

Visvesvaraya Technological University

**BELGAVI, KARNATAKA**

**COMPUTER GRAPHICS MINI PROJECT REPORT ON ARCHERY GAME WITH A TWIST**

Mini Project submitted in partial fulfillment of the Requirement for the 6th semester of

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted by**

**Name USN**

**SANATH GOUTHAM 4JN19CS089**

**PRATHVI NADIG 4JN19CS071**

**SAMRUDH H 4JN19CS087**

**SHOAIB KALEEM SAYAD 4JN19CS090**

**Under the guidance of**

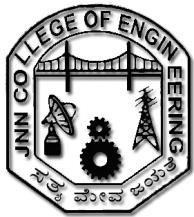
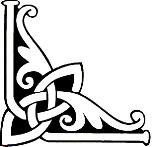
**Mrs. SUSHMA R B B.E, MTech. Mr. CHAKRAPANI D S B.E, MTech.**

**Asst.Prof., Dept. of CS&E Asst.Prof., Dept. of CS&E**

**JNNCE, SHIMOGA JNNCE, SHIMOGA**

Department of Computer Science & Engineering Jawaharlal Nehru New College of Engineering Shimoga - 577 204

JULY 2022



**National Educational Society ®**

**J.N.N COLLEGE OF ENGINEERING SHIVAMOGGA-577204.**

CERTIFICATE

***This is to certify that the mini project report entitled***

“ARCHERY GAME WITH A TWIST**”**

**is carried out by**

**N a m e U S N**

**SANATH GOUTHAM 4JN19CS089**

**PRATHVI A NADIG 4JN19CS071**

**SAMRUDH H 4JN19CS087**

**SHOAIB KALEEM SAYAD 4JN19CS090**

**In partial fulfillment of the requirement for the 6 t h semester of CSE as per the university regulations during the academic year 2021-22 .**

**Signature of Guide Signature of Guide**

**Mrs. Sushma R B B.E, MTech. Mr. Chakrapani D S B.E, MTech**

**Asst.Prof., Dept. of CS&E Asst.Prof., Dept. of CS&E**

**JNNCE, SHIMOGA JNNCE, SHIMOGA**

**Signature of HOD**

**Dr. K M Poornima MTech., Ph.D.**

**Professor & HOD, Dept of CS&E**.

**Signature of Examiners**

**Examiner 1: Examiner 2:**

**ABSTRACT**

A 2d graphics-based archery game is a great start for a student who starts learning computer graphics & visualization. The development of the game has large scope to learn computer graphics from scratch. We used the OpenGL utility toolkit to implement the algorithm, writing it in C++ language.

Finally, we could say by developing the game we have learned the basics of computer graphics and in the future, by developing it further we shall learn more. It will be our pleasure if we could develop in 3d graphics package.

**ACKNOWLEDGEMENT**

At the very outset of this report on **“ARCHERY GAME WITH A TWIST”**, we would like to extend our sincere & heartfelt obligation towards all the persons who have helped us in this endeavor. Without their active guidance, help, cooperation & encouragement, we would not have made headway in the project.

We are grateful to our institution **JNN College of Engineering** and the **Department of Computer Science and Engineering** for imparting us the knowledge with which we can do our best.

We would like to thank our beloved guides **Mrs. Sushma R B,** Asst. professor, Dept. Of Computer Science and Engineering and **Mr. Chakrapani D S***.* Asst. Professor, Dept. Of Computer Science and Engineering*.* We are highly indebted to them for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would also like to express our gratitude and thanks to **Dr. K M Poornima** Professor and Head of the Department, of Computer Science and Engineering, and **Dr*.* K. Nagendra Prasad,** Principal, JNN College of Engineering, Shivamogga for all their support and encouragement.

We are thankful for and fortunate enough to get constant encouragement, support, and guidance from all Teaching staff of the Department of Computer Science and Engineering which helped us in successfully completing our project work. Also, I would like to extend our sincere regards to all the non-teaching staff of the Department of Computer Science and Engineering for their timely support.

Thanking you all,

PROJECT ASSOCIATES

**SANATH GOUTHAM 4JN19CS089 PRATHVI A NADIG 4JN19CS071 SAMRUDH H 4JN19CS087**

**SHOAIB KALEEM SAYAD 4JN19CS090**

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **ABSTRACT** | **i** |
| **ACKNOWLEDGEMENT** | **ii** |
| **TABLE OF CONTENTS** | **iii** |
| **LIST OF FIGURES** | **iv** |
| **CHAPTER 1: INTRODUCTION** | **1-3** |
| 1.1 Overview of computer Graphics | 1 |
| 1.2 History | 1 |
| 1.3 Application of computer graphics | 1 |
| 1.4 User Interface | 3 |
| 1.5 Problem statement | 3 |
| 1.6 Objectives of the project | 3 |
| 1.7 Organization of the report | 3 |
| **CHAPTER 2: INTRODUCTION TO OPEN GL** | **4-9** |
| 2.1 Introduction | 4 |
| 2.2 OpenGL Command Syntax | 5 |
| 2.3 Pixel Operations | 5 |
| 2.4 Texture Assembly | 6 |
| 2.5 Rasterization | 6 |
| 2.6 Immediate Mode and Display Lists | 7 |
| 2.7 Advantages of using open GL | 9 |
| **CHAPTER 3: BASIC CONCEPTS AND WORKING PRINCIPLE** | **10** |
| 3.1 Rules and Working | 10 |
| 3.2 Game Instructions | 10 |
| **CHAPTER 4: DESIGN AND IMPLEMENTATION** | **11-19** |
| 4.1 Design | 11 |
| 4.2 Implementation | 11 |
| 4.3 System Requirement | 18 |
| 4.4 Flow Chart | 19 |
| **CHAPTER 5: RESULTS AND SNAPSHOTS** | **20-23** |
| **CHAPTER 6: CONCLUSION AND FUTURE ENHANCEMENT** | **24** |
| **REFERENCES** | **25** |

**LIST OF FIGURES**

**Fig No Fig Name Page no**

[Fig 2.1 Organization of the libraries for an X Window System environment. 5](#_Toc109396049)

Fig 2.2 Block diagram showing Rasterization 6

[Fig 4.1 Flow Chart 19](#_Toc109396050)

Fig 5.1 Snapshot showing the front sheet of project 20

[Fig 5.2: Snapshot showing the rules page 21](#_Toc109396051)

[Fig 5.3: Snapshots showing the levels 22](#_Toc109396052)

[Fig 5.4: Snapshots showing how loss is shown in the game 23](#_Toc109396053)

[Fig 5.5: Snapshot showing that the game has been cleared 23](#_Toc109396054)

**Chapter 1**

**INTRODUCTION**

* 1. **Overview of computer graphics**

Computer Graphics become a powerful tool for the rapid and economical production of pictures. There is virtually no area in which Graphical displays cannot be used to some advantage, so it is not surprising to find the use of CG so widespread.

Although early applications in engineering & science had to rely on expensive and cumbersome equipment, advances in computer technology have made interactive computer graphics a practical tool. Today people find Computer Graphics in a diverse area such as science, engineering, medicine, business, industry, government, art, entertainment, education, and training. Computer graphics as generalized tool for drawing and creating pictures and simulate the real-world situations within a small computer window.

* 1. **History**

William fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeing. One of the first displays of computer animation was future world (1976), which included an animation of a human face and was hand-produced by Carmel and Fred Parkle at the University of Utah.

There are several international conferences and journals where the most significant results in computer graphics are published. Among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on Graphics journals.

* 1. **Applications of computer graphics**

Nowadays Computer Graphics are used in almost all areas ranging from science, engineering, medicine, business, industry, government, art, entertainment, education, and training.

* + 1. **CG in the field of CAD**

Computer-Aided Design methods are routinely used in the design of buildings, Automobiles, aircraft, watercraft, spacecraft computers, textiles, and many other applications.

**1.3.1 CG in presentation Graphics**

Another major application area is presentation graphics used to produce illustrations for reports or generate slides. Presentation graphics are commonly used to summarize financial, statistical, mathematical, and scientific data for research reports and other types of reports.2D and 3D bar charts to illustrate some mathematical or statistical report.

* + 1. **CG in computer Art**

CG methods are widely used in both fine art and commercial art applications. Artists use a variety of computer methods including special-purpose hardware, artist’s paintbrush program (lumena), other pain packages, desktop packages, mathematics packages, and animation packages that provide facility for designing object motion. Ex: Cartoon design is an example of computer art that uses CG.

* + 1. **Entertainment**

Computer graphics methods are now commonly used in making motion pictures, music, videos, games, and sounds. Sometimes graphics objects are combined with the actors and live scenes.

* + 1. **Education and Training**

Computer-generated models of physical financial, and economic system is often educational aids. For some training applications, special systems are designed. Ex: the specialized system is a simulator for practice sessions or training of ship captains, aircraft pilots, and traffic control.

* + 1. **Image Processing**

Although the methods used in CG image processing overlap, the 2 areas are concerned with fundamentally different operations. In CG a computer is used to create a picture. Image processing on the other hand applies techniques to modify existing pictures such as photo scans, and TV scans.

* 1. **User interface**

It is common for software packages to provide a graphical interface. A major component of a graphical interface is a window manager that allows a user to display multiple window areas. The interface also displays menus, and icons for fast selection and processing.

* 1. **Problem statement**

Learning computer graphics as a part of academics and implementing it as a project on “ARCHERY GAME WITH A TWIST” in OpenGL which included all the basic library functions. An archery game with the twist where there are infinite invisible arrows and a moving platform which shoots them through which the target blocks must be destroyed.

* 1. **Objectives of the project**
* The main objective of this project is to show the “ARCHERY GAME WITH A TWIST” with the help of built-in graphics library functions.
* To design a simple game using Visual studio C++ Software and OpenGL Library.
* To explore and customize objects using the front panel.
* Illustrating the keyboard interaction.
* To provide entertainment to the user.
  1. **Organization of the report**

This section deals with the Introduction and organization of the project report. Chapter 2 discusses the basic concepts of OpenGL. Chapter 3 discusses the basic concept and working principle. Chapter 4 gives information about the design and implementation Chapter 5 includes results and snapshots. Chapter 6 gives the conclusion and future enhancement of the project.

**Chapter 2**

**INTRODUCTION TO OPENGL**

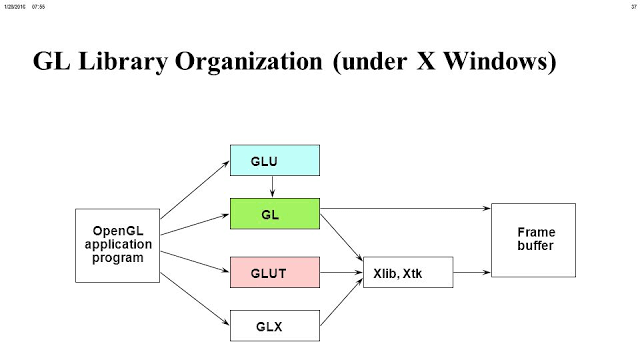
* 1. **Introduction**

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

Most of our applications will be designed to access OpenGL directly through functions in three libraries. Functions in the main GL (or OpenGL in windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in windows). The second is the OpenGL Utility Library (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with the letters glu.

To interface with the window system and to get input from external devices into our programs, the developer needs at least one more library. For each major window system, there is a system-specific library that provides the glu between the window system and OpenGL. For the X window system, this library is called GLX, for windows, it is wgl, and for the Macintosh, it is agl. Rather than using a different library for each system, here it uses a readily available library called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.

Fig 2.1 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.



# Fig 2.1 Organization of the libraries for an X Window System environment.

* 1. **OpenGL command syntax**

OpenGL commands use the prefix gl and initial capital letters for each word making up the command name. Similarly, OpenGL defined constants begin with GL\_, use all capital letters, and use underscores to separate words (like GL\_TRUE).

Some extraneous letters are appended to some command names (for example, the 3f in glColor3f() and glVertex3f()). It's true that the Color part of the command name glColor3f() is enough to define the command as one that sets the current color. However, more than one such command has been defined to use different types of arguments. In particular, the 3 part of the suffix indicates that three arguments are given; another version of the Color command takes four arguments. The f part of the suffix indicates that the arguments are floating-point numbers. Having different formats allows OpenGL to accept the user's data.

* 1. **Pixel operation**

Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next, the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the Rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory.

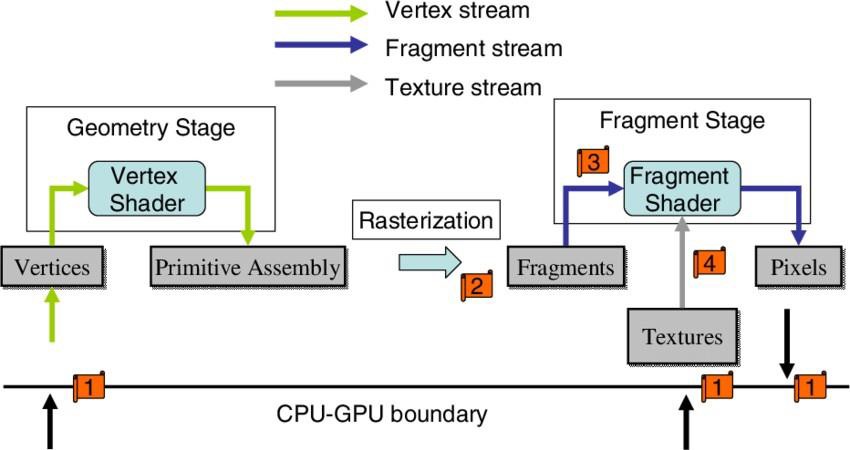
There are special pixel copy operations to copy data in the frame buffer to other parts of the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer.

* 1. **Texture assembly**

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory.

* 1. **Rasterization**

Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support initializing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.



**Figure 2.2 Block diagram showing Rasterization**

* 1. **Immediate mode and display lists**

All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

* **Transforming to Window Coordinates**

Before clip coordinates can be converted to window *coordinates*, they are normalized by dividing by the value of w to yield *normalized device coordinates*. After that, the viewport transformation applied to these normalized coordinates produces window coordinates. One can control the viewport, which determines the area of the on-screen window that displays an image, with glDepthRange () and glViewport ().

* **Matrix Transformations**

Vertices and normals are transformed by the model-view and projection matrices before they're used to produce an image in the frame buffer. You can use commands such as glMatrixMode (), glMultMatrix (), glRotate (), glTranslate (), and glScale () to compose the desired transformations, or you can directly specify matrices with glLoadMatrix () and glLoadIdentity (). Use glPushMatrix () and glPopMatrix () to save and restore model view and projection matrices on their respective stacks.

The basic model for OpenGL command interpretation is the immediate mode, in which a command is executed as soon as the server receives it; vertex processing, for example, may begin even before the specification of the primitive of which it is a part has been completed. Immediate mode execution is well-suited to interactive applications in which primitives and modes are constantly altered. In OpenGL, the fine-grained control provided by immediate mode is taken as far as possible: even individual lighting parameters (the diffuse reflectance color of a material, for instance) and texture images are set with individual commands that have an immediate effect. While immediate mode provides flexibility, its use can be inefficient if unchanging parameters or objects must be re-specified. To accommodate such situations, OpenGL provides display lists. A display list encapsulates a sequence of OpenGL commands (all but a handful of OpenGL commands may be placed in a display list) and is stored on the server. The display list is given a numeric name by the application when it is specified; the application need only name the display list to cause the server to effectively execute all the commands contained within the list. This mechanism provides a straightforward, effective means for an application to transmit a group of commands to the server just once even when those same commands must be executed many times.

* + 1. **Display List Optimization**

Accumulating commands into a group for repeated execution presents possibilities for optimization. Consider, for example, specifying a texture image. Texture images are often large, requiring a large, and therefore possibly slow, data transfer from client to server (or from the server to its graphics subsystem) whenever the image is re-specified. For this reason, some graphics subsystems are equipped with sufficient storage to hold several texture images simultaneously. If the texture image definition is placed in a display list, then the server may be able to load that image just once when it is specified. When the display list is invoked (or re-invoked), the server simply indicates to the graphics subsystem that it should use the texture image already present in its memory, thus avoiding the overhead of re-specifying the entire image. Examples like this one indicate that display list optimization is required to achieve the best performance. In the case of texture image loading, the server is expected to recognize that a display list contains texture image information and to use that information appropriately. This expectation places a burden on the OpenGL implementer to make sure that special display list cases are treated as efficiently as possible. It also places a burden on the application writer to know to use display lists in cases were doing so could improve performance. Another possibility would have been to introduce special commands for functions that can be poor performers in immediate mode. But such specialization would clutter the API and blur the clear distinction between immediate mode and display lists.

* 1. **Advantages of using OpenGL**
* Industry standard: An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.
* Stable: OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.
* Reliable and portable: All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.
* Evolving: Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

**Chapter 3**

**BASIC CONCEPT AND WORKING PRINCIPLE**

* 1. **Rules and Working**

This is an archery game built using OPEN GL. When the program is executed the front sheet of the project is displayed. Then the player must press the ‘t’ button to start the game. The game screen is displayed where initially the rules are displayed through which the user understands the game and then ‘m’ needs to pressed to start the game when the game starts the user needs to clear three levels in order to complete the game throughout which he has infinite invisible arrows but a limited movement.

* 1. **Game Instructions**
     + The player should press the ‘m’ key to start the game.
     + There are 3 Levels in the game.
     + The player should destroy the targets in order to win the levels.
     + The player should press 'w' to shoot the arrow which is invisible.
     + The player should press 'a' to move the platform left.
     + The player should press 'd' to move the platform right.
     + There is a restriction on the number of platform movements that can be performed.
     + The targets must be destroyed within the total number of movements provided.
     + Every level has a different type of target design.
     + There are twelve positions to which the platform can move to.

**Chapter 4**

* 1. **Design**

**DESIGN AND IMPLEMENTATION**

The whole program has been implemented in the C language. The bottom line of the design is keyboard interaction, and some minor functions are used to print the text on the screen.

* 1. **Implementation**

The OpenGL provides very powerful translation facilities which relive the programmers by allowing them to concentrate on their jobs rather than focusing on how to implement these operations

* + 1. **Built-in functions**
       - **glutCreateWindow (char\* name)**

The glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name. Implicitly, the *current window* is set to the newly-created window. Each created window has a unique associated OpenGL context. *name* - ASCII character string for use as window name.

* + - * **glutDestroyWindow (int win)**

This function destroys the specified window. glutDestroyWindow destroys the window specified by the win and the window's associated OpenGL context, logical colormap (if the window is color index), and overlay and related state (if an overlay has been established). Any sub windows of destroyed windows are also destroyed by glutDestroyWindow. If the win was the current window, the current window becomes invalid (glutGetWindow will return zero).

*win –* identifies the GLUT window to destroy.

* + - * **glutInit (int argc, char\*\* argv)**

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program

with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command-line options. glutInit also processes command-line options, but the specific options parse is window system dependent. *argc -* A pointer to the program's *unmodified* argc variable from main. Upon return, the value pointed to by argc will be updated, because glutInit extracts any command line options intended for the GLUT library.

argv -The program's *unmodified* argv variable from main. Like argc, the data for argv will be updated because glutInit extracts any command-line options understood by the GLUT library.

* + - * **glutInitDisplayMode (unsigned int mode)**

glutInitDisplayMode sets the *initial display mode*. The *initial display mode* is used when creating top-level windows, sub windows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

*Mode* - Display mode, normally the bitwise *OR*-ing of GLUT display mode bit masks. It can have values like the following:

GLUT\_SINGLE - Bit mask to select a single buffered window. GLUT\_DOUBLE - Bit mask to select a double-buffered window. GLUT\_DEPTH - Bit mask to select a window with a depth buffer. GLUT\_RGB - Bit mask to select an RGB mode window.

GLUT\_INDEX - Bit mask to select a color index mode window. GLUT\_STEREO - Bit mask to select a stereo window.

* + - * **glClearColor()**

glClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range [0,1]. It can have parameters like red, green, blue, and alpha. Specify the red, green, blue, and alpha values used when the color buffers are cleared. The initial values are all 0.

* + - * **glutDisplayFunc(void (\*func)(void))**

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 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.

*func -* The new display callback function.

* + - * **glutPostRedisplay(void)**

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.

* + - * **glutKeyboardFunc(void (\*func)(unsigned char key, int x, int y))** 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 keystrokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

*func* – The new keyboard callback function.

* + - * **glutMainLoop(void)**

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

* + - * **glutFullScreen(void)**

glutFullScreen requests that the *current window* be made full screen. The exact semantics of what full screen means may vary by window system. The intent is to make the window as large as possible and disable any window decorations or borders added the window system. The window width and height are not guaranteed to be the same as the screen width and height, but that is the intent of making a window full screen. glutFullScreen is defined to work only on top-level windows.

* + - * **glClear(GLbitfield mask)**

glClear sets the bitplane area of the window to values previously selected by glClearColor, glClearIndex, glClearDepth, glClearStencil, and glClearAccum. Multiple color buffers can be cleared simultaneously by selecting more than one buffer at a time using [glDrawBuffer.](https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glDrawBuffer.xml) glClear takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.

*mask* - Bitwise OR of masks that indicate the buffers to be cleared. The four masks are GL\_COLOR\_BUFFER\_BIT, GL\_DEPTH\_BUFFER\_BIT, GL\_ACCUM\_BU FFER\_BIT, and GL\_STENCIL\_BUFFER\_BIT.

The values for glClear are:

GL\_COLOR\_BUFFER\_BIT - Indicates the buffers currently enabled for color writing.

GL\_DEPTH\_BUFFER\_BIT - Indicates the depth buffer. GL\_ACCUM\_BUFFER\_BIT - Indicates the accumulation buffer. GL\_STENCIL\_BUFFER\_BIT - Indicates the stencil buffer.

* + - * **glShadeModel(GLenum mode)**

glShadeModel 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. In either case, the computed color of a vertex is the result of lighting if the lighting is enabled, or it is the current color at the time the vertex was specified if the lighting is disabled.

*mode -* Specifies a symbolic value representing a shading technique. Accepted values are GL\_FLAT and GL\_SMOOTH. The initial value is GL\_SMOOTH.

* + - * **glOrtho()**

glOrtho multiplies the current matrix with an orthographic matrix. glOrtho describes a transformation that produces a parallel projection. The current matrix is multiplied by this matrix and the result replaces the current matrix. It can have parameters like:

*left, right -* Specify the coordinates for the left and right vertical clipping planes. *bottom, top -*Specify the coordinates for the bottom and top horizontal clipping planes. *nearVal, farVal -* Specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

* + - * **glSwapBuffers(void)**

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 have no effect.

* + - * **glFlush(void)**

This function force execution of GL commands in finite time. Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time, it does complete in finite time. Because any GL program might be executed over a network, or on an accelerator that buffers commands, all programs should call glFlush whenever they count on having all their previously issued commands

completed. For example, call glFlush before waiting for user input that depends on the generated image.

* + - * **glPushMatrix(void)**

glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical to the one below it.

* + - * **glPopMatrix(void)**

[glPopMatrix](https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glPopMatrix.xml) pops the current matrix stack, replacing the current matrix with the one below it on the stack.

* + - * **glutMainLoop(void)**

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any call-backs that have been registered.

* + - * **glutInitWindowPosition(int x, int y)**

glutInitWindowPosition set the initial window position. The initial value of the initial window position GLUT state is -1 and -1. If either the X or Y component to the initial window position is negative, the actual window position is left to the window system to determine. The intent of the initial window position values is to provide a suggestion to the window system for a window's initial position. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specified position. A GLUT program should use the window's reshape callback to determine the true size of the window.

*x –* Window X location in pixels.

*y –* Window Y location in pixels.

* + - * **glutInitWindowSize(int width, int height)**

The initial value of the initial window size GLUT state is 300 by 300. The initial window size components must be greater than zero. The intent of the initial window position values is to provide a suggestion to the window system for a window's initial

size. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specified size. A GLUT program should use the window's reshape callback to determine the true size of the window.

*width* – Width in pixels.

*height* – Height in pixels.

* + - * **glRectf(x1, y1, x2, y2)**

glRect supports efficient specification of rectangles as two corner points. Each rectangle command takes four arguments, organized either as two consecutive pairs of x y coordinates or as two pointers to arrays, each containing an x y pair. The resulting rectangle is defined in the z = 0 plane.

*x1, y1 -* Specify one vertex of a rectangle.

*x2, y2 -* Specify the opposite vertex of a rectangle.

* + 1. **User-define function**
       - **void target()**

This function is used to render the targets present in the game according to the level the user is playing it is also made so that after the user has destroyed the specific target it is not to be displayed the function uses ‘countXX’ of integer type to check if the target blocks need to displayed or not.

* + - * **void frontsheet()**

This function is used to write the contents of the front sheet. glClearColor function is used to give the color to the front sheet. glColor3f is used to give color to the strings.

* + - * **void platform(int PlatformPosition)**

This function is used to draw the platform and control its movement accordingly it is written in a way such that when the keys to move the platform(‘a’ or ‘d’) are clicked, it moves the platform from the present location to required location

* + - * **void platform()**

This function is used to check if the game has been won. It uses ‘winX’ a variable of integer type to check if all the targets have been destroyed under the given moves and it they have it displays that the level has been cleared.

* + - * **void loss()**

This function is used to check if the user overuses the movement chance(moves) if so then this function displays the ‘You lost level’ message

* + - * **void keys(unsigned char key, int x, int y)**

This function is used to assign keys to the different operations such as increment of vary the value of a variable.

‘t’ is used to move from front page to actual game

‘m’ is used to start the game and move to level 1

‘n’ is assigned to level 2

‘b’ is assigned to level 3

‘a’ and ‘d’ are used to move the platform

‘w’ is used to fire the invisible arrow

* + - * **void alwaysDisplay()**

This function is used to display the rules at the beginning of the game and then display the total moves and which level the user is playing and draw the border to the game

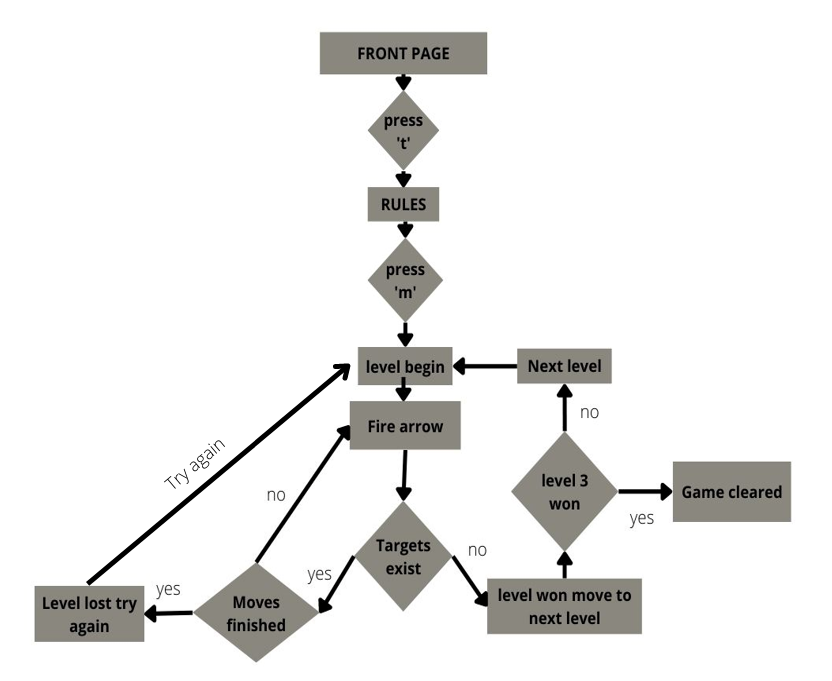
* + - * **void print\_text(int x, int y, char str[], float r, float g, float b)**
      * **void print\_text1(int x, int y, char str[], float r, float g, float b)**

These functions are used to display the front page onto the screen.

* 1. **System requirements**
     1. **Minimum Software Requirements**

1. Operating system: Windows XP/ Windows7/ Windows 8
2. Compiler Used: Visual C++ 6.0
3. Language Used: C language
   * 1. **Minimum Hardware Requirements**
4. Main processor: PENTIUM III
5. Processor Speed: 800 MHz
6. RAM Size: 128 MB DDR
   1. **Flow Chart**

The following diagram shows the flow chart of the project. It shows the flow of control throughout the program. By seeing this flow chart users can easily analyze the working of the archery game.

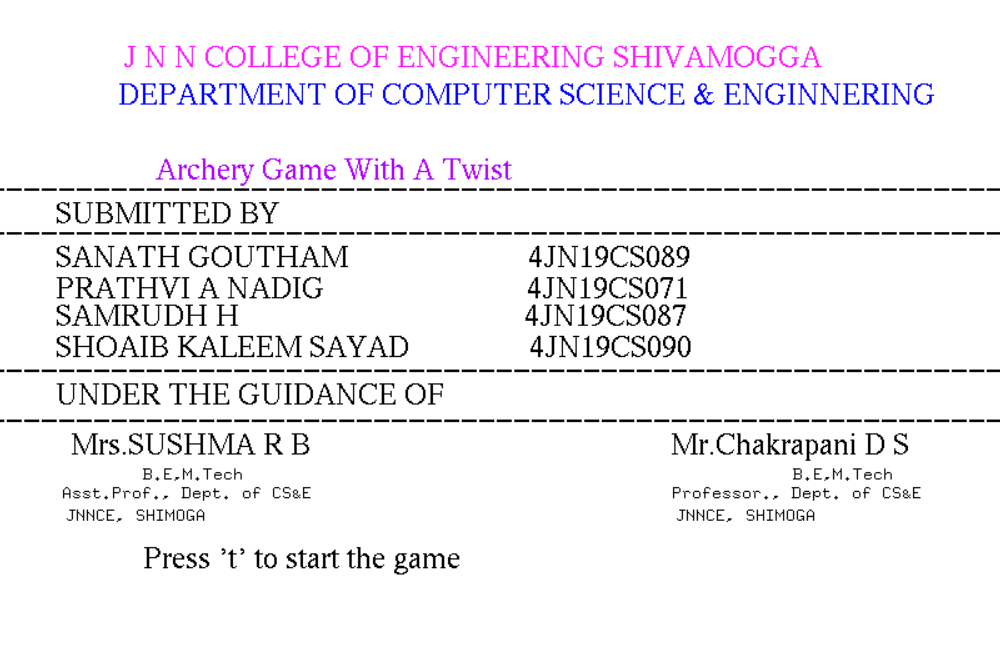


# Fig 4.1 Flow Chart

**Chapter 5**

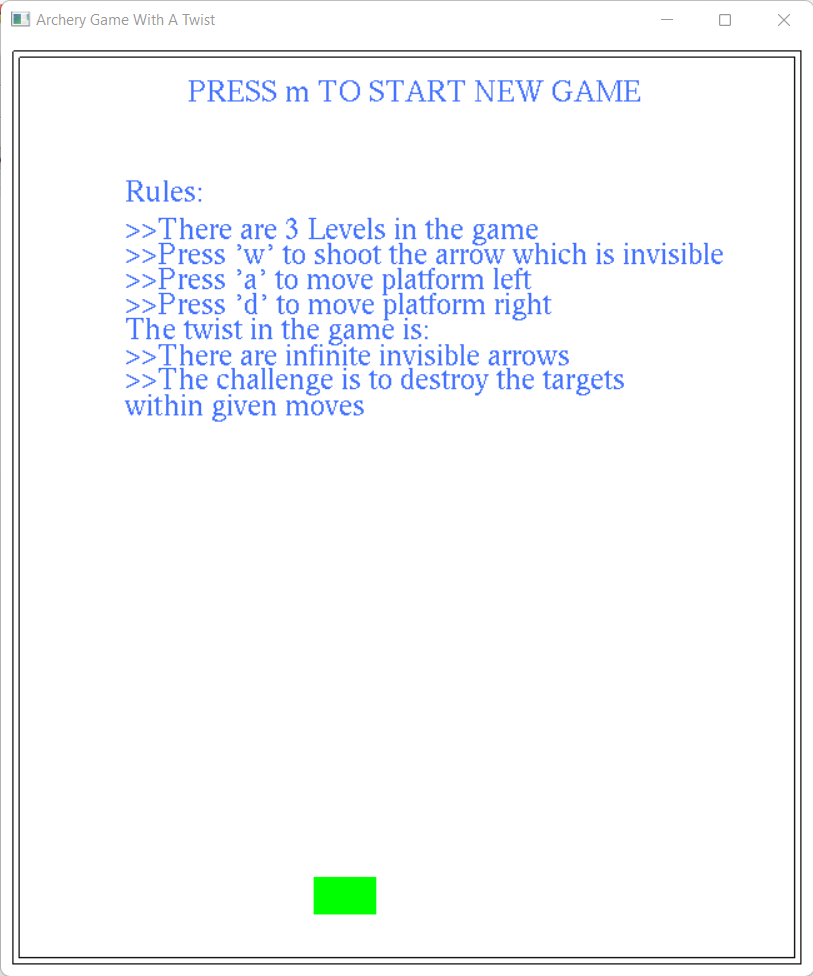
**RESULTS AND SNAPSHOTS**

This chapter includes the screenshots of the project executed. It contains the front sheet, game execution, and the game over screen.

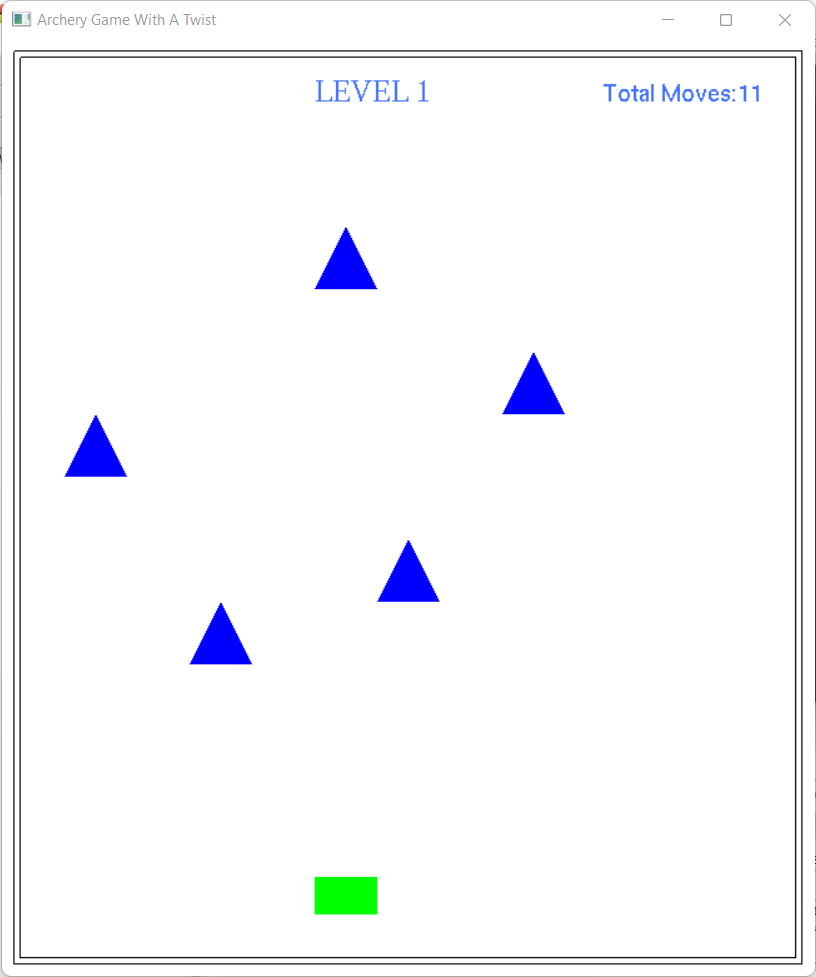


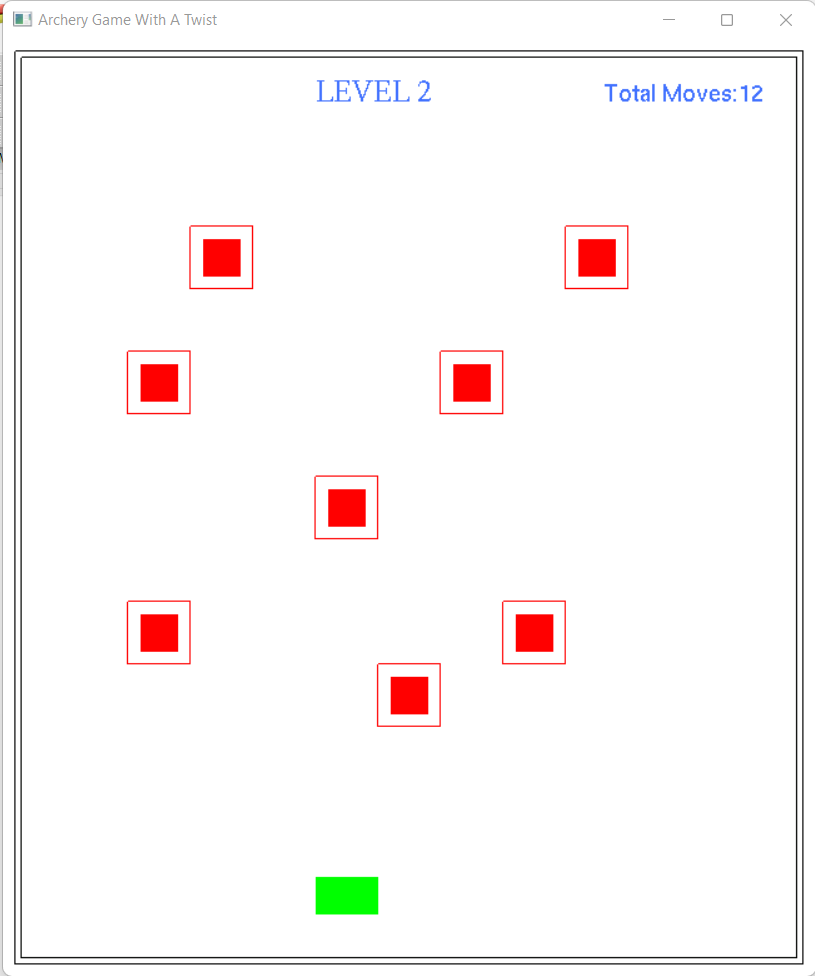
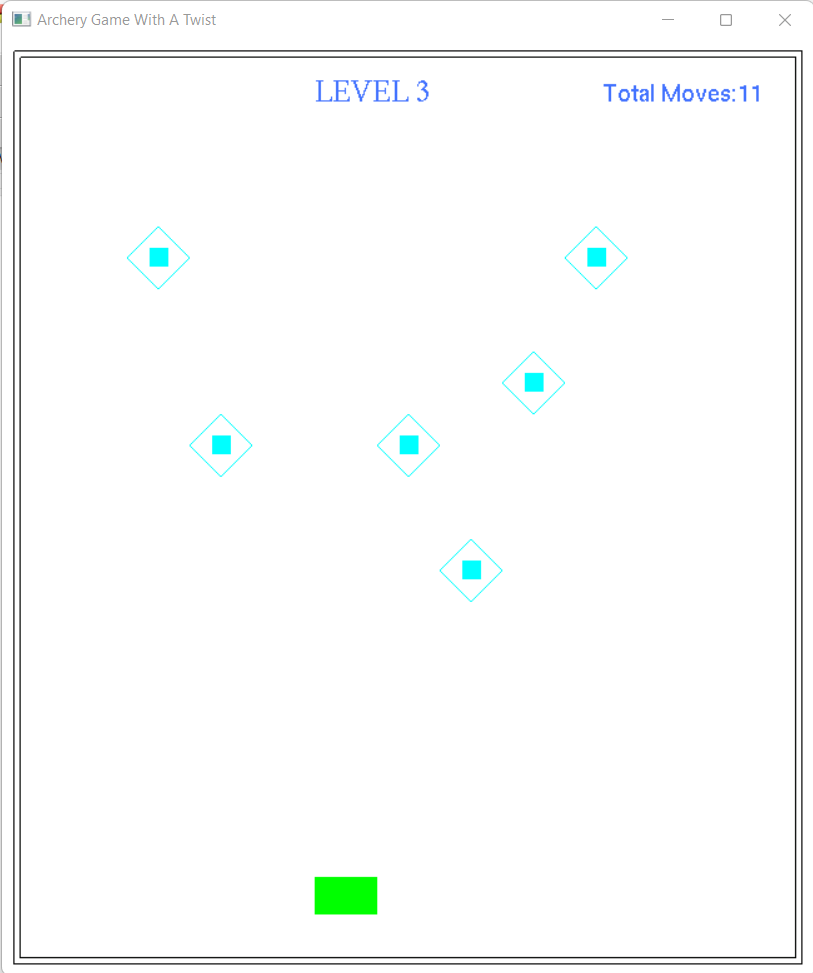
**Fig 5.1: Snapshot showing Front Sheet of the project**

The above snapshot shows front sheet of the project.



# Fig 5.2: Snapshot showing the rules page

 The above snapshot shows the rules of the game.



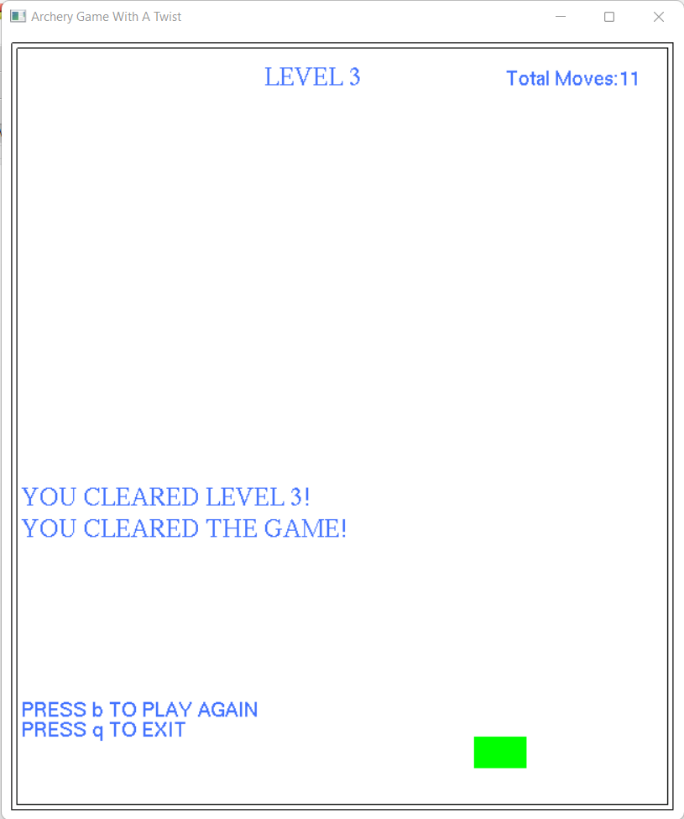
# 

# Fig 5.3: Snapshots showing the levels

The above snapshots show the different levels of the game.



# Fig 5.4: Snapshots showing how loss is shown in the game

 The above snapshot shows the display when a level is lost.

# Fig 5.5: Snapshot showing that the game has been cleared

The above snapshot specifies that the game has been cleared by winning all three levels of the game.

**Chapter 6**

**CONCLUSION AND FUTURE ENHANCEMENT**

This project aims in designing a simple “ARCHERY GAME WITH A TWIST” using OpenGL. The main objective of this project is to show the functions of OpenGL with the help of built-in graphics library functions. The code has been written in OpenGL. OpenGL supports enormous flexibility in the design and the use of OpenGL graphics programs. The code contains the function that is necessary to accomplish all the tasks required for this project. It works well with the Windows platform. The presence of many built-in functions takes care of much functionality and reduces the job of coding as well as makes the implementation simpler. The project started with the designing phase in which we listed the requirements needed, the game design, then the detailed designing of each function for the proper working of the game, the next stage is the testing and debugging stage.

It can be further improved to provide better facilities such as shading and lighting effects and can be developed using 3D effects. In the future the following enhancements can be added:

* More levels to the game.
* Instead of total moves, moves left can be displayed.
* Targets can be redesigned for visual enhancement.
* New options such as right-clicking the mouse for the rules page.
* A more detailed menu.
* Numbers of arrows consumed can be displayed at the end.
* Multi-player feature can be added to this game.

**REFERENCES**

* Edward Angel**,** Interactive Computer Graphics,5th Edition, Addison-Wesley, 2008.
* Donald Hearn and Pauline Baker: Computer Graphics- OpenGL Version,3rd Edition, Pearson Education,2011.
* [http://www.opengl.org/documentation/books.](http://www.opengl.org/documentation/books)