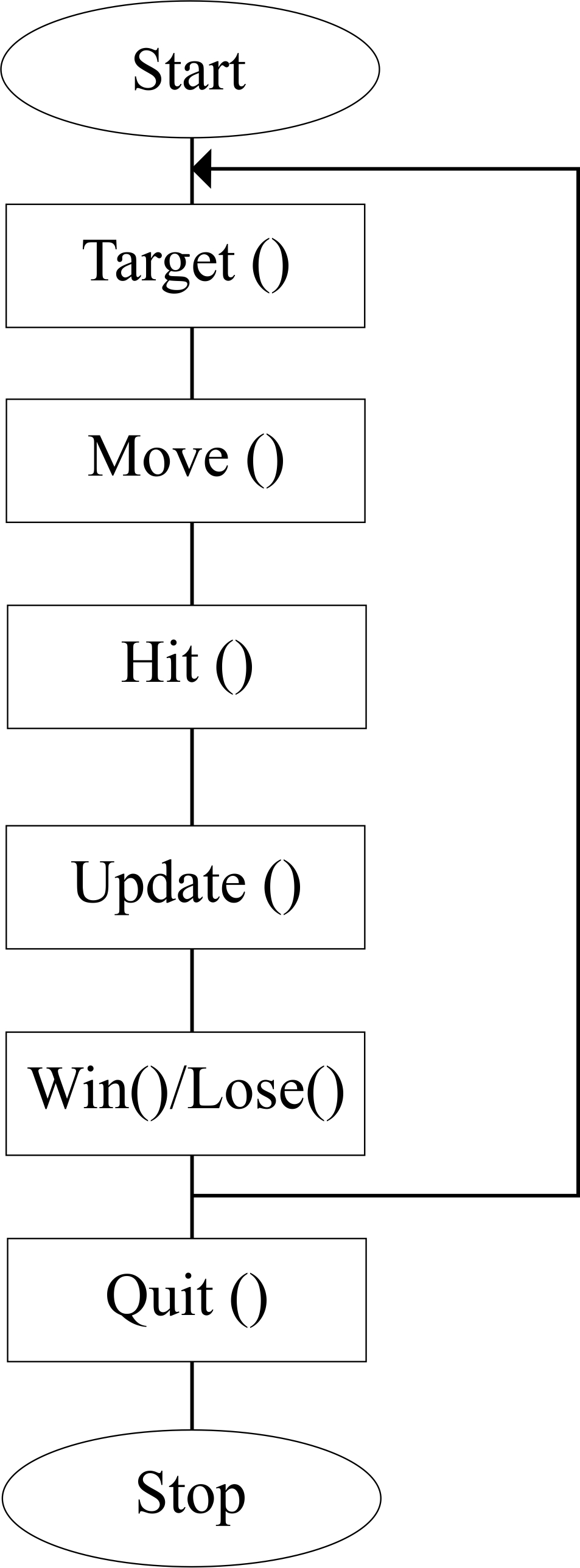
**Step 8**: else

**Step 9**: **Congratulation you won**.

**Step 10:** By pressing a key ‘n’ the player can start new game.

**Step 11**: By pressing a key **‘q’** the player can quit/exit the game at any point of the stage.

* 1. **DATA FLOW DIAGRAM**



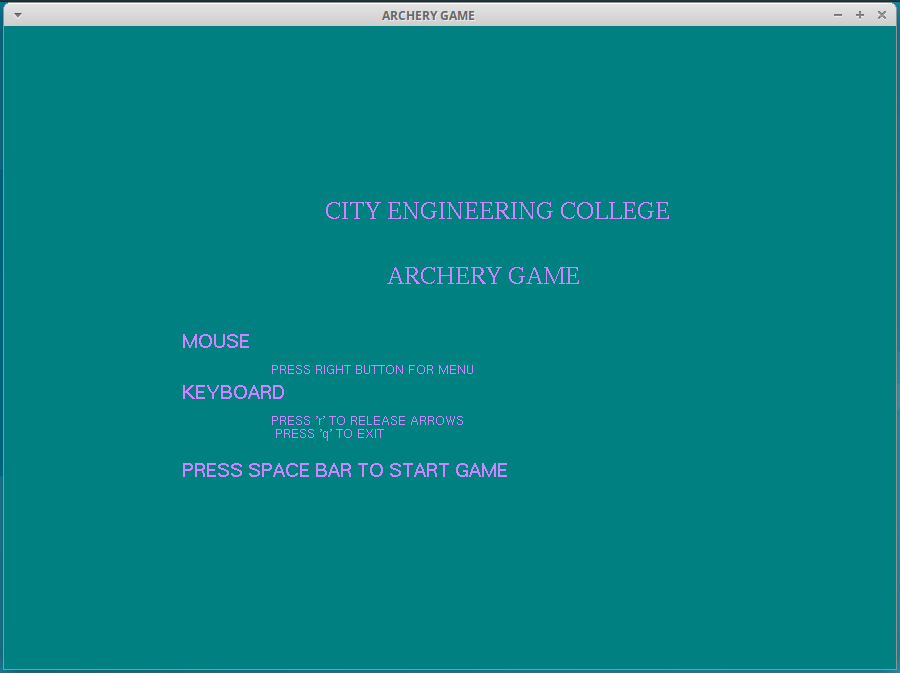
**Fig5.3: Data flow diagram**

**5.4 CODE**

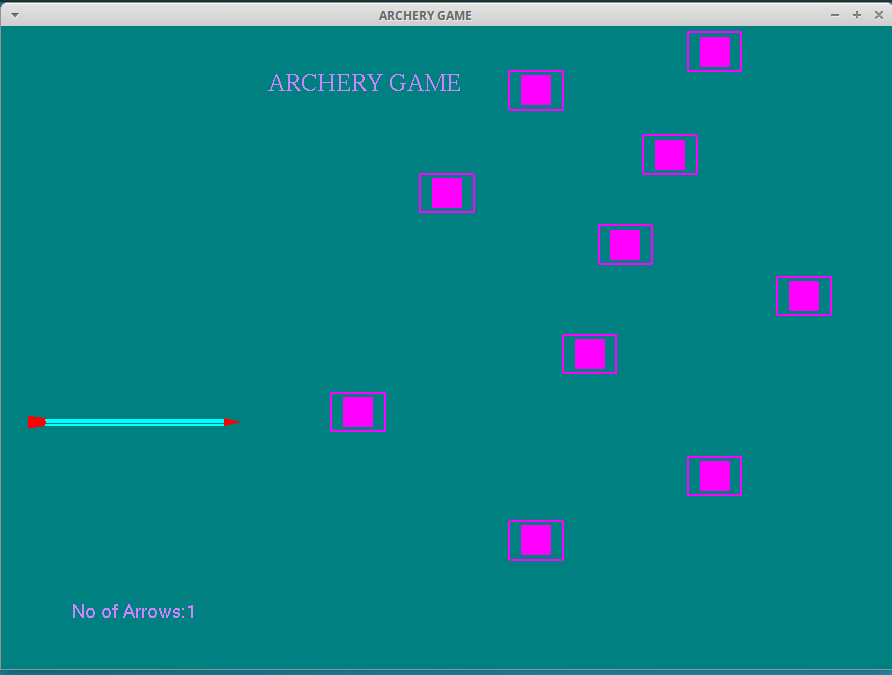
#include<stdio.h>  
#include<GL/glut.h>  
#include<string.h>  
int maxy=600;  
int count=0;  
int maxx=500;  
int view=0;  
int n=3;  
int m=3;  
int count1=0,count2=0,count3=0,count4=0,count5=0,count6=0,count7=0,count8=0,count9=0,count10=0;  
int x=25, y=50;  
char str [10];  
void id1();  
void id();  
void draw\_target();  
  
  
  
/\*based on count display no of arrows and result of game\*/  
void counting()  
{  
    sprintf(str,"No of Arrows:%d",count);  
    display\_string(40,40,str,2);  
    if(count1==1&&count2==1&&count3==1&&count4==1&&count5==1&&count6==1&&count7==1&&count8==1&&count9==1&&count10==1)  
    {  
        display\_string(5,300,"CONGRATULATION U WON",1);  
        display\_string(5,100,"PRESS n TO PLAY AGAIN",2);  
        display\_string(5,80,"PRESS q TO EXIT",2);  
  
        glutIdleFunc(NULL);  
    }  
    else if(count>=15)  
    {  
        display\_string(5,300," GAME OVER YOU LOSE",1);  
        display\_string(5,100,"PRESS n TO PLAY AGAIN",2);  
        display\_string(5,80,"PRESS q TO EXIT",2);  
  
  
        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()  
{  
    if(view==1)  
    {  
        y+=n;  
        disa();  
        if(y>maxy)  
        {  
            y=0;  
            count++;  
        }  
    }  
    glutPostRedisplay();  
}  
  
/\*to display bitmap char (strings)\*/  
void display\_string(int x, int y, char \*string, int font)  
{  
    int len,i;  
    glColor3f(0.8,0.52,1.0);  
    glRasterPos2f(x, y);  
    len = (int) strlen(string);  
    for (i = 0; i < len; i++) {  
    if(font==1)  
    glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24,string[i]);  
    if(font==2)  
    glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18,string[i]);  
    if(font==3)  
            glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12,string[i]);  
    if(font==4)  
    glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_10,string[i]);  
    }  
  
}  
  
void display1(void)  
{  
  
    glClearColor(0.0,0.5,0.5,1.0);  
    display\_string(180,350,"CITY ENGINEERING COLLEGE",1);  
    display\_string(215,300,"ARCHERY GAME",1);  
    display\_string(100,250,"MOUSE",2);  
    display\_string(150,230,"PRESS RIGHT BUTTON FOR MENU",3);  
    display\_string(100,210,"KEYBOARD",2);  
    display\_string(150,190,"PRESS 'r' TO RELEASE ARROWS",3);  
    display\_string(150,180," PRESS 'q' TO EXIT",3);  
    display\_string(100,150,"PRESS SPACE BAR TO START GAME",2);  
    glutPostRedisplay();  
    glutSwapBuffers();  
  
}  
  
  
  
/\*to clear screen & set projection mode\*/  
void init()  
{  
    glClearColor(0,0.5,0.5,1);  
    glColor3f(1,0,0);  
    glMatrixMode(GL\_PROJECTION);  
    glLoadIdentity();  
    gluOrtho2D(0,500,0,500);  
    glMatrixMode(GL\_MODELVIEW);  
}  
  
/\*to draw the arrow\*/  
void disp()  
{  
    glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);  
    if(view==0)  
    {  
        init();  
        display1();  
    }  
  
    else  
    {  
        glLoadIdentity();  
        glColor3f(1,1,0);  
        display\_string(150,450,"ARCHERY GAME",1);  
        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);  
        glColor3f(1,0,0);//color of tip  
        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 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);  
glEnd();  
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 when '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)  
{  
    switch(k){  
  
    case ' ':  
            disp();  
            view=1;  
            break;  
    case 'r': glutIdleFunc(id1);  
    break;  
    case 'n': /\* new game \*/  
        maxy=600;  
         count=0;  
         maxx=500;  
         view=1;  
         n=3;  
         m=3;  
        count1=0,count2=0,count3=0,count4=0,count5=0,count6=0,count7=0,count8=0,count9=0,count10=0;  
         x=25, y=50;  
  
        disp();  
        glutIdleFunc(id);  
  
    break;  
    case 'q':        /\* quit game \*/  
    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,submenu;  
glutInit(&argc,argv);  
glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);  
glutInitWindowSize(900,900);  
glutCreateWindow("ARCHERY GAME");  
sub\_menu=glutCreateMenu(demo\_menu);  
glutAddMenuEntry("r to release arrow",5);  
glutAddMenuEntry("15 arrows and 10 blocks present",6);  
glutAddMenuEntry("lost if arrow count exceeds blocks",7);  
glutAddMenuEntry("otherwise win",8);  
submenu=glutCreateMenu(demo);  
glutAddMenuEntry("AKASH",9);  
glutAddMenuEntry("ZEESHAN",10);  
glutCreateMenu(game);  
glutAddSubMenu("INSTRUCTION",sub\_menu);  
glutAddSubMenu("ABOUT",submenu);  
glutAddMenuEntry("QUIT",2);  
glutAttachMenu(GLUT\_RIGHT\_BUTTON);  
glutDisplayFunc(disp);  
glutIdleFunc(id);  
glutKeyboardFunc(keys);  
glEnable(GL\_DEPTH\_TEST);  
glutMainLoop();  
return 0;  
}

**Chapter 6**

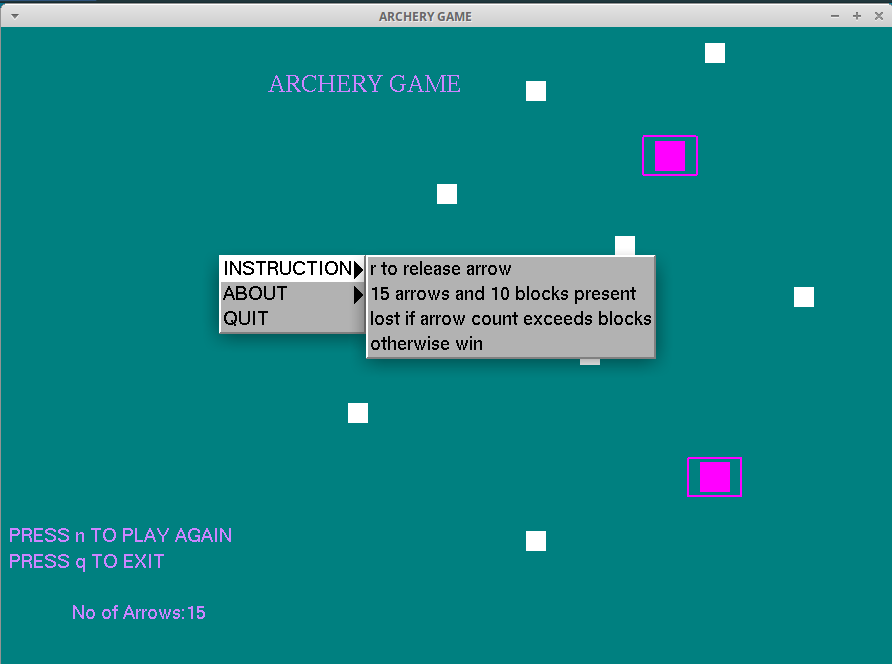
**SNAPSHOTS**



**Figure 6.1: Staring page**



**Figure 6.2: Initialization of arrow and blocks**



**Figure: 6.3: Instruction about the game, how to play**



**Figure: 6.4: Game over you lose**



**Figure: 6.5: Won the game**

**CONCLUSION AND FUTURE ENHANCEMENTS**

We have tried out level best to build the project efficiently and correctly and have succeeded in building a better project, but may not be a best project. We have implemented the required functions which we had stated earlier. After all testing process, the game is now ready to be played.

We strive to provide clear explanation and inter-subject continuity in our presentation illustrative examples are used to substantiate discussion on abstract concepts while the primary mission is to offer relative well-focused introduction to the fundamental theory and underlying technology, significant variations in such matters as basic definitions and implementation protocols are presented in order to have a reasonably broad coverage of the field. In addition, interesting applications are introduced as early as possible to highlight the usefulness of the graphics technology. However, user requirements tend to change and provision for change in design has been allowed .if no major requirements, which are too complex for current design, are made then the current design holds.

Algorithms and programming examples are given in the pseudocode that resembles the programming language, which shares similar syntax and basic constructs with other widely used languages such as c++ and java. We hope that the relative simplicity of the c style code presents little grammatical difficulty and hence makes it easy for you to focus your attention on the technical substance of the code.

**REFERENCES**

(1) Interactive Computer Graphics A Top-Down Approach with OpenGL-Edward Angel, 5th Edition, Addison-Wesley, 2008.

(2) Computer Graphics Using OpenGL-F.S. Hill, Jr., 2nd Edition, Pearson education, 2001.

(3) Computer Graphics-James D Foley, Andries Van Dam, Steven K Fiener, John F Hughes, Addison-Wesley, 1997.

(4) www.wikepedia.com

(5) www.opengl.org

(6) www.stackoverflow.com