VISVESVARAYA TECHNOLOGICAL UNIVERSITY

A MINI PROJECT REPORT

ON

2D TRAVELLER GAME

Submitted in partial fulfillment of the requirements

For the award of degree of

Bachelor of Engineering

In

Computer Science and Engineering

By

Nikhil N. Prasad [1KS13CS066]

Under the guidance of

Mrs. Sougandhika Narayan

Asst. Prof, CS&E Department

Mrs. Deepa S.R.

Asst. Prof, CS&E Department

Department of Computer Science & Engineering

K.S. INSTITUTE OF TECHNOLOGY

#14, Raghuvanahalli, Kanakapura Main Road, Bangalore-560109.

K.S. Institute of Technology

#14, Raghuvanahalli, Kanakapura Main Road, Bangalore-560109.

Department of Computer Science & Engineering

CERTIFICATE

This is to certify that mini project work entitled **"2D TRAVELLER GAME"** carried out by **Mr. Nikhil N. Prasad** bearing USN **1KS13CS066** bonafide student of **K.S. Institute of Technology** in the partial fulfillment for the award of the **Bachelor of Engineering** in **Computer Science & Engineering** of the **Visvesvaraya Technological University**, Belgaum, during the year 2016. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of mini Project work prescribed for the said degree.

Mrs. Rekha B. Venkatapur
Prof. & HOD, CS&E Department

Dr. T.V. Govindaraju
Principal, KSIT

Mrs. Sougandhika Narayan
Asst. Prof, CS&E Department

Mrs. Deepa S.R.
Asst. Prof, CS&E Department

Name of the Examiners **Signature with date**

1.

2.

ACKNOWLEDGEMENT

I consider it a privilege to whole-heartedly express my gratitude and respect to each and everyone who guided and helped us in the successful completion of this project.

First and foremost, I would like to thank **K. S. Institute of Technology** for providing us an opportunity to work on the project. I also would like to thank the management for providing all the resources required for the project.

I wish to acknowledge my sincere gratitude to our beloved principal, **Dr. T.V. Govindaraju** for his encouragement and providing all facilities for the accomplishment of this project.

The project would not have been possible without the support of our beloved **Rekha B. Venkatapur**, Professor & HOD, Department of Computer Science and Engineering.

I would like to express my immense gratitude towards **Mrs. Sougandhika Narayan**, Asst. Prof., CS&E Department, **Mrs. Deepa S.R.**, Asst. Prof., CS&E Department, and **Mr. Pradeep Kumar G.H.**, Asst. Prof., CS&E Department, under whose guidance I successfully completed my project. They have been very generous is assisting and supporting me to do this project, **"2D TRAVELLER GAME"**.

Finally, I would like to thank all the other teaching and non-teaching staff members who have extended their support and kind co-operation while bringing up this project.

Nikhil N. Prasad

(1KS13CS066)

ABSTRACT

OpenGL remains the standard programmer's interface both for writing application programs and developing high-level products for multi-platform applications. OpenGL supports applications ranging from large scientific visualizations to user interfaces on cellular phones.

The aim in developing this program was to design an interactive game, **2D TRAVELLER**. The project has been implemented under the Linux Environment using a standard developer OpenGL package. The illustration of OpenGL features are included and the application program is efficiently developed.

The **2D TRAVELLER GAME** can be played either by using the keyboard, or by using the mouse buttons. The objective of the game is to reach the goal without encountering any obstacles, where the user will be notified of his victory or defeat appropriately. Using the various options provided, or by the mouse menus, the user can increase or decrease the difficulty of the game, further enhancing the gaming experience.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Perhaps the dominant characteristic of this new millennium is how computers and communication technologies have become dominant forces in our lives. Activities as wide ranging as film-making, publishing, banking, and education continue to undergo revolutionary changes as these technologies alter the ways in which we conduct our daily activities. The combination of computers, networks, and the complex human visual system, through computer graphics, has led to new ways of displaying information, seeing virtual worlds, and communicating with people and machines.

1.1 Introduction to Computer Graphics

Computer Graphics is one of the most powerful and interesting facets of computers. There is a lot we can do in graphics apart from drawing figures of various shapes. All video games, animation, multimedia predominantly works using computer graphics. There are so many applications of computer graphics, which make it significant.

Computer graphics is concerned with all aspects of producing pictures or images using a computer. The field began humbly almost 50 years ago, with the display of a few lines on a cathode-ray tube (CRT). Now, we can create images on a computer that are indistinguishable from photographs of real objects. We routinely train pilots with simulated airplanes, generating graphical displays of a virtual environment in real time. Feature-length movies made entirely using computers have been successful, both critically and financially. Massive multiplayer games can involve tens of thousands of concurrent participants.

Figure 1.1 shows the application programmers model of a graphics system.

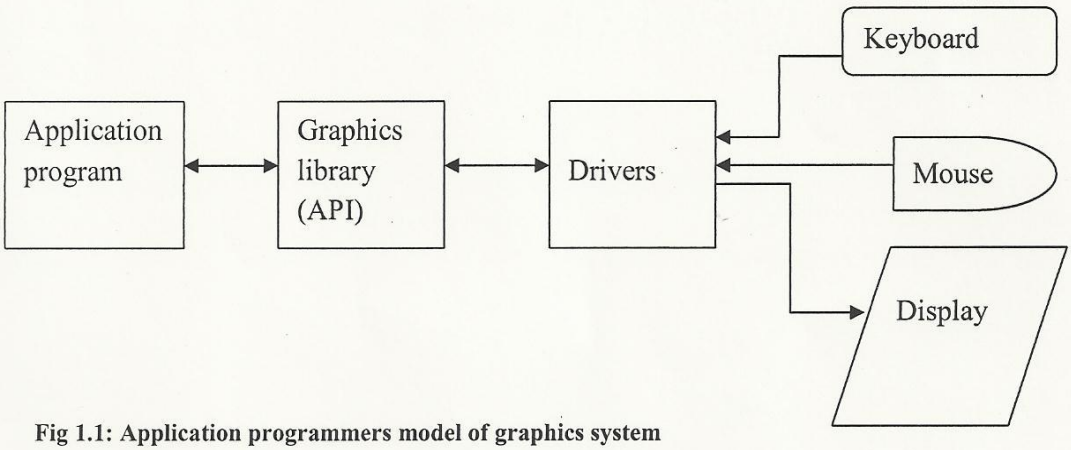

Fig 1.1: Application programmers model of graphics system

It includes the creation, storage, and manipulation of models and images of objects. These models include physical, mathematical, engineering, architectural and so on. Computer graphics today is largely interactive - the user controls the contents, structure, and appearance of objects and their displayed images by using input devices, such as a keyboard, mouse of touch-sensitive panel on the screen. The interface between an application program and a graphics system can be specified through a set of functions that reside in the graphics library. These specifications are called application programmer's interface (API). The software drivers are responsible for interpreting the output of the API and converting this data to a form that is understood by the particular hardware.

1.2 Introduction to OpenGL

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 (**Open G**raphics **L**ibrary) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional 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 Direct3Don Microsoft Windows platforms. OpenGL is managed by the non-profit technology consortium, The Khronos Group.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, the user must work through whatever windowing system controls the particular hardware that the user is using. Similarly, OpenGL does not provide high-level commands for describing models of three-dimensional objects. Such commands might allow the user to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadratic surfaces and NURBS curves and surfaces.

OpenGL serves two main purposes:

- To hide the complexities of interfacing with different 3D accelerators, by presenting the 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 (using software emulation if necessary).

1.2.1 The OpenGL Architecture

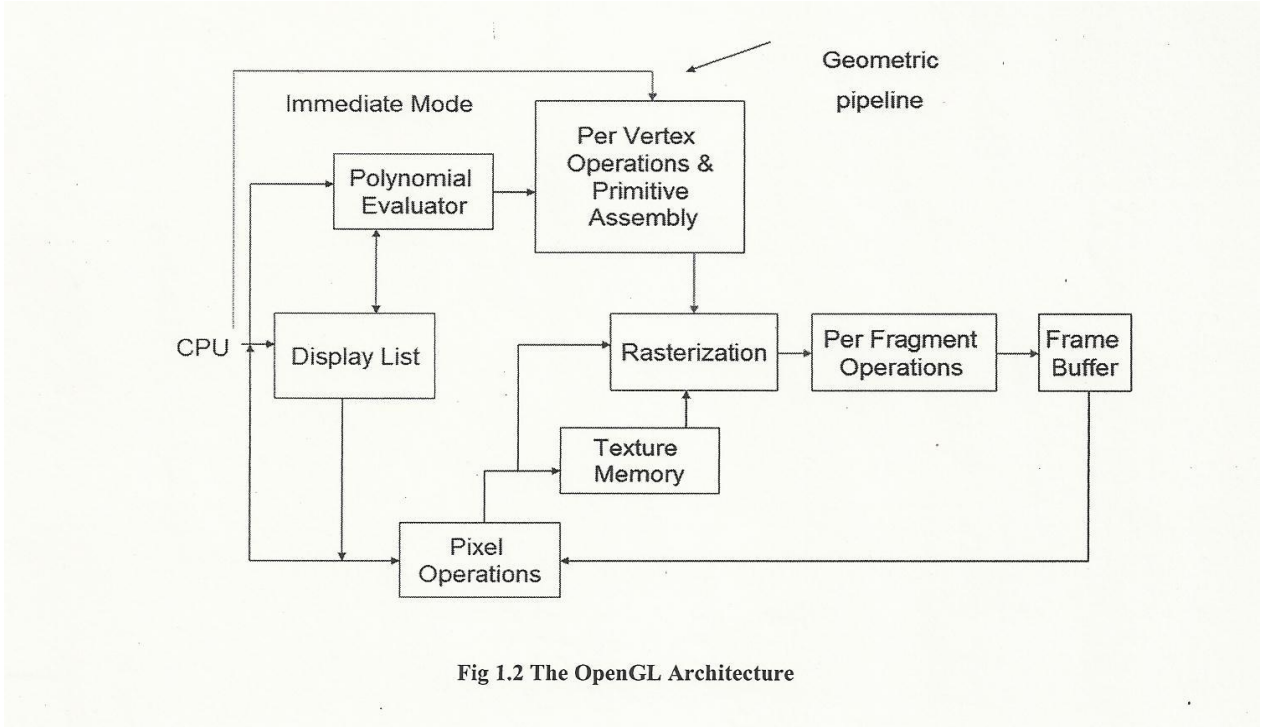

Fig 1.2 The OpenGL Architecture

The following list briefly describes the major graphics operations of the graphics pipeline which OpenGL performs to render an image on the screen.

- Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects (OpenGL considers points, lines, polygons, images, and bitmaps to be primitives).
- Arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene.
- Calculate the color of all the objects. The color might be explicitly assigned by the application, determined from specified lighting conditions, obtained by pasting a texture onto the objects, or some combination of these three actions.
- Convert the mathematical description of objects and their associated color information to pixels on the screen. This process is called *rasterization.*

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. Given below are the key stages in the OpenGL rendering pipeline.

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.

Evaluators

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface from the control points.

Per-Vertex Operations

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on the screen.

Primitive Assembly

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped. The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

Pixel Operations

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. 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.

Texture Assembly

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic.

Rasterization

Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the frame buffer. Color and depth values are assigned for each fragment square.

Fragment Operations

Before values are actually stored into the frame buffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

1.2.2 The OpenGL Library

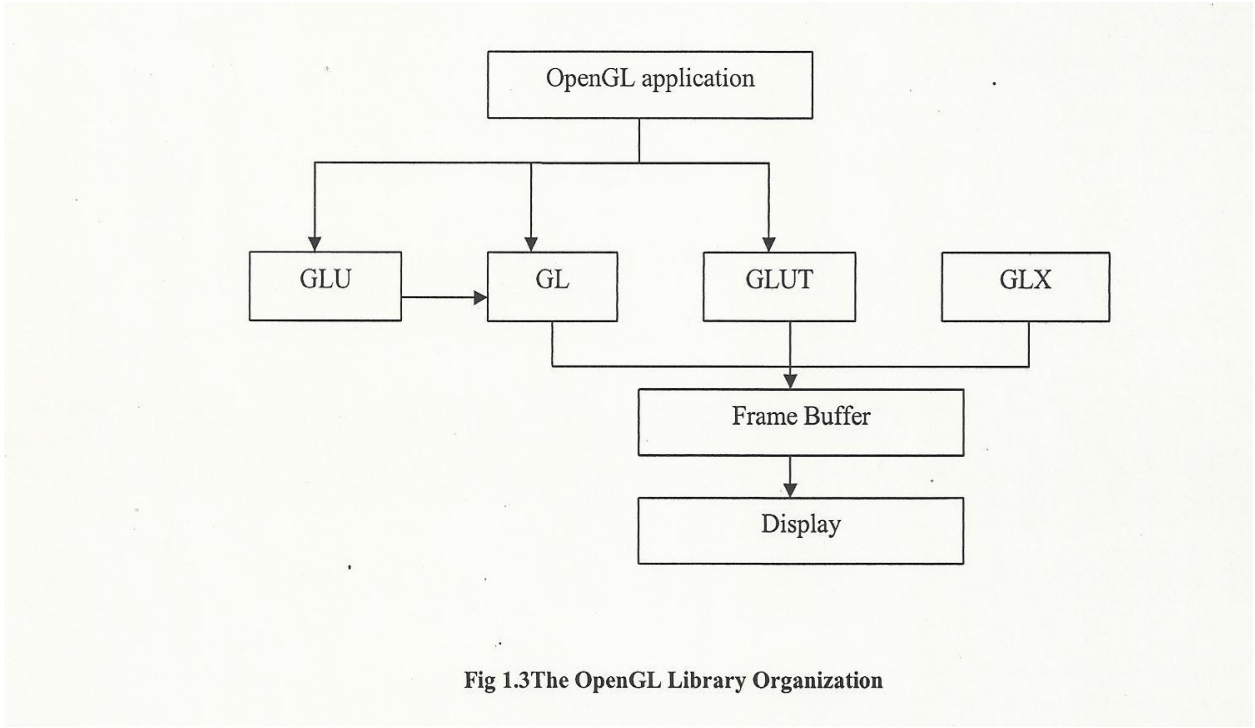

Fig 1.3The OpenGL Library Organization

A sophisticated library that provides these features could certainly be built on top of OpenGL.

- The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces. GLU is a standard part of every OpenGL implementation.

- Also, there is a higher-level, object-oriented toolkit, Open Inventor, which is built atop OpenGL, and is available separately for many implementations of OpenGL.

- The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit, to hide the complexities of differing window system APIs.

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow the user to simplify the programming tasks, including the following:

The OpenGL Utility Toolkit (GLUT) is a programming interface with ANSI C and FORTRAN bindings for writing window system independent OpenGL programs. The toolkit supports the following functionality:

- Multiple windows for OpenGL rendering.
- Callback driven event processing.
- Sophisticated input devices.
- An “idle” routine and timers.
- A simple, cascading pop-up menu facility.
- Utility routines to generate various solid and wire frame objects.
- Support for bitmap and stroke fonts.

Miscellaneous window management functions, including managing overlays.

1.2.3 Parts of OpenGL

- *Initialization*: initializeGL()
  - Enable/Disable OpenGL features.
  - Configure OpenGL state variables.
- *Resizing*: resizeGL()
  - Resize OpenGL viewport.
  - Configure view/camera matrix.
- *Painting*: paintGL()
  - Clear the buffers (frame buffer, depth buffer, etc.)
  - Render the scene.

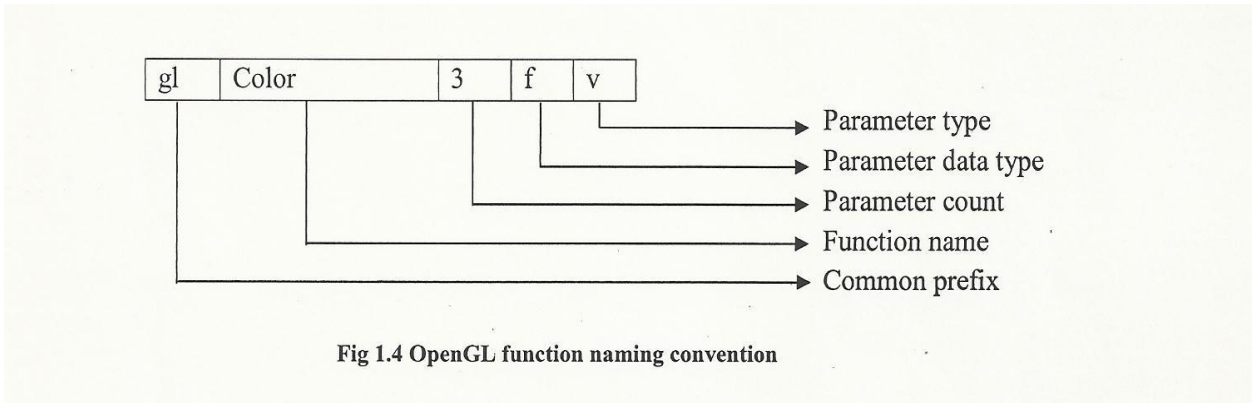

Fig 1.4 OpenGL function naming convention

1.2.4 OpenGL Data Types

Suffix	Data Type	Typical C/C++ Type	OpenGL Type Name
B	8-bit integer	signed char	GLbyte
S	16-bit integer	short	GLshort
I	32-bit integer	int or long	GLint, GLsize
F	64-bit integer	float	GLdouble
D	64-bit floating point	double	GLdouble, GLclamped
Ub	8-bit unsigned number	unsigned char	GLubyte, GLboolean
Us	16-bit unsigned number	unsigned short	GLushort
Ui	32-bit unsigned number	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

Table 1.1 Command Suffixes and Argument Data Types

1.2.5 Advantages of 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, multi-platform 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.

- *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.

- *Scalable*: OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

- *Easy to use*: OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer form having to design for specific hardware features.

- *Well-documented*: Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

1.2.6 Applications of OpenGL

The development of computer graphics has been driven both by the user community and by advances in hardware and software. The applications of computer graphics is many and varied. We can, however, divide them into four major areas:

- Display of Information
- Simulation and Animation
- Design
- User Interfaces

Chapter 2

REQUIREMENT ANALYSIS

The requirement analysis specifies the requirements needed to develop a graphic project. 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* that describe all the interactions the users will have with the software. Use cases are also known as *functional requirements*. In addition to use cases, the SRS also contains non-functional (or supplementary) requirements. *Non-functional requirements* are requirements which impose constraints on the design or implementation (such as *performance engineering requirements, quality standards* or *design constraints*).

2.1 Requirements of the project

A graphics package that attracts the attention of the viewers is to be implemented. The package should provide a platform for user to perform animation.

2.2 Resource Requirements

The requirement analysis phase of the project can be classified into:

- Hardware Requirements
- Software Requirements
- Functional Requirements
- Non-Functional Requirements

2.2.1 Software Requirements

This document will outline the software design and specification of our application. The application is a Linux based implementation with OpenGL. Our main UI will be developed in C, which is a high level language.

Characteristics required of the interface between the software product and each of the hardware components:

- Mouse interface allows the user to control the functions of the game.
- Keyboard interface to play the game.

Interface with other software components or products, including other systems, utility software, databases, and operating systems.
OpenGL libraries for required are:

- GLUT library
- STDLIB library

The application is very self-contained. Robust error handling will be implemented and code will be object-oriented to allow for easier maintenance and feature additions.

2.2.2 Hardware Requirements

There are no rigorous restrictions on the machine configuration. The model should be capable of working on all machines capable of supporting recent versions of any Linux-based operating system (Ubuntu 14.04.4 LTS, Fedora 23 Workstation, etc.)

- Processor: Intel® Pentium 4 CPU
- Hard disk Capacity: 80 GB
- RAM: 1 GB

- CPU Speed: 2.9 GHz
- Keyboard: Standard
- Mouse: Standard

Some specific hardware requirements are needed for installing the Linux operating system. They can be listed as:

Requirement	Professional
Processor	700 MHz processor (about Intel Celeron or better)
RAM	512 MiB RAM or higher (system memory)
Available Hard Disk Space	5 GB of hard-drive space (or USB stick, memory card or external drive but see LiveCD for an alternative approach)
Video	VGA capable of 1024x768 screen resolution.
CD/DVD Drive	Either a CD/DVD drive or a USB port for the installer media.
Mouse	Microsoft mouse or compatible pointing device.
Keyboard	Standard.

Table 2.1 Minimum Hardware Requirements Specification

Performance has not been tuned for minimum system configuration. Increasing the RAM above the recommended system configuration will improve performance, specifically

when there are running multiple applications, working with large projects, or doing enterprise-level development.

2.3 Functional requirements

A **functional requirement** defines a function of a system and its components. A function is described as a set of inputs, the behavior, and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. The functional requirements used in the project are

- Mouse function
- Keyboard function

2.4 Non-functional requirements

A **non-functional requirement** is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. Non-functional requirements define how a system is supposed to be. It essentially specifies how the system should behave and that it is a constraint upon the systems behavior.

- **Reliability** describes the ability of a system or component to function under stated conditions for a specified period of time. Reliability is defined as the probability of success as the frequency of failures.
- **Maintainability** is the ease with which a product can be maintained in order to isolate defects, correct defects or their cause, maximize a product's useful life, maximize efficiency, reliability, and safety.

- **Availability** is the probability that a system, at a point in time, will be operational and able to deliver the requested services. It is typically measured as a factor of its reliability - as reliability increases, so does availability.

CHAPTER 3

DESIGN

3.1 Overview

Design is the second stage in the software life cycle of any engineered product or system. Design is a creative process. A good design is the key to effective system. It may be defined as “the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization.”

The design specification describes the features of the system, the components or elements of the system and their appearance to the end users. In system design, high-end decisions are taken regarding the basic system architectures, platforms and tools to be used. The system design transforms a logical representation of what a given system is required into the physical specification. The significant design factors to be considered are reliability, response time, throughput of the system, maintainability, expandability etc.

Design is the place where quality is fostered in software development. During design, decisions are made that ultimately affect the success of the software construction. Software design is an iterative process through which the requirements are translated into a “blue print” for constructing the software. Software design serves as the foundation for all software engineering and software maintenance steps that follow.

3.2 Algorithm

The entire design process can be explained using a simple algorithm. The algorithm gives a detailed description of the design process.

The various steps involved in the design process are as shown below:

Step 1: Start the Program.

Step 2: Interaction with windowing system is initiated.

Step 3: Set initial Display Mode.

Step 4: Set initial Window Size to (800,600).

Step 5: Set initial Window Position to (0, 0).

Step 6: Create Window “Traveller”.

Step 7: Front page is displayed.

Step 8: If 'Space Bar' is pressed, GOTO Step 12.

Step 9: If 'h' or 'H' is pressed, GOTO Step 11.

Step 10: If 'q' or 'Q' is pressed, GOTO Step 36.

Step 11: Help page is displayed.

Step 12 : The game begins.

Step 13: In the game, if 'Up Arrow Key' is pressed, GOTO Step 20.

Step 14: In the game, if 'Left Arrow Key' is pressed, GOTO Step 21.

Step 15: In the game, if 'Right Arrow Key' is pressed, GOTO Step 22.

Step 16: In the game, if 'n' or 'N' is pressed, GOTO Step 23.

Step 17: In the game, if 'd' or 'D' is pressed, GOTO Step 24.

Step 18: In the game, if '+' is pressed, GOTO Step 25.

Step 19: In the game, if '-' is pressed, GOTO Step 26.

Step 20: The traveller moves upward.

Step 21: The traveller moves towards the left.

Step 22: The traveller moves towards the right.

Step 23: The game switches to night mode.

Step 24: The game switches to day mode.

Step 25: Increase the speed of the sharks.

Step 26: Decrease the speed of the sharks.

Step 27: On a right click menu is created.

On a left click options in the menu can be selected.

Step 28: If 'Up' option is selected in 'Play Game' menu GOTO Step 20.

Step 29: If 'Left' option is selected in 'Play Game' menu GOTO Step 21.
Step 30: If 'Right' option is selected in 'Play Game' menu GOTO Step 22.
Step 31: If 'Night Mode' option is selected in 'Functions' menu GOTO Step 23.
Step 32: If 'Day Mode' option is selected in 'Functions' menu GOTO Step 24.
Step 33: If 'Increase Shark Speed' option is selected in 'Functions' menu GOTO Step 25.
Step 34: If 'Decrease Shark Speed' option is selected in 'Functions' menu GOTO Step 26.
Step 35: If 'Exit Game' option is selected, GOTO Step 36.
Step 36: Stop.

3.2 Data flow Diagram

A flowchart is a common type of chart that represents an algorithm or process showing the steps as boxes of various kinds, and their order by connecting these with arrows. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields. Flowcharts used to be a popular means for describing computer algorithms. They are still used for this purpose; modern techniques such as UML activity diagrams can be considered to be extensions of the flowchart.

However, their popularity decreased when, in the 1970s, interactive computer terminals and third-generation programming languages became the common tools of the trade, since algorithms can be expressed much more concisely and readably as source code in such a language. Often, pseudo-code is used, which uses the common idioms of such languages without strictly adhering to the details of a particular one.

There are many different types of flowcharts. On the one hand there are different types for different users, such as analysts, designers, engineers, managers, or programmers. On the other hand those flowcharts can represent different types of objects. Sterneckert (2003) divides four more general types of flowcharts:

- Document flowcharts, showing a document flow through system
- Data flowcharts, showing data flows in a system
- System flowcharts showing controls at a physical or resource level
- Program flowchart, showing the controls in a program within a system

Data flow diagrams show how data is processed at different stages in the system. Data flow models are used to show how data flows through a sequence of processing steps. The data is transformed at each step before moving on to the next stage. These processing steps of transformations are program functions when data flow diagrams are used to document a software design.

The following Data Flow Diagram shows the processing of different operations in this project.

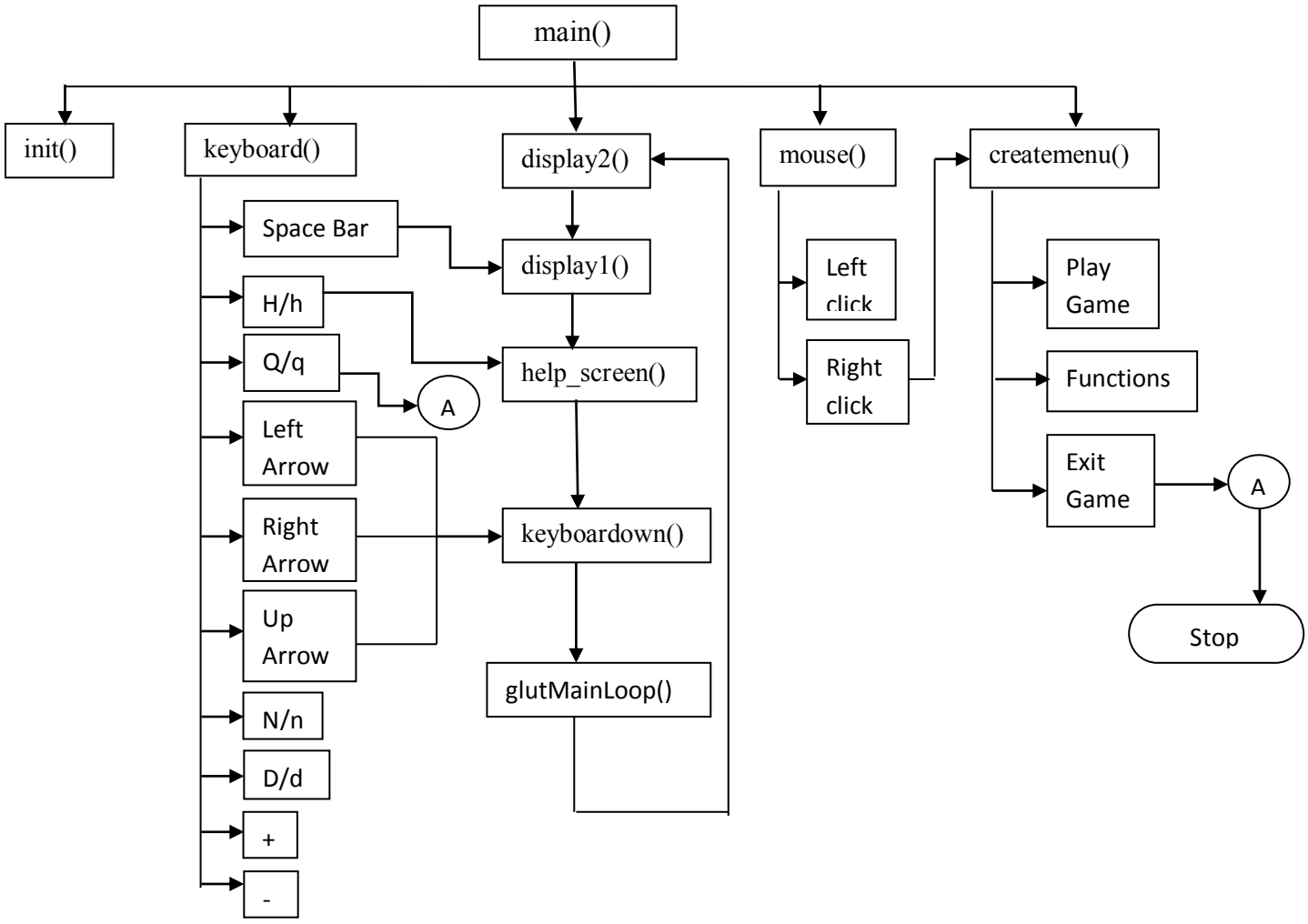

Fig 3.1 Data Flow Diagram

Chapter 4

IMPLEMENTATION

The implementation stage of this model involves the following phases.

- Implementation of OpenGL built in functions.
- User defined function Implementation.

4.1 Implementation of OpenGL Built-In Functions

- ### glClearColor()

  glClearColor is used to clear the color buffer.

  Usage: void glClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a)

  Description: 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.

- ### glutInit( )

  glutInit is used to initialize the GLUT library.

  Usage: void glutInit(int *argcp, char **argv);

  Description: glutInit will initialize the GLUT library and negotiate a session with the window system.

- ### 2. glutInitDisplayMode( )

  glutInitDisplayMode sets the initial display mode.

  Usage: void glutInitDisplayMode (unsigned int mode);

  Mode - Display mode, normally the bitwise OR-ing of GLUT display mode bit masks. See values below:

  GLUT_RGBA- Bit mask to select an RGBA mode window.

  GLUT_RGB- An alias for GLUT_RGBA.

GLUT_SINGLE-Bit mask to select a single buffered window.

GLUT_DOUBLE or GLUT_SINGLE are specified.

GLUT_DOUBLE-Bit mask to select a double buffered window. This overrides GLUT_SINGLE if it is also specified.

GLUT_DEPTH- Bit mask to select a window with a depth buffer.

Description: 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.

- **glutCreateWindow( )**

glutCreateWindow creates a top-level window.

Usage: intglutCreateWindow(char *name);

Name - ASCII character string for use as window name.

Description: 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.

- **glutDisplayFunc( )**

glutDisplayFunc sets the display callback for the current window.

Usage: void glutDisplayFunc(void (*func)(void));

Func- The new display callback function.

Description: glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and 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.

- **glutKeyboardFunc( )**

  glutKeyboardFunc sets the keyboard callback for the current window.

  Usage: void glutKeyboardFunc(void (*func)(unsigned char key,int x, int y));

  Func- The new keyboard callback function.

  Description: glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

- **glutMouseFunc( )**

  glutMouseFunc sets the mouse callback for the current window.

  Usage: void glutMouseFunc(void(*func)(int button, int state, int x, int y))

  Description: glutMouseFunc sets the mouse callback for the current window. When a user presses and releases mouse buttons in the window each press and each release generates a mouse callback. The button parameter is one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON or GLUT_RIGHT_BUTTON. The state parameter is either GLUT_UP or GLUT_DOWN indicating whether the callback was due to press or release. The x and y callback parameters indicate the window relative coordinates when the mouse button state is changed. If a menu is attached to a button for a window, mouse callbacks will not be generated for that button.

- **glutMainLoop( )**

  glutMainLoop enters the GLUT event processing loop.

  Usage: void glutMainLoop(void);

  Description: 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.

- **glMatrixMode( )**

  The two most important matrices are the model-view and projection matrix. At any time, the state includes values for both of these matrices, which are initially set to identity matrices. There is only a single set of functions that can be applied to any type of matrix. Select the matrix to which the operations apply by first set in the matrix mode, a variable that is set to one type of matrix and is also part of the state.

- **gluOrtho2D( )**

  It is used to specify the two dimensional orthographic view. It takes four parameters; they specify the leftmost corner and rightmost corner of viewing rectangle.

- **glutPostRedisplay( )**

  glutPostRedisplay marks the current window as needing to be redisplayed.

  Usage: void glutPostRedisplay(void);

  Description: Mark the normal plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. Texture Name is the ID of the texture (which is acquired via glGenTextures()).

4.2 Implementation of User Defined Functions

- **display_string()**

  Usage: void display_string(int x, int y, char *string, int font)

  Description: This function is used to display the contents of the cover page and the help screen.

- **display2()**

  Usage: void display2()

  Description: This function is used as a callback in the main function to display the initial screen.

- **help_screen()**

  Usage: void help_screen();

  Description: This function is used to display the help screen to the user at any time while playing the game.

- **display_win()**

  Usage: void display_win()

  Description: This function is called when the user successfully reaches the goal and completes the game. A 'CONGRATULATIONS YOU WIN' message is displayed.

- **display_lose()**

  Usage: void display_lose()

  Description: This function is called when the user encounters any obstacle. a 'GAME OVER' message is displayed.

- **traveller()**

  Usage: void traveller()

  Description: This function is used to display the traveller's boat.

```
{
  glColor3f(1.0,0.0,0.0);
  glBegin(GL_POLYGON);
  glVertex2f(380.0,20.0);
  glVertex2f(380.0,40.0);
  glVertex2f(400.0,40.0);
```

```
    glVertex2f(400.0,20.0);
    glEnd();
    glBegin(GL_TRIANGLES);
    glVertex2f(380.0,40.0);
    glVertex2f(360.0,40.0);
    glVertex2f(380.0,20.0);
    glEnd();
    glBegin(GL_TRIANGLES);
    glVertex2f(400.0,40.0);
    glVertex2f(420.0,40.0);
    glVertex2f(400.0,20.0);
    glEnd();
}
```

- **river_banks()**

  Usage: void river_banks()

  Description: This function displays the river banks on either side of the window. This is used to define the starting point and the ending point of the game.

- **obstacles()**

  Usage: void obstacles()

  Description: This function displays the obstacles in the playing area that the use has to avoid in order to reach the goal.

- **sharks()**

  Usage: void sharks()

  Description: This function displays the sharks, which are at constant motion throughout the river. The speed of their movements can be controlled by using the '+' and '-' keys for added difficulty.

- **collision()**

  Usage: void collision()

  Description: This function is used to determine if the player has encountered any obstacle. If so, then the display_lose() function is called. Otherwise, on successful completion of the game, the display_win() function is called.

- **keyboardown()**

  Usage: void keyboardown(unsigned char key, int x, int y)

  Description: This function is used to define actions of special keys such as Arrow Keys on the traveller.

- **createmenu()**

  Usage: void createmenu()

  Description: This function is used to handle operations and functions generated on the right mouse click while playing the game.

Chapter 5

TESTING AND SNAPSHOTS

Testing is a major part of any software product life cycle. The purpose of testing is to establish the presence of defects in the software and is usually the most time consuming part of the software development process. According to most estimates testing requires 30 to 50 percent of the total development effort. There are many testing procedures for software depending on the application. This section describes the testing procedure adopted in the project and the result of the tests.

5.1 Testing Process

The most widely used testing process consist of different stages. The testing process of this project is based on four stages.

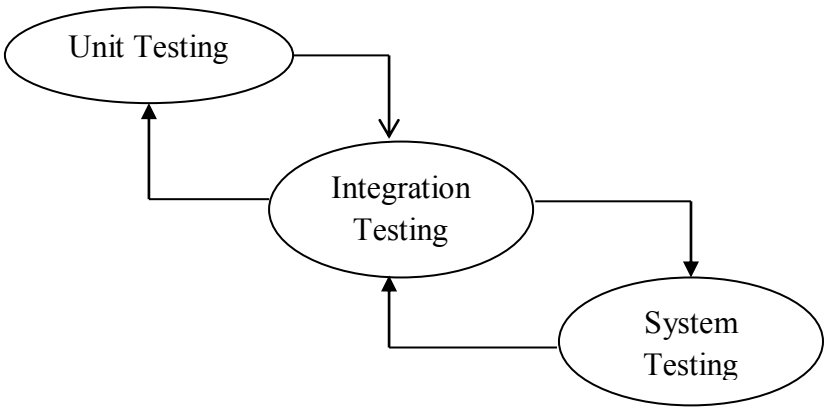

Fig 5.1 Testing Process

5.1.1 Unit Testing

Unit testing is essential for the verification of the code pr6duced during the coding phase and hence the goal is to test the internal logic of the components. Individual components are tested to ensure that they operate correctly. Each component is tested independently without other system components.

The individual components may be the creation of sharks, obstacles, or the movement of the traveller. The display of each component was tested separately here to verify that the desired results were achieved. The testing was carried out during the coding phase.

S1# Test Case:	1
Name Of Test:	Test of Source Code
Item Being Tested:	Source Code
Expected Output:	2D Traveller Game
Actual Output:	2D Traveller Game
Remarks:	Test Pass

Table 5.1 Unit Testing

5.1.2 Integration Testing

The process of system integration involves building a system from it components and testing the resultant system for problems that arise from component interactions. The components that are integrated may be off the shelf components, reusable. Components that have been adapted for a particular system or newly developed components. For many large systems, all three types of components are likely to be used. Integration testing checks that these components actually work together are called correctly and transfer the right data at the right time across their interfaces.

The integration testing in this project comprised of combining and testing the related components together as a whole. The shark creation, traveller movement etc. were integrated and tested together as an entire component. This test was carried out upon completion of the user defined functions.

Global data structures can present problems.

S1# Test Case:	2
Name Of Test:	Compilation of Source Code
Item Being Tested:	Source Code
Expected Output:	Executed Command Prompt
Actual Output:	Executed Command Prompt
Remarks:	Test Pass

Table 5.2 Integration Testing

5.1.3 Validation Testing

The whole of validation testing is to expose latent defects in a software system before the system is delivered. This contrasts with validation testing which is intended to demonstrate that a system meets its specification. Validation testing requires the system to perform correctly using given acceptance test cases.

S1# Test Case:	3
Name Of Test:	Mouse and Keyboard Testing
Item Being Tested:	Mouse and Keyboard buttons
Expected Output:	Corresponding action to be executed
Actual Output:	Corresponding action executed
Remarks:	Test Pass

Table 5.3 Validation Testing

5.1.4 System Testing

The different components are integrated to make up the entire system. The testing process is concerned with finding errors, which result from unanticipated interactions between the different components. It is also concerned with validating that the system meets its functional and non-functional requirements.

The testing procedure was carried out iteratively with addition of every new component. The initial component was the creation of the river banks. This was system tested with the addition of the sharks and the obstacles. These two were retested with the traveller movement and collision detection.

S1# Test Case:	4
Name Of Test:	Creation of Application
Item Being Tested:	Source Code
Expected Output:	2D Traveller Game with 0 error and 0 warnings
Actual Output:	2D Traveller Game with 0 error and 0 warnings
Remarks:	Test Pass

Table 5.4 System Testing

5.2 Snapshots

Snapshots are the pictures representing different phases of the program execution. In computer file systems, a snapshot is a copy of a set of files and directories as they were at a particular point in the past. The term was coined as an analogy to that in photography.

To avoid downtime, high-availability systems may instead perform the backup on a snapshot - read-only copies of the data set frozen at a point in time - and allow applications to continue writing to their data. Most snapshot implementations are efficient and can create snapshots in O(1). In other words, the time and I/O needed to create the snapshot does not increase with the size of the data set, whereas the same for a direct backup is proportional to the size of the data set. These are as shown below in the figures.

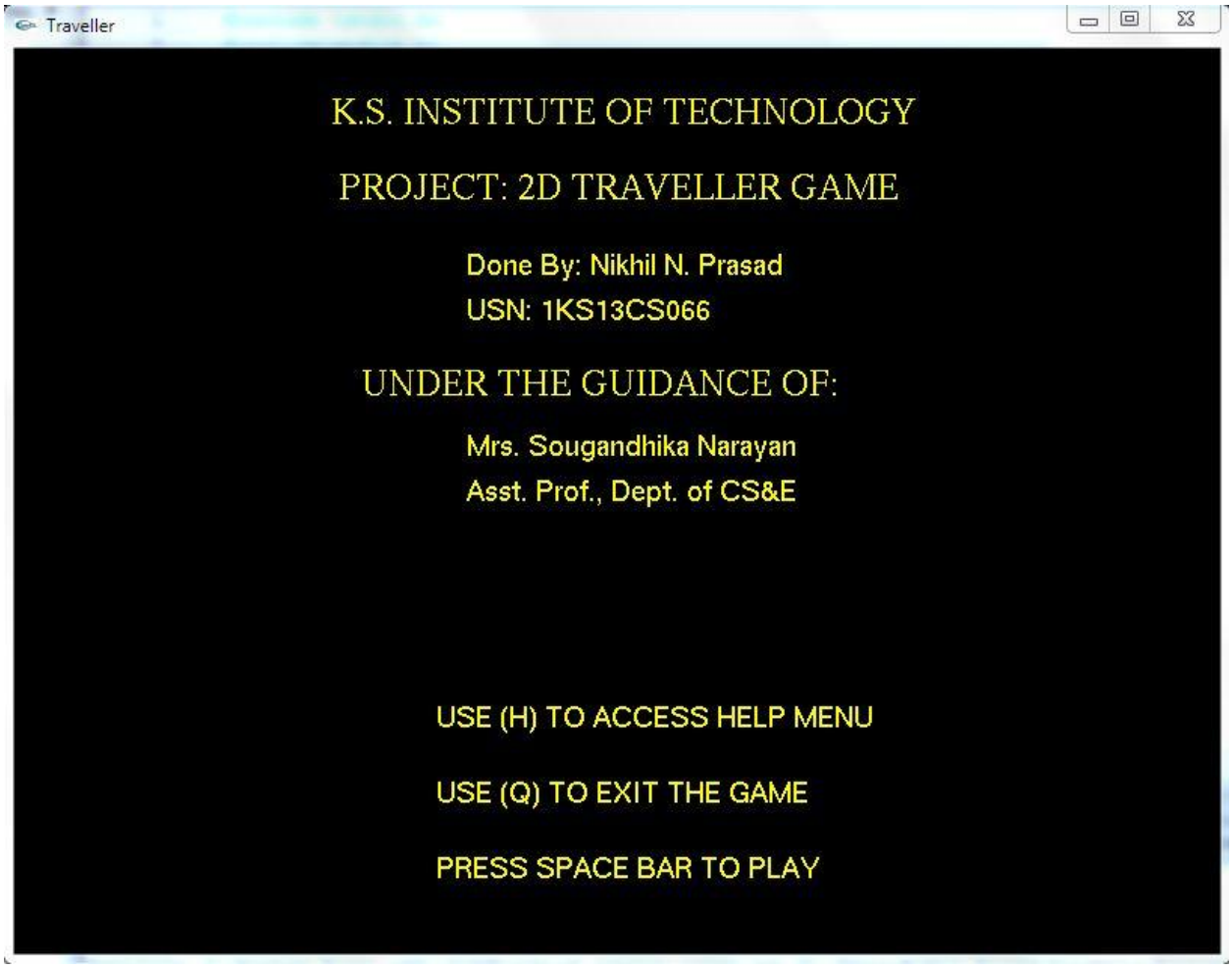

Fig 5.2.1 Executed Window - Front Page

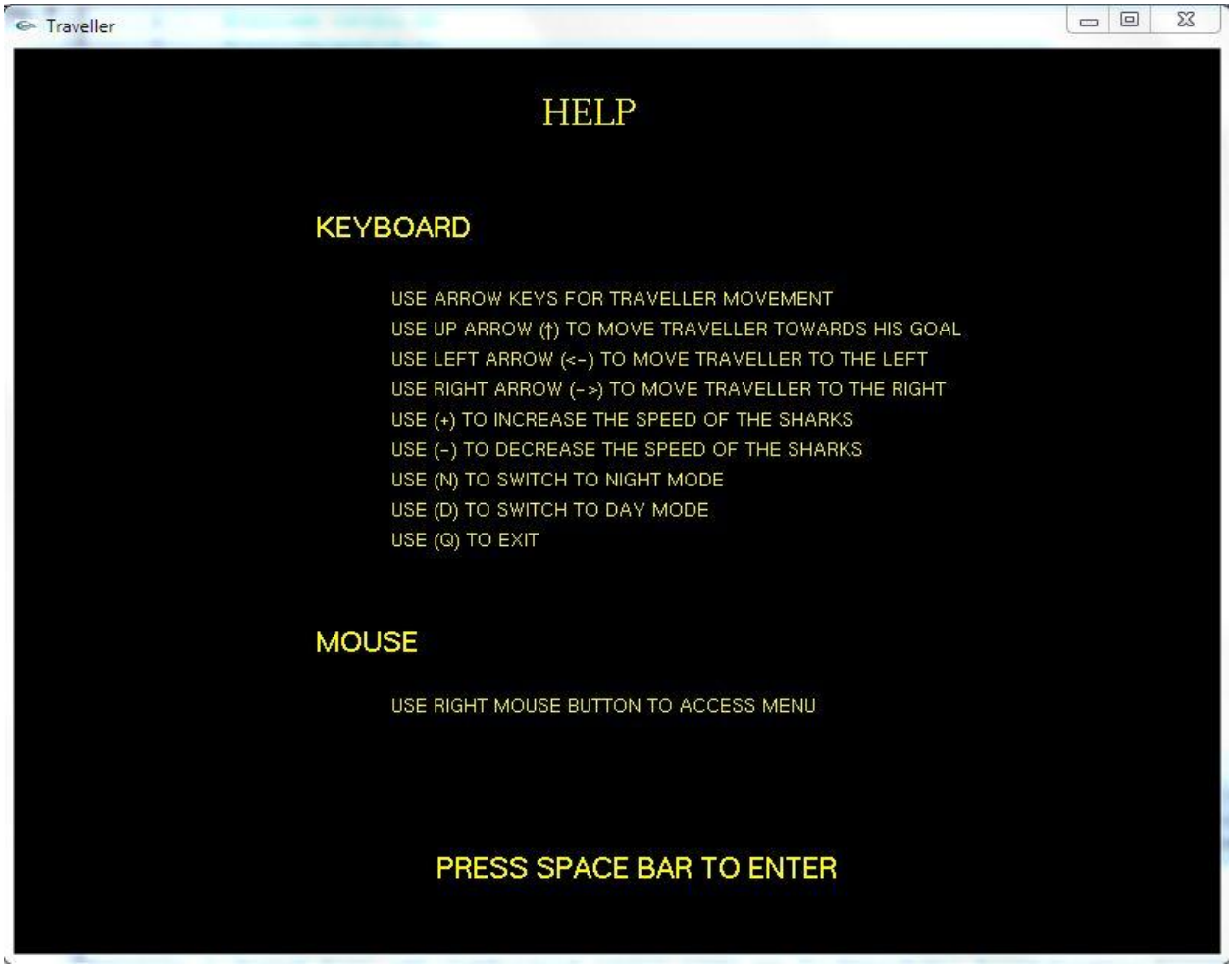

Fig 5.2.2 Help Screen

The help screen can be accessed at any time by pressing the 'h' or 'H' button on the keyboard.

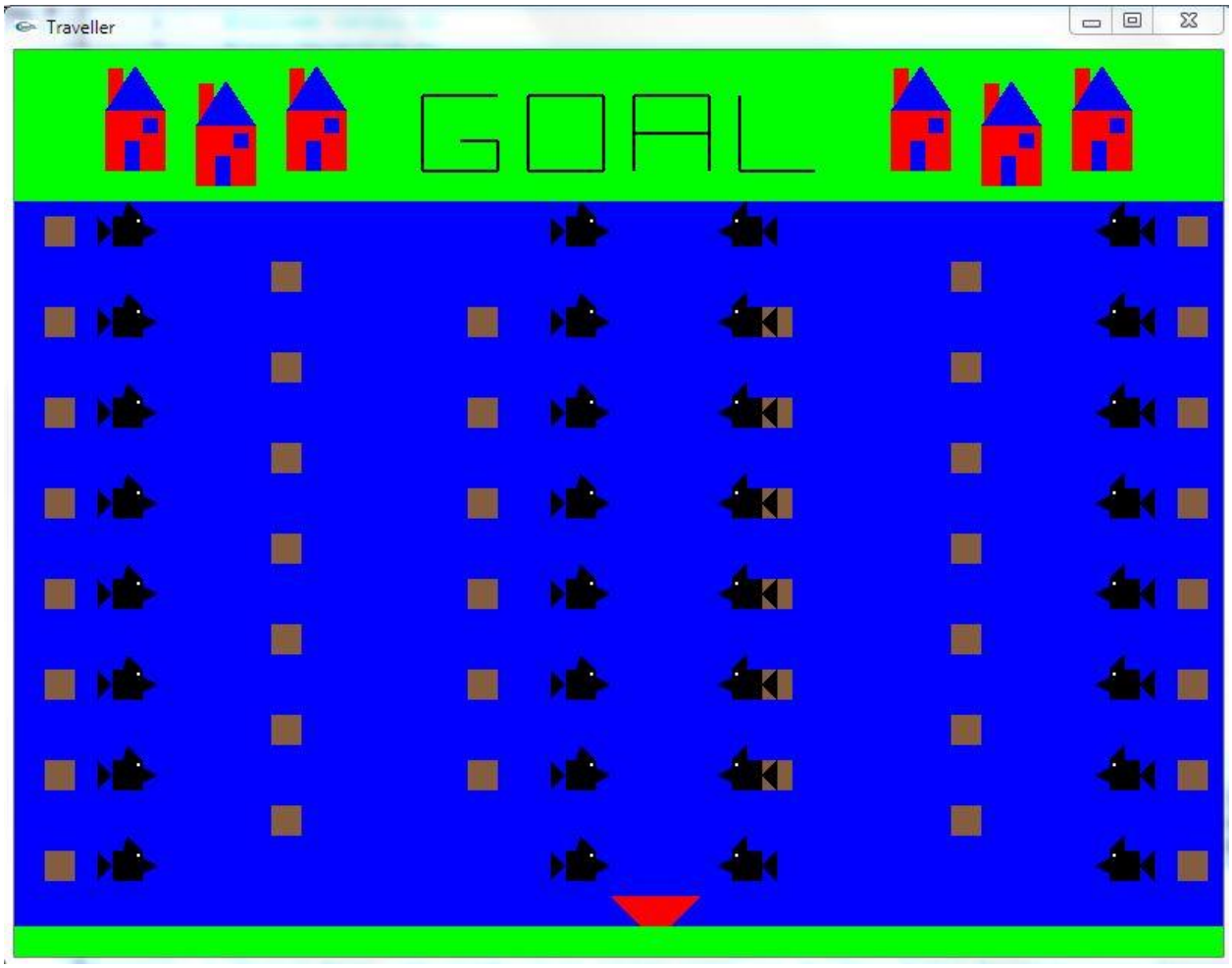

Fig 5.2.3 The 2D Traveller Game - Day Mode

The game can be initialized by pressing the 'Space Bar' key on the keyboard. The game opens in day mode by default. The user can switch to day mode by pressing the 'd' or 'D' key on the keyboard, or by using the mouse menu.

Fig 5.2.4 The 2D Traveller Game - Night Mode

The user can switch to night mode by pressing the 'n' or 'N' key on the keyboard, or by using the mouse menu. In this mode, the user cannot see the sharks. Rather, the user can only see the eyes of the shark and has to judge its position while playing so as to not collide with it. This increases the difficulty of the game.

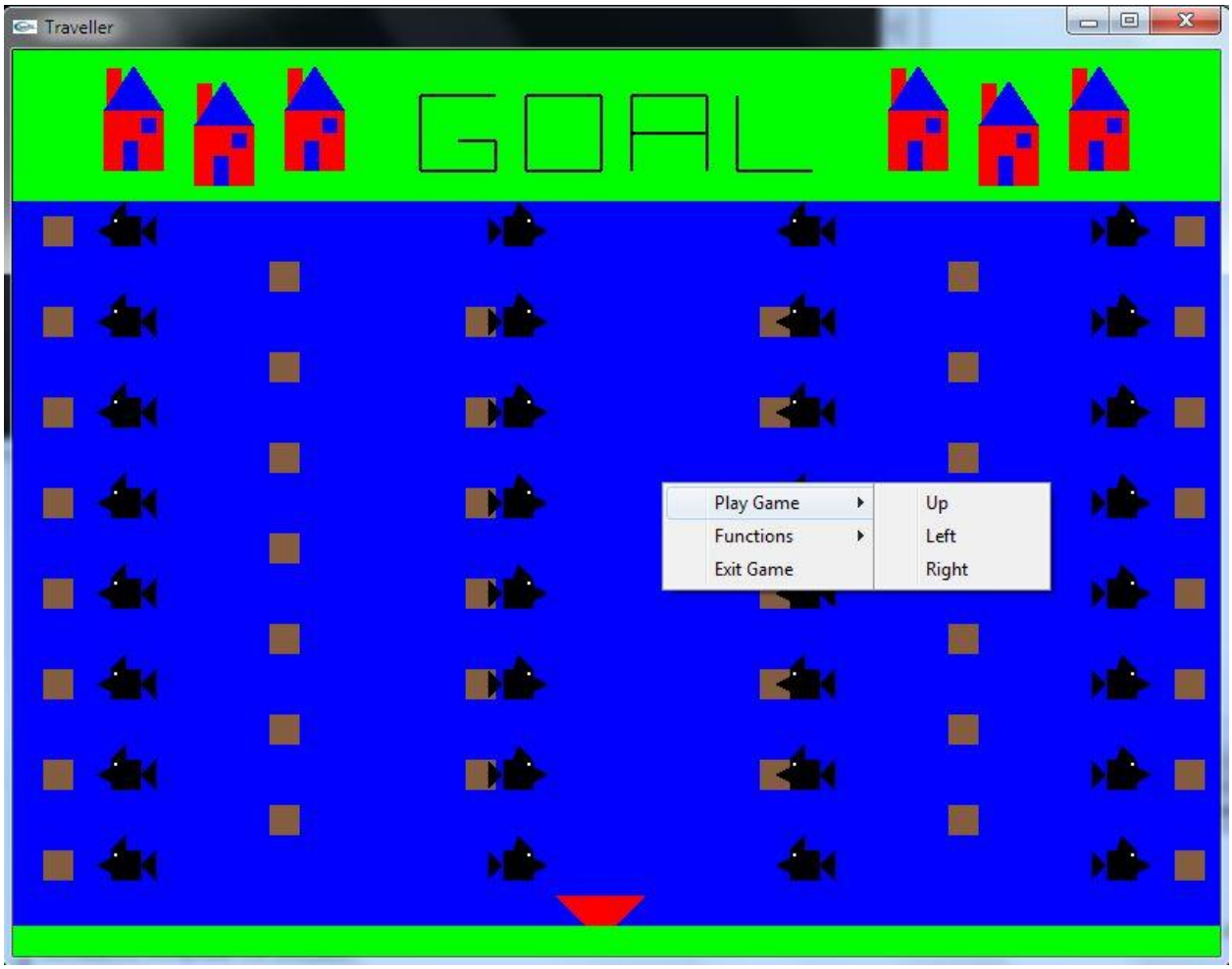

Fig 5.2.5 Playing the Game 1

The user has to reach the goal without colliding with any of the sharks, or the obstacles in the path. The user can use the arrow keys or the mouse menu to move.

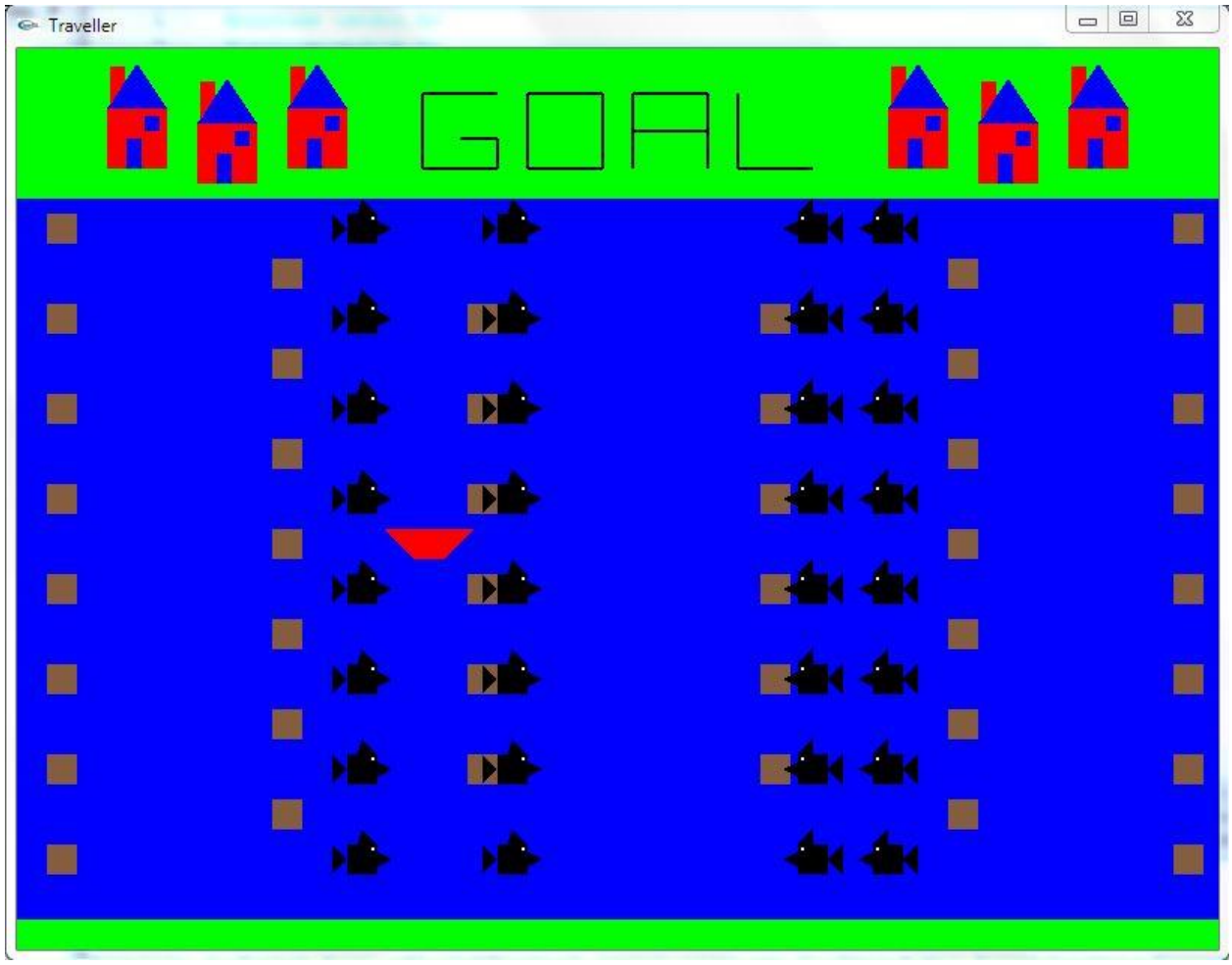

Fig 5.2.6 Playing the Game 2 - Day Mode

Fig 5.2.7 Playing the Game 3 - Changing to Night Mode

The user can switch to night mode by pressing the 'n' or 'N' key on the keyboard, or by using the mouse menu. This can give an additional challenge to the user. Also, the user can increase or decrease the speed of the moving sharks by using the '+' or '-' keys respectively on the keyboard, or by using the mouse menu as shown.

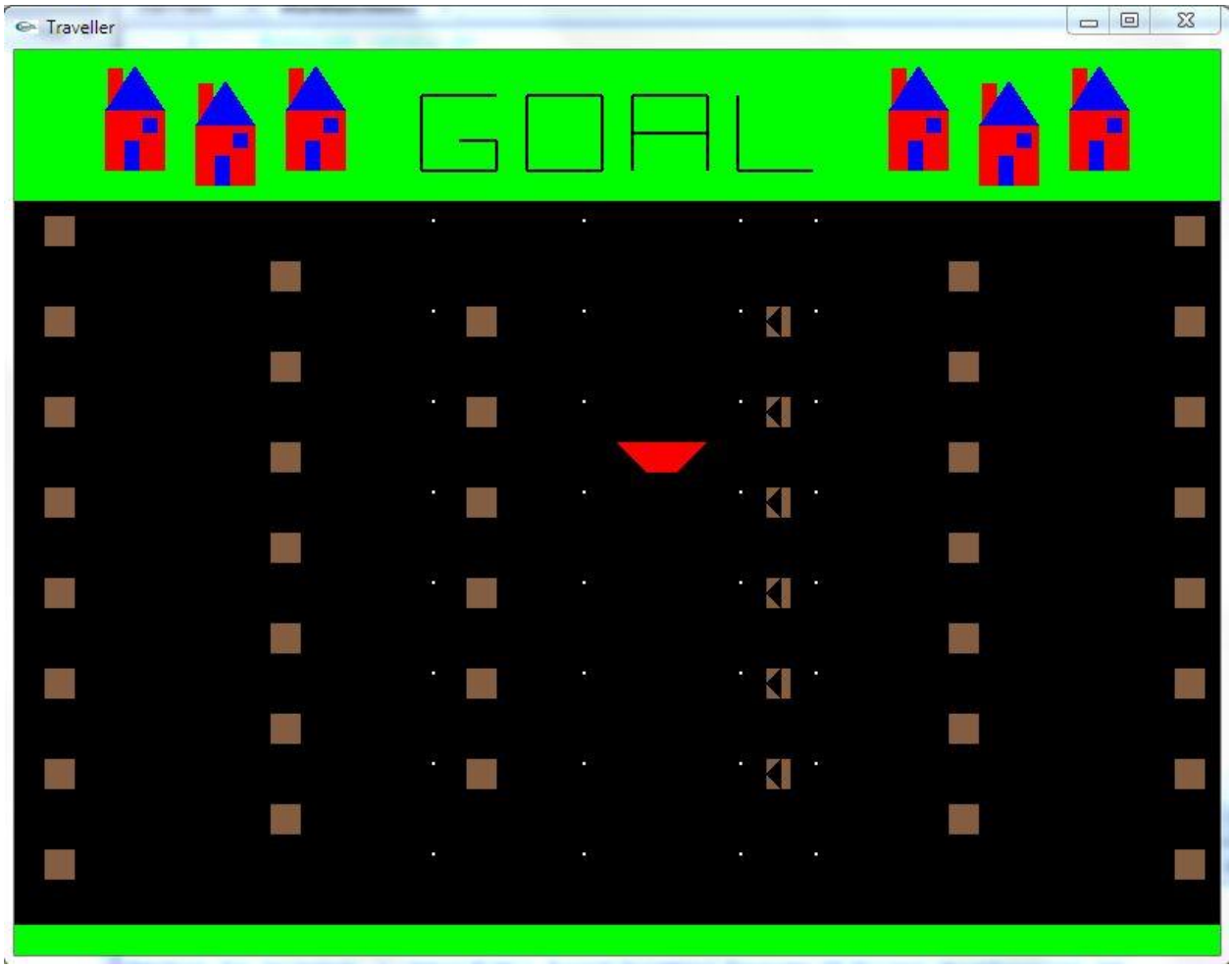

Fig 5.2.8 Playing the Game 4 - Night Mode

Fig 5.2.9 Win Screen Display

If the user manages to reach the goal without any collisions, the above display is used to indicate the user's victory.

Fig 5.2.10 Lose Screen Display

If the user collides with any of the sharks, or any obstacle in the path, the above display is used to indicate the user's defeat.

Chapter 6

FUTURE ENHANCEMENTS

An attempt has been made to develop an OpenGL package which meets the necessary requirements of the user successfully. Since it is user friendly, it enables the user to interact efficiently and easily.

The development of the mini project has given us a good exposure to OpenGL, by which we have learnt some of the techniques which help in development of animated pictures, gaming, etc.

Hence it is helpful for us to take up this field as our career as well, and develop some other features in OpenGL, as a token of contribution to the graphics world.

The currently developed project can improve in many areas. Time constraints and deficient knowledge has been a major factor in limiting the features offered by this model.
The following features were thought a proposal as, but not implemented.

- Improving the design of objects.
- Addition of more interactive features.
- Implementing the project in 3D.

A sincere effort has been made to make this application bug free and user friendly. But, as it is the case with all applications, it may be prone to errors and drawbacks. Removal of these is also a possible enhancement to this application.

Chapter 7

CONCLUSION

This mini project is about the design and implementation of the '2D TRAVELLER GAME' developed using OpenGL. The very purpose of developing this project is to exploit the strength of OpenGL graphic capabilities and to illustrate about game development process.

This project helped us a lot to learn about the proper utilization of various graphics library functions that are defined in GLUT, GLU and GL. The package can be used for educational purposes to demonstrate game development process.

We started with modest aim with no prior experience in any programming projects as this, but ended up in learning many things, fine tuning our programming skills and getting into the real world of software development with an exposure to the corporate environment. During the development of any software of significant utility, we are faced with the trade-off between speed of execution and amount of memory consumed. This is simple interactive application. It is extremely user friendly as the only skill required in executing this program is the knowledge in graphics, and has the user required features, which makes it a simple graphics project. It has an open source code, and no security features has been included. The user is free to alter the code for future enhancement. Checking and verification of all possible types of the functions are taken care of. Care was taken to avoid any errors. Bugs may be reported to creator as the need may be.

So, we conclude on note that we are looking forward to develop more such projects in the future with an appetite to learn more in computer graphics.

REFERENCES

The following books have been referred to complete this project.

- **Interactive Computer Graphics - A Top down Approach Using OpenGL, 5th Edition** by **Edward Angel.**
- **Computer Graphics with OpenGL, 3rd Edition** by **Donald Hearn** and **Pauline Baker.**

Also some of the Online Resources have also been utilized.

- **http://www.opengl.org**
- **http://www.opengl-tutorial.org**