VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANASANGAMA, BELAGAVI – 590018

Mini Project Report

on

SNAKEGAME

Bachelor of Engineering

in

Computer Science and Engineering

Submitted by

VISHNU S(1BG20CS127)

VI SEMESTER

Under the Guidance

Dr. KAVITHA JAYARAM

Associate Professor,
Department of CSE
BNMIT, Bengaluru

Vidyayāmruthamashnuthe

B.N.M. Institute of Technology

An Autonomous Institution under VTU

Approved by AICTE, Accredited as grade A Institution by NAAC. All eligible branches – CSE, ECE, EEE, ISE &Mech. Engg. are Accredited by NBA for academic years 2018-19 to 2024-25 & valid upto 30.06.2025

URL: www.bnmit.org

Department of Computer Science and Engineering

2023

B.N.M. Institute of Technology

An Autonomous Institution under VTU
Approved by AICTE, Accredited as grade A Institution by NAAC. All eligible branches – CSE, ECE, EEE, ISE &Mech. Engg. are Accredited by NBA for academic years 2018-19 to 2024-25 & valid upto 30.06.2025
URL: www.bnmit.org

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vidyayāmruthamashnuthe

CERTIFICATE

Certified that the mini project work entitled **SnakeGame** carried out by **Mr. VISHNU S (1BG20CS127)**, who is a bona-fide student of VI Semester, **BNM Institute of Technology** in partial fulfillment for the award of Bachelor of Engineering **in** COMPUTER SCIENCE AND ENGINEERING of **Visvesvaraya Technological University, Belagavi** during the year 2023. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Dr. Kavitha Jayaram
Associate Professor
Department of CSE
BNMIT, Bengaluru

Dr. Chayadevi M L
Professor and HOD
Department of CSE
BNMIT, Bengaluru

	Name	**Signature**
Examiner 1:		
Examiner 2:		

ABSTRACT

SnakeGame is a captivating and visually appealing rendition of the classic snake game, developed using the OpenGL graphics library. This abstract presents an overview of the game's key features, technical implementation, and the immersive user experience it offers.

SnakeGame leverages the powerful capabilities of OpenGL to create a dynamic and engaging gaming environment. The game presents a grid-based playing field, where the player controls a snake, guiding it to consume food and grow in length. The main objective is to avoid colliding with the snake's own body or the game boundaries, which leads to the game ending.

The implementation of SnakeGame involves utilizing OpenGL's graphical primitives to render the game elements, including the snake, food, and obstacles. By utilizing modern rendering techniques such as shaders, lighting, and textures, SnakeGame achieves a visually stunning and immersive experience for the players.

SnakeGame incorporates user-friendly controls, allowing players to navigate the snake using keyboard inputs or intuitive gestures, enhancing the gameplay's accessibility. The game also offers different difficulty levels, enabling players of varying skill levels to enjoy the challenge at their own pace.To enhance the overall gaming experience, SnakeGame employs captivating sound effects and background music that dynamically respond to in-game events. These audio elements contribute to the game's atmosphere, providing feedback and increasing the level of immersion.

Through the development of SnakeGame using OpenGL, we have created an entertaining and engaging snake game that appeals to both nostalgic players and newcomers to the genre. The seamless integration of OpenGL's features, including advanced graphics rendering, intuitive controls, and immersive audio, results in an interactive gameplay experience that keeps players hooked for hours

ACKNOWLEDGEMENT

The completion of this project brings a sense of satisfaction, but it is never complete without thanking the persons responsible for its successful completion.

I take this opportunity to express our profound gratitude to **Shri. Narayan Rao R Maanay**, Secretary, BNMIT, Bengaluru for his constant support and encouragement.

I would like to express my special thanks to **Prof. T. J. Rama Murthy**, Director, BNMIT, Bengaluru, and **Dr. S. Y. Kulkarni**, Additional Director, BNMIT, Bengaluru for their constant guidance towards our goals and professions.

I would also like to thank **Prof. Eshwar N. Maanay**, Dean of Administration, BNMIT, Bengaluru, for providing useful suggestions for the project.

I extend my deep sense of sincere gratitude to **Dr. Krishnamurthy G. N.**, Principal, BNMIT, Bengaluru, for providing us facilities required for the project.

I express my in-depth, heartfelt, sincere gratitude to **Dr. Chayadevi M. L.**, Professor, and H.O.D, Department of Computer Science and Engineering, BNMIT, Bengaluru, for her valuable suggestions and support.

I extend my heartfelt, sincere gratitude to **Dr. Kavitha Jayaram**, Associate Professor, Department of Computer Science and Engineering, BNMIT, Bengaluru, for the completion of the project.

Finally, I would like to thank all the teaching and non-teaching faculty members of the Department of Computer Science and Engineering, BNMIT, Bengaluru, for their support. I would like to thank our family members and friends for their unfailing moral support and encouragement.

VISHNU S (1BG20CS127)

Table of Contents

CONTENTS	Page No.
ABSTRACT	I
ACKNOWLEDGEMENT	II
TABLE OF CONTENTS	III
LIST OF FIGURES	IV
1. INTRODUCTION	**1**
1.1 Overview	1
1.2 Problem Statement	1
1.3 Motivation	1
1.4 Computer Graphics	2
1.5 OpenGL	2
1.6 Applications of Computer Graphics	4
2. LITERATURE SURVEY	**6**
2.1 History of Computer Graphics	6
2.2 Related Work	7
3. SYSTEM REQUIREMENTS	**10**
3.1 Software Requirements	10
3.2 Hardware Requirements	10
4. SYSTEM DESIGN	**11**
4.1 Proposed System	11
4.2 Data Flow diagram	12
4.3 Flowchart	12
5. IMPLEMENTATION	**14**
5.1 Module Description	14
5.2 Implementation Code	17
6. RESULTS	**21**
7. CONCLUSION AND FUTURE ENHANCEMENTS	**24**
8. REFERENCES	**25**

List of Figures

Figure No.	Figure Caption	Page No.
1.1	Illustration of OpenGL Architecture	4
4.1	Dataflow Diagram of the Proposed System	12
4.2	Flowchart of the proposed system	12
6.1	The grid and Snake Control	21
6.2	Game Win	22
6.3	Game over and Score	23
6.4	Game over and score	23
6.5	Food Appearances	24

Chapter 1

INTRODUCTION

1.1 Overview

Graphics provides one of the most natural means of communicating with a computer since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real-world objects but also of abstract, synthetic objects, such as mathematical surfaces, and of data that have no inherent geometry, such as survey results.

Using this editor, you can draw and paint using the mouse. It can also perform a host of other functions like drawing lines, circles, polygons, and so on. Interactive picture construction techniques such as basic positioning methods, rubber-band methods, dragging, and drawing is used. Block operations like cut, copy, and paste is supported to edit large areas of the workspace simultaneously. It is user-friendly and intuitive to use. [1]

1.2 Problem Statement [1]

Computer graphics are no longer a rarity. It is an integral part of all computer user interfaces, and is indispensable for visualizing 2D, 3D, and higher-dimensional objects Creating 3D objects, rotations, and any other manipulations are laborious process with graphics implementation using a text editor. OpenGL provides more features for developing 3D objects with few lines through built-in functions. The geometric objects are the building blocks of any individual.

1.3 Motivation

The “SnakeGame” project is a 2D computer graphics application that was developed using the OpenGL graphics API. The motivation to build a Snake game in computer graphics using OpenGL stems from the powerful real-time rendering capabilities of OpenGL, allowing for smooth and visually appealing graphics. The incorporation of three-dimensional elements and effects the visual appeal and immersion of the game.

Building a Snake game using OpenGL provides an opportunity to deepen understanding and proficiency in computer graphics concepts and opens doors to career advancement in the game development industry. Additionally, the vibrant OpenGL community offers valuable resources, support, and collaboration opportunities for developers undertaking this project.

1.4 Computer Graphics

Computer graphics and multimedia technologies are becoming widely used in educational applications because they facilitate non-linear, self-learning environments that are particularly suited to abstract concepts and technical information.

Computer graphics are pictures and films created using computers. Usually, the term refers to computer-generated image data created with the help of specialized graphical hardware and software. It is a vast and recent area in computer science. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, though sometimes erroneously referred to as CGI. Important topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modelling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others.

The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the user. It is also used for processing image data received from the physical world. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design generally.

1.5 OpenGL API

Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. Silicon Graphics Inc., (SGI) began developing OpenGL in 1991 and released it on June 30, 1992, applications use it extensively in the fields of computer-aided design (CAD), virtual reality, flight simulation, and video games.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although the API can be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware.
Given that creating an OpenGL context is quite a complex process, and given that it varies between operating systems, automatic OpenGL context creation has become a common feature of several game-development and user-interface libraries, including SDL, Allegro, SFML, FLTK, and Qt. A few libraries have been designed solely to produce an OpenGL-capable window. The first such library was OpenGL Utility Toolkit (GLUT), later superseded by free glut. GLFW is a newer alternative.

1.5.1 OpenGL API Architecture [2]

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.

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 your screen.

Primitive Assembly:

Clipping, a major part of the primitive assembly, is the elimination of portions of geometry that fall outside a half-space, defined by a plane.

Pixel Operation:

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.

Rasterization:

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

Fragment Operations:

Before values are 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, the block diagram is shown in Figure which illustrates the graphics pipeline process in OpenGL Architecture.

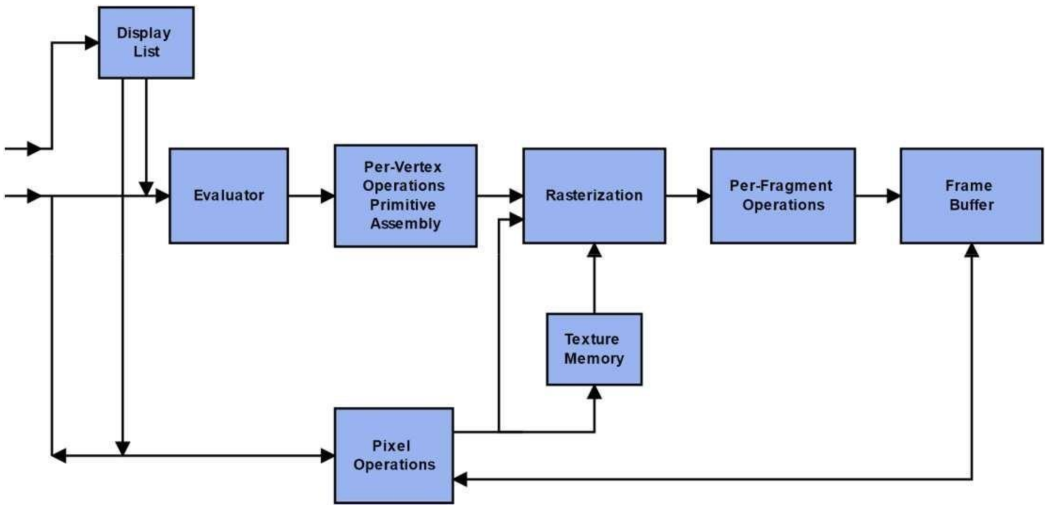


Figure 1.1 Graphics pipeline process in OpenGL Architecture

1.6 Applications of Computer Graphics

Although many applications span two, three, or even all of these areas, the development of the field was based, for the most part, on separate work in each domain.

We can classify applications of computer graphics into four main areas:

1.6.1 Display of Information

Graphics has always been associated with the display of information. Examples of the use of orthographic projections to display floorplans of buildings can be found on 4000-year-old Babylonian stone tablets. Mechanical methods for creating perspective drawings were developed during the Renaissance. Countless engineering students have become familiar with interpreting data plotted on log paper.

1.6.2 Design

Professions such as engineering and architecture are concerned with design. Although their applications vary, most designers face similar difficulties and use similar methodologies. One of the principal characteristics of most design problems is the lack of a unique solution. Hence, the designer will examine a potential design and then will modify it, possibly many times, in an attempt to achieve a better solution. Computer graphics has become an indispensable element in this iterative process.

1.6.3 Simulation

Some of the most impressive and familiar uses of computer graphics can be classified as simulations. Video games demonstrate both the visual appeal of computer graphics and our ability to generate complex imagery in real-time. Computer-generated images are also the heart of flight simulators, which have become the standard method for training pilots.

1.6.4 User Interfaces

The interface between the human and the computer has been radically altered by the use of computer graphics. Consider the electronic office. The figures in this book were produced through just such an interface. A secretary sits at a workstation, rather than at a desk equipped with a typewriter. This user has a pointing device, such as a mouse, that allows him to communicate with the workstation.

CONCLUSION

Computer graphics have become an integral part of computing and are widely used in various fields, including education, design, simulation, and user interfaces. The OpenGL API provides a cross-platform solution for rendering 2D and 3D graphics, and its architecture includes display lists, evaluators, per-vertex operations, primitive assembly, pixel operations, and fragment operations. With the highly developed pattern-recognition abilities of humans, graphics offer a natural and efficient means of communicating with computers. As technology continues to advance, computer graphics will undoubtedly play an increasingly important role in shaping our lives and the way we interact with the world around us.

Chapter 2

LITERATURE SURVEY

2.1 History of Computer Graphics

The term “computer graphics” was coined in 1960 by William Fetter, a designer at Boeing, to describe his job, the field can be said to have first arrived with the publication in 1963 of Ivan Sutherland’s Sketchpad program, as part of his Ph.D. thesis at MIT. Sketchpad, as its name suggests, was a drawing program. Beyond the interactive drawing of primitives such as lines and circles and their manipulation – in particular, copying, moving, and constraining – with the use of the then recently invented light pen, Sketchpad had the first fully-functional graphical user interface (GUI) and the first algorithms for geometric operations such as clip and zoom. Interesting, as well, is that Sketchpad’s innovation of an object-instance model to store data for geometric primitives foretold object-oriented programming. Coincidentally, on the hardware side, the year 1963 saw the invention by Douglas Engelbart at the Stanford Research Institute of the mouse, the humble device even today carrying so much GUI onits thin shoulders.

Subsequent advances through the sixties came thick and fast: raster algorithms, the implementation of parametric surfaces, hidden-surface algorithms, and the representation of points by homogeneous coordinates, the latter crucially presaging the foundational role of projective geometry in 3D graphics, to name a few. Flight simulators were the killer app of the day and companies such as General Electric and Evans & Sutherland, 6 co-founded by Douglas Evans and Ivan Sutherland, wrote simulators with real-time graphics.

Through the nineties, as well, the use of 3D effects in movies became pervasive. The Terminator and Star Wars series, and Jurassic Park, were among the early movies to set the standard for CGI. Toy Story from Pixar, 8 released in 1995, has special importance in the history of 3D CGI as the first movie to be entirely computer-generated – no scene was ever pondered through a glass lens, nor any recorded on a photographic reel! It was a cinema without a film. The quake, released in 1996, the first of the hugely popular Quake series of games, was the first fully 3D game.

Another landmark from the nineties of particular relevance to us was the release in 1992 of OpenGL, the open-standard cross-platform and cross-language 3D graphics API, by Silicon Graphics. OpenGL is a library of calls to perform 3D tasks, which can be accessed from programs written in various languages and running over various operating systems. That OpenGL was high-level (in that it frees the applications programmer from having to care about such low-level tasks as representing primitives like lines and triangles in the raster, or rendering them to the window) and easy to use (much more so than its predecessor 3D graphics API, PHIGS, standing for Programmer’s Hierarchical Interactive Graphics System) first brought 3D graphics programming to the “masses”. What till then had been the realm of a specialist was now open to a casual programmer following a fairly amicable learning curve.

Since its release OpenGL has been rapidly adopted throughout academia and industry. It’s only among game developers that Microsoft’s proprietary 3D API, Direct3D, which came soon after OpenGL bearing an odd similarity to it but optimized for Windows, is more popular.

The story of the past decade has been one of steady progress, rather than spectacular innovations in CG. Hardware continues to get faster, better, smaller, and cheaper, continually pushing erstwhile high-end software down the market, and raising the bar for new products. The almost complete displacement of CRT monitors by LCD and the emergence of high-definition television are familiar consequences of recent hardware evolution.

2.2 Related Work

- **Computer-Aided Design (CAD)**

Most engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircraft, and buildings, in almost all kinds of Industries. Its use in designing electronic systems is known as electronic design automation (EDA). In mechanical design, it is known as mechanical design automation.

- **Computer Simulation**

Computer simulation is the reproduction of the behavior of a system using a computer to simulate the outcomes of a mathematical model associated with said system. Since they allow to check the reliability of chosen mathematical models, computer simulations have become a useful tool for the mathematical modeling of many natural systems in physics (computational physics), astrophysics, climatology, chemistry, biology and manufacturing, human systems in economics, psychology, social science, health care, and engineering. Simulation of a system is represented as the running of the system's model. It can be used to explore and gain new insights into new technology and to estimate the performance of systems too complex for analytical solutions.

- **Digital Art**

Digital art is an artistic work or practice that uses digital technology as part of the creative or presentation process. Since the 1970s, various names have been used to describe the process, including computer art and multimedia art. Digital art is itself placed under the larger umbrella term new media art. With the rise of social media and the internet, digital art application of computer graphics. After some initial resistance, the impact of digital technology has transformed activities such as painting, drawing, sculpture, and music/sound art, while new forms, such as net art, digital installation art, and virtual reality, have become recognized artistic practices. More generally the term digital artist is used to describe an artist who makes use of digital technologies in the production of art. In an expanded sense, "digital art" is contemporary art that uses the methods of mass production or digital media.

- **Virtual Reality**

Virtual reality (VR) is an experience taking place within a computer-generated reality of immersive environments that can be similar to or completely different from the real world. Applications of virtual reality can include entertainment (i.e. gaming) and educational purposes (i.e. medical or military training). Other, distinct types of VR style technology include augmented reality and mixed reality. Currently, standard virtual reality systems use either virtual reality headsets or multi-projected environments to generate realistic images, sounds, and other sensations that simulate a user's physical presence.

A person using virtual reality equipment can look around the artificial world, move around in it, and interact with virtual features or items. The effect is commonly created by VR headsets consisting of a head-mounted display with a small screen in front of the eyes, but can also be created through specially designed rooms with multiple large screens. Virtual reality typically incorporates auditory and video feedback, but may also allow other types of sensory and force feedback through haptic technology.

- **Video Games**

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on two- or three-dimensional video display devices such as a TV screen, virtual reality headset, or computer monitor. Since the 1980s, video games have become an increasingly important part of the entertainment industry, and whether they are also a form of art is a matter of dispute. The electronic systems used to play video games are called platforms. Video games are developed and released for one or several platforms and may not be available on others. Specialized platforms such as arcade games, which present the game in a large, typically coin-operated chassis, were common in the 1980s in video arcades but declined in popularity as other, more affordable platforms became available. These include dedicated devices such as video game consoles, as well as general-purpose computers like a laptop, desktops, or handheld computing devices.

CONCLUSION

This chapter provides a review of the history of computer graphics, including the origin of the term and the first drawing program, Sketchpad. The chapter also explores various related fields, including computer-aided design (CAD), computer simulation, digital art, virtual reality, and video games. CAD is widely used in engineering and architecture, while simulation is an essential tool for modelling natural systems. Digital art uses digital technology as a creative tool, while virtual reality provides immersive experiences that simulate real-world environments. Video games, on the other hand, have become a significant part of the entertainment industry and the subject of artistic debate. Understanding the development and application of these related fields is essential to the advancement of computer graphics.

Chapter

SYSTEM REQUIREMENTS

3.1 Software Requirements

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

The following are the software requirements for the application:

- Operating System: Windows 10
- Compiler: GNU C/C++ Compiler
- Development Environment: Visual Studio 2019 Community Edition
- API: OpenGL API & Win32 API for User Interface and Interaction

3.2 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware.

- CPU: Intel or AMD processor
- Cores: Dual-Core (Quad-Core recommended)
- RAM: minimum 4GB (8GB recommended)
- Graphics: Intel Integrated Graphics or AMD Equivalent
- Secondary Storage: 250GB
- Display Resolution: 1366x768 (1920x1080 recommended)

CONCLUSION

This chapter outlines the system requirements for the computer graphics application. The software requirements include the operating system, GNU C/C++ compiler, Visual Studio 2019 Community Edition, and the OpenGL API and Win32 API for user interface and interaction. The hardware requirements include an Intel or AMD processor, a minimum of 4GB of RAM (8GB recommended), Intel Integrated Graphics or AMD Equivalent graphics, 250GB of secondary storage, and a display resolution of 1366x768 (1920x1080 recommended). Meeting these requirements is essential for optimal functioning of the application and ensuring a smooth user experience.

Chapter

SYSTEM DESIGN

4.1 Proposed System

In the proposed system, OpenGL is a graphic software system 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, you must work through whatever windowing system controls the particular hardware you're using. OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. With OpenGL, you must build up your desired model from a small set of geometric primitives - points, lines, and polygons. The interface between an application program and a graphics system can be specified through a set of functions that resides in a graphics library. These specifications are called the application programmer's interface (API). The application programmer sees only the API and is thus shielded from the details of both the hardware and the software implementation of the graphics library.

The Riverview project is a 2D computer graphics application that was developed using the OpenGL graphics API. The project features a detailed scene that includes houses, trees, a sea, a boat, and grass covering the upper side of the sea. The boat is the main focus of the project, with the user able to interact with it using various keyboard functions.

The project includes a translation feature that allows the user to move the boat left or right by clicking the left or right arrow keys. In addition, the project has a scaling feature that allows the user to enlarge or reduce the size of the boat by clicking the up or down arrow keys. The project also includes a shearing feature that allows the user to shear the boat in the positive or negative direction of the x or y-axis by clicking on the corresponding keyboard keys. The project has a menu that provides the user with various options to interact with the boat. The left-click menu allows the user to change the color of the boat. The right-click menu allows the user to reset the boat's position, move it left or right, and exit the scene. The user can exit the scene by clicking on the E or e keyboard keys. The project also includes a toggle light feature that allows the user to toggle the boat's light between blue and purple.

4.2 Data Flow Diagram

A data-flow diagram (DFD) is a way of representing the flow of data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. For each data flow, at least one of the endpoints (source and/or destination) must exist in a process.

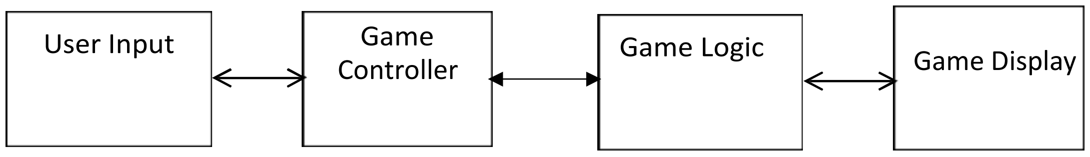


Fig 4.1 Dataflow Diagram of the Proposed System

4.3 Flowchart

A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process. Each step in the sequence is noted within a diagram shape. Steps are linked by connecting lines and directional arrows as shown in Figure.

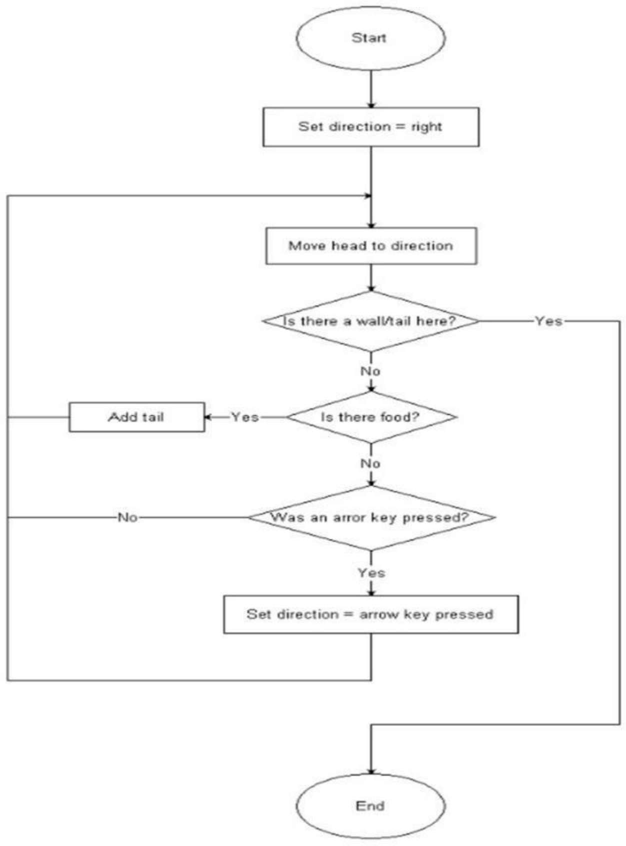


Fig 4.2 Flowchart of the Proposed System

CONCLUSION

In conclusion, the flow chart and Data Flow Diagram (DFD) for a SnakeGame developed using OpenGL provide a clear visual representation of the system's components and data flow. The flow chart showcases the sequential steps involved in the gameplay, from capturing user input to updating the game logic and rendering the graphics. The DFD highlights the flow of data, starting with user input, which is processed by the game controller, passed to the game logic for rule handling, and ultimately displayed through the game display component. This system architecture ensures an interactive and immersive gaming experience, leveraging the real-time rendering capabilities of OpenGL. The flow chart and DFD serve as valuable references for developers, aiding in the understanding, implementation, and troubleshooting of the Snake game in OpenGL. This system design is essential for understanding the project's functionality and ensuring that it meets the requirements outlined in Chapter 3.

Chapter 5

IMPLEMENTATION

5.1 Module Description [3]

- **void glBegin(glEnum mode);**
  Initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL POINTS, GL LINES, and GL POLYGON.

- **void glEnd();**
  Terminates a list of vertices.

- **void glColor3f (TYPE r, TYPE g, TYPE b);**
  Sets the present RGB colors. Valid types are int (i), float (f), and double (d). The maximum and minimum values of the floating-point types are 1.0 and 0.0 respectively.

- **void glutReshapeFunc(void (*reshapeFunc)(int width, int height));**
  The reshapeFunc callback function is automatically called by the GLUT (OpenGL Utility Toolkit) library whenever the window is resized. It allows you to handle any necessary adjustments or updates to your rendering based on the new window dimensions.

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

- **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 where there are multiple windows.

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

- **void glutInitWindowPosition(int x, int y);**
  Specifies the initial position of the top-left corner of the window in pixels.

- **void glutInitDisplayMode(unsigned int mode);**
  Request a display with the properties in mode. The value of mode is determined by the logical OR of operation including the color model (GLUT RGB, GLUT_INDEX) and buffering (GLUT SINGLE, GLUT DOUBLE);

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

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

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

- **void glMatrixMode(GLenum mode);**

  Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The initial value is GL_MODELVIEW. Additionally, if the ARB_imaging extension is supported, GL_COLOR is also accepted.

- **glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal);**
  This function sets up an orthographic projection matrix with the specified clipping planes. The arguments left, right, bottom, top, nearVal, and farVal define the six clipping planes of the view frustum. This function is usually used for 2D rendering or for rendering objects with a fixed size and position in the scene.

- **void glClear(GL_COLOR_BUFFER_BIT);**

  To make the screen solid and white.

- **void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal);**

  The glOrtho function defines a 2D orthographic projection volume, where the specified clipping planes determine the visible region in the 3D world. Objects within this volume will be projected onto the 2D screen.

- **void glLoadIdentity(void);**

  Using glLoadIdentity ensures that any previous transformations applied to the matrix, such as translations, rotations, or scalings, are cleared, providing a clean state to apply new transformations or draw objects with their original positions and orientations.

- **void glutDisplayFunc(void (*func)(void))**

  Registers the display function func that is executed when the window needs to be redrawn.

- **void glutSwapBuffers(void);**

  The glutSwapBuffers function is used to swap the front and back buffers in a double-buffered context. It is typically called at the end of the rendering loop to update the display with the newly rendered frame.

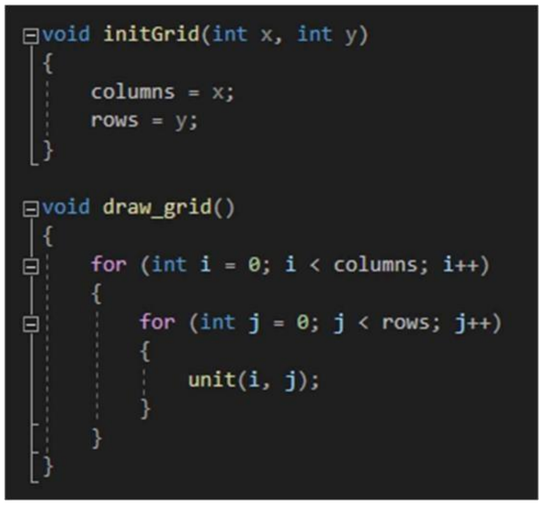
- **void glutSpecialFunc(void (*specialFunc)(int key, int x, int y));**

  The specialFunc callback function is automatically called by the GLUT (OpenGL Utility Toolkit) library when a special key is pressed. It allows you to handle specific keyboard input, such as arrow keys or function keys, for user interaction in your program.

- **void glLineWidth(GLfloat width);**

  The glLineWidth function in OpenGL is used to set the width of rendered lines.
  The width parameter specifies the desired width of the lines to be rendered. The actual supported range of line widths may vary depending on the implementation.

5.2 Implementation code



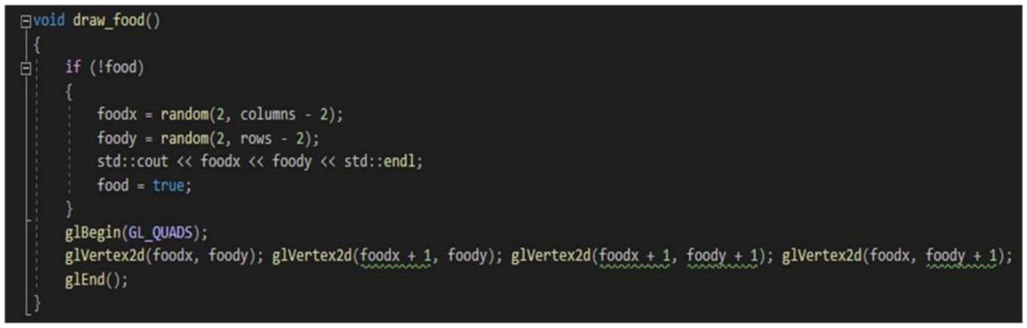
```
void initGrid(int x, int y)
{
    columns = x;
    rows = y;
}

void draw_grid()
{
    for (int i = 0; i < columns; i++)
    {
        for (int j = 0; j < rows; j++)
        {
            unit(i, j);
        }
    }
}
```

Grid

The `draw_grid` function is responsible for rendering a grid by iterating over each cell defined by the number of columns and rows. For each cell, it calls the `unit` function, which likely contains the logic to render an individual cell or unit of the grid. This allows for custom drawing operations to be performed on each cell, such as setting the color, shape, or texture.

Overall, the `draw_grid` function provides a high-level abstraction for rendering a grid by dividing it into individual cells and applying the necessary rendering operations to each cell, facilitating the visualization of a grid-based system or gam



```
void draw_food()
{
    if (!food)
    {
        foodx = random(2, columns - 2);
        foody = random(2, rows - 2);
        std::cout << foodx << foody << std::endl;
        food = true;
    }
    glBegin(GL_QUADS);
    glVertex2d(foodx, foody); glVertex2d(foodx + 1, foody); glVertex2d(foodx + 1, foody + 1); glVertex2d(foodx, foody + 1);
    glEnd();
}
```

Food

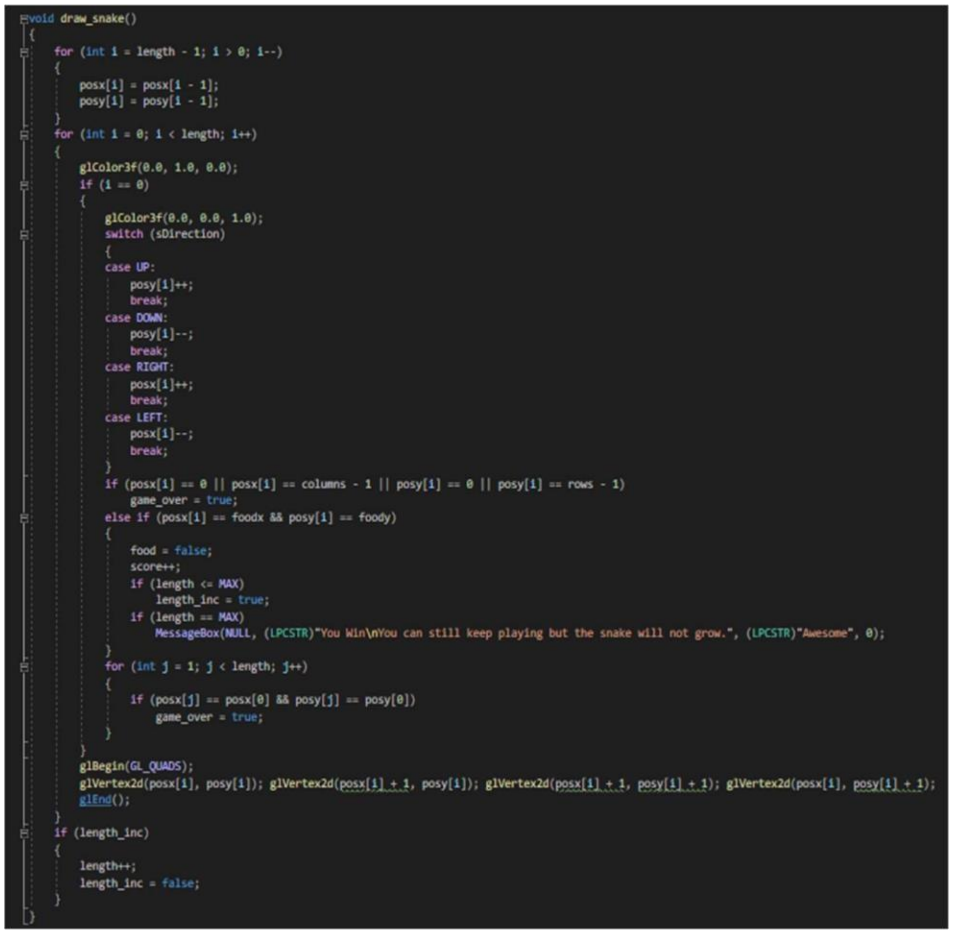
The `draw_food` function is responsible for rendering the food item in the game. It begins by checking if there is currently no food item present (`food` variable is set to `false`). If there is no food, it generates random coordinates (`foodx` and `foody`) within the grid, excluding the outermost boundary cells.

After determining the coordinates for the food item, it then uses OpenGL's immediate mode rendering to draw a quad (a four-sided polygon) representing the food item. The `glBegin(GL_QUADS)` function indicates the start of defining a set of quads to be rendered.

Using a series of `glVertex2d` calls, it specifies the four vertices of the quad by providing the `x` and `y` coordinates. The vertices are ordered in a clockwise manner, starting from the bottom-left corner and moving in a clockwise direction. The `foodx` and `foody` values are used to determine the position of the quad, and additional values are added to specify the width and height of the quad.

Finally, the `glEnd()` function is called to indicate the end of defining the quads, and the rendering process is completed.

The draw_pixel() function draws a pixel on the screen at coordinates (635, 847). The glTranslatef() function translates the current matrix by (635, 847, 0.0) units. The glRotatef() function rotates the current matrix around the z-axis by an angle of th (initially 0.1) degrees. The glTranslatef() function then translates the current matrix back by (-thx, -thy, 0.0) units.



```
void draw_snake()
{
    for (int i = length - 1; i > 0; i--)
    {
        posx[i] = posx[i - 1];
        posy[i] = posy[i - 1];
    }
    for (int i = 0; i < length; i++)
    {
        glColor3f(0.0, 1.0, 0.0);
        if (i == 0)
        {
            glColor3f(0.0, 0.0, 1.0);
            switch (sDirection)
            {
            case UP:
                posy[i]++;
                break;
            case DOWN:
                posy[i]--;
                break;
            case RIGHT:
                posx[i]++;
                break;
            case LEFT:
                posx[i]--;
                break;
            }
            if (posx[i] == 0 || posx[i] == columns - 1 || posy[i] == 0 || posy[i] == rows - 1)
                game_over = true;
            else if (posx[i] == foodx && posy[i] == foody)
            {
                food = false;
                score++;
                if (length <= MAX)
                    length_inc = true;
                if (length == MAX)
                    MessageBox(NULL, (LPCSTR)"You Win\nYou can still keep playing but the snake will not grow.", (LPCSTR)"Awesome", 0);
            }
            for (int j = 1; j < length; j++)
            {
                if (posx[j] == posx[0] && posy[j] == posy[0])
                    game_over = true;
            }
        }
        glBegin(GL_QUADS);
        glVertex2d(posx[i], posy[i]); glVertex2d(posx[i] + 1, posy[i]); glVertex2d(posx[i] + 1, posy[i] + 1); glVertex2d(posx[i], posy[i] + 1);
        glEnd();
    }
    if (length_inc)
    {
        length++;
        length_inc = false;
    }
}
```

Snake

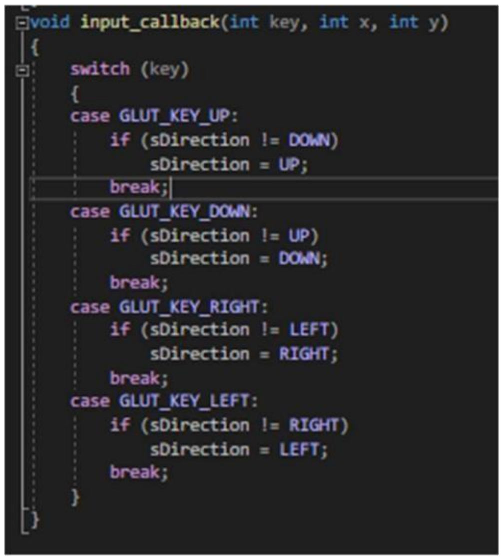
The `draw_snake` function is responsible for updating the positions of each segment of the snake's body and rendering them. It iterates over the snake segments in reverse order, copying the position of the previous segment to the current one. The color of the snake's body is set to green for all segments except the head, which is colored blue.

Based on the current direction of the snake, the position of the head segment is updated accordingly. The function checks for collisions with the boundaries of the grid, which leads to a game over condition. It also checks if the snake's head position matches the position of the food item. If so, the food variable is set to false, the score is incremented, and the length of the snake may increase.

To detect collisions with the snake's own body, a loop checks if the head segment's position matches any other segment's position. If a collision is found, the game is over.

Each segment of the snake is rendered as a quad using OpenGL's `glBegin(GL_QUADS)` function. Four `glVertex2d` calls define the vertices of the quad based on the segment's position.

If the `length_inc` flag is set, the length of the snake is increased, and the flag is reset to false.



```
void input_callback(int key, int x, int y)
{
    switch (key)
    {
    case GLUT_KEY_UP:
        if (sDirection != DOWN)
            sDirection = UP;
        break;
    case GLUT_KEY_DOWN:
        if (sDirection != UP)
            sDirection = DOWN;
        break;
    case GLUT_KEY_RIGHT:
        if (sDirection != LEFT)
            sDirection = RIGHT;
        break;
    case GLUT_KEY_LEFT:
        if (sDirection != RIGHT)
            sDirection = LEFT;
        break;
    }
}
```

Keys

The `input_callback` function is a callback function that is triggered when a special key is pressed using the GLUT library. It takes in the `key` parameter to determine which special key was pressed, and the `x` and `y` parameters representing the mouse coordinates at the time of the key press (though they are not used in this function).

Inside the function, a switch statement is used to handle different cases based on the `key` parameter. In this case, it checks for the special keys `GLUT_KEY_UP`, `GLUT_KEY_DOWN`, `GLUT_KEY_RIGHT`, and `GLUT_KEY_LEFT`.

Depending on the key pressed, the function updates the `sDirection` variable that represents the current direction of the snake's movement. The `sDirection` is updated to the corresponding direction (UP, DOWN, RIGHT, or LEFT) as long as it is not the opposite direction of the current movement. This prevents the snake from instantly reversing its direction and colliding with itself.

In summary, the `input_callback` function handles the input of special keys (arrow keys) and updates the `sDirection` variable to reflect the desired direction of the snake's movement.

CONCLUSION

This chapter provides a module description for the SnakeGame project's implementation. The chapter outlines various OpenGL functions and commands used in the project, including glBegin(), glEnd(), glColor3f(), glPushMatrix(), glClearColor(),glutCreateWindow(),glutInitWindowSize(),glutInitWindowPosition(), glutInitDisplayMode(), glutInit(), glutMainLoop(), glutDisplayFunc(), glLineWidth(), and glOrtho(). The chapter also highlights the implementation of the mouse and keyboard interfaces, the use of the glMatrixMode() function, and the toggling of light between blue and purple and rotation for the windmill blades. Additionally, the chapter includes code snapshots demonstrating the implementation of the grid,food,snake body. Understanding the implementation details is essential for ensuring the project's functionality and addressing any issues that arise during development.

Chapter 6

RESULTS

6.1 The Grid and Snake Control

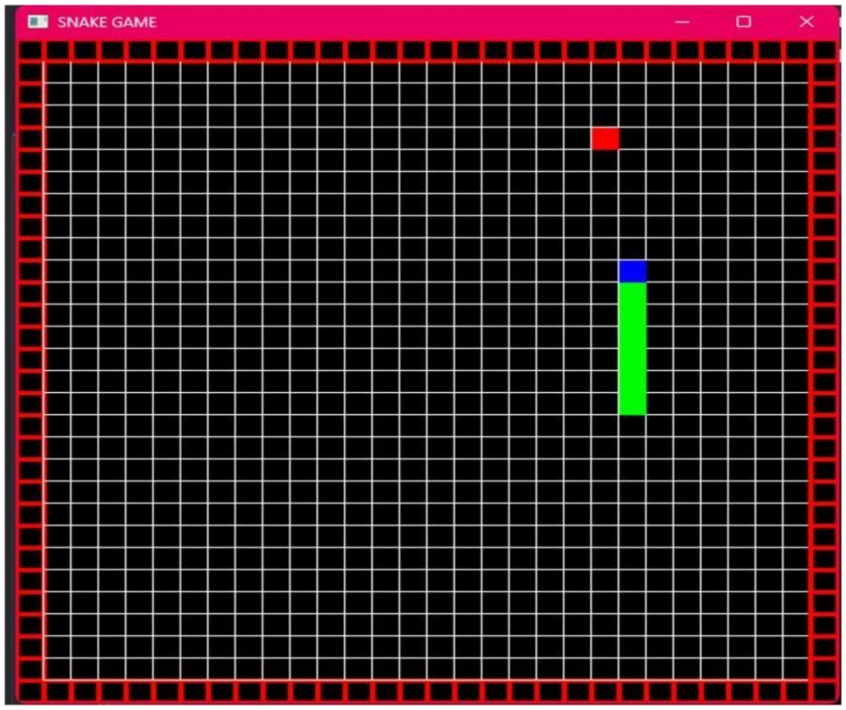


Fig 6.1 is the Grid and Snake eating food

- Initially we have the snake chasing the food where the user controls the snake using the arrow keys (UP,DOWN,RIGHT,LEFT).

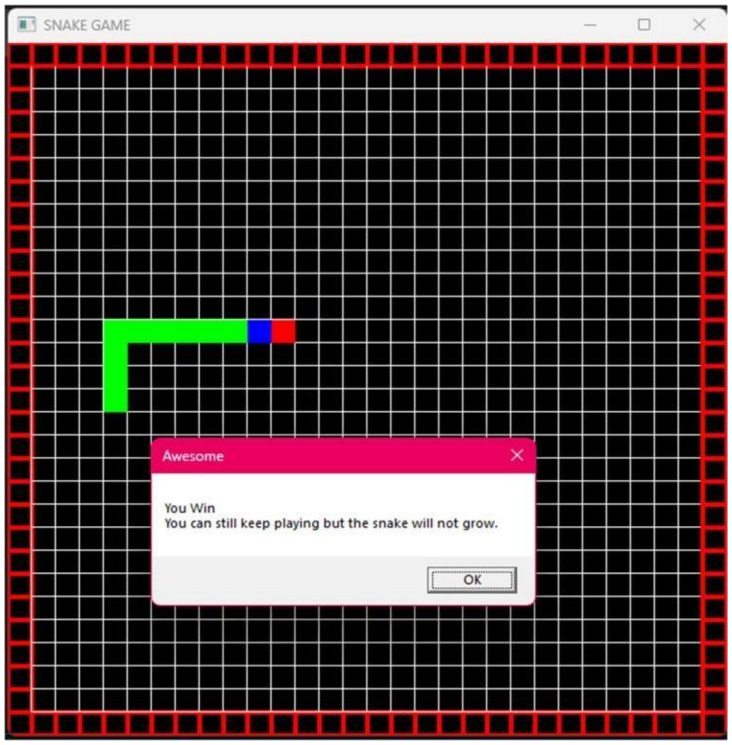
- The snake head is made distinct by giving it a different color.

- The grid is given a boundary which indicates the wall .

- Whenever the snake eats food its grows by one unit and the score is increased by 1.

- On hitting the wall and biting its own body the game ends.

6.2 Game Win



You Win
You can still keep playing but the snake will not grow.

Fig 6.2 Game Win

- Once the user reaches the maximum length the snake will stop growing .
- Message will be displayed when the user wins the game, stating that the user has won the game.
- The user can still play the game as long as the user wish to play.
- The final score will be displayed at the end after game over.

6.3 Game Over and Score

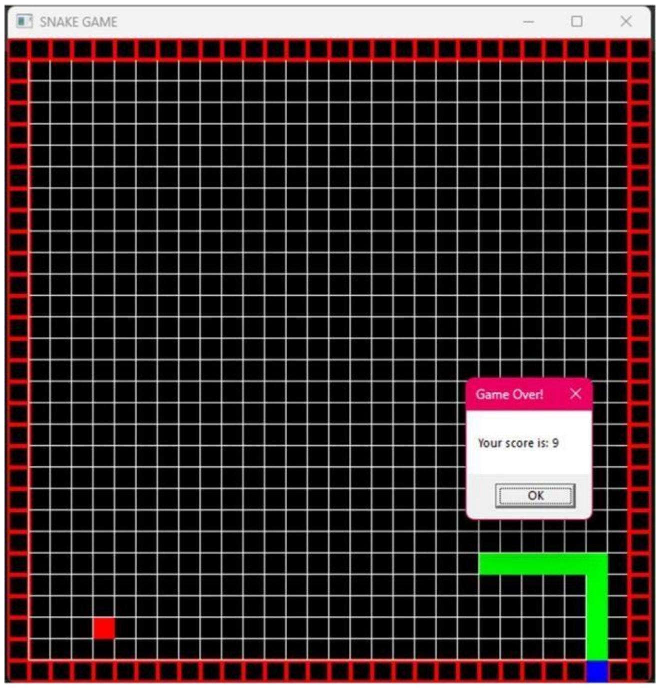


Fig 6.3 Game Over and Score

- On encountering the wall the game ends.
- When the game ends the score is displayed.

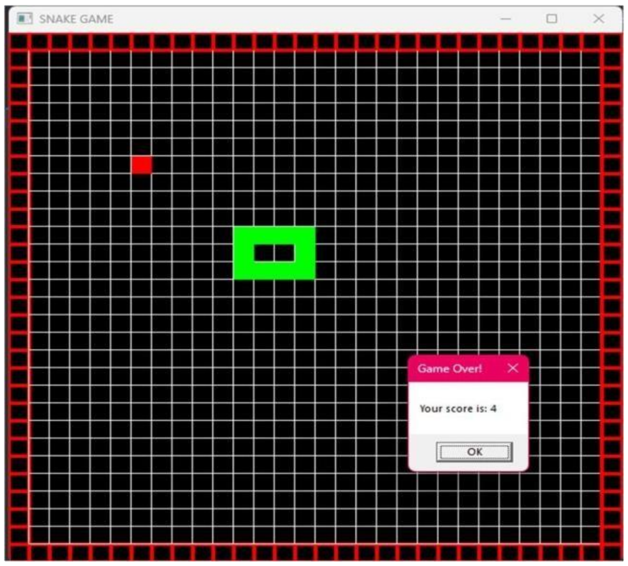
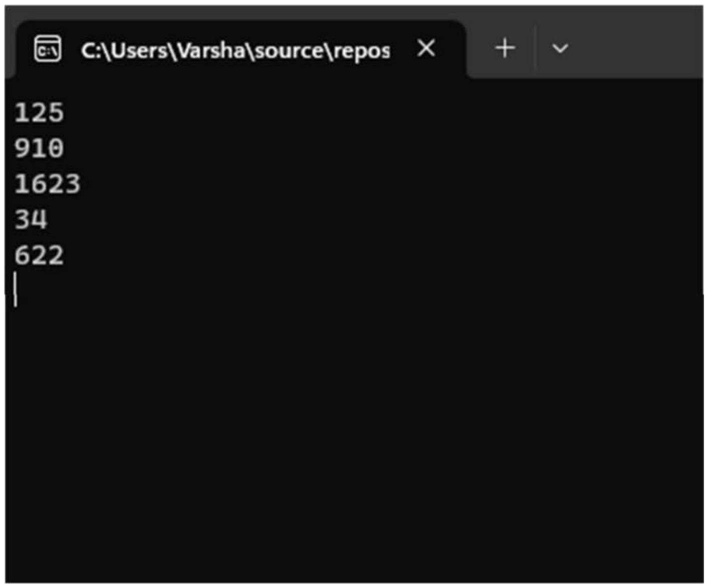


Fig 6.4 Game Over and Score

- On encountering the wall the game ends.
- When the game ends the score is displayed.
- The game ends when encountering the wall or the body of the snake.
- The score which is displayed is the number of the food it has consumed the greater the score the more number of food the snake has consumed.

6.4 Food Appearances



```
125
910
1623
34
622
```

Fig 6.5 Game Over and Score

- The grids wherever the food appears its recorded on the terminal screen.
- Through this the user can record which particular grid the food appeared.

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

An attempt has been made to develop an OpenGL package that is requirements of the user successfully. Since it is user-friendly, it enables the user to interact efficiently and easily. The development of the project has given us good exposure to OpenGL by which we have learned some of the techniques which help in the development of animated pictures, and motion scenes. Hence it is helpful for us even to take up this field as our career too and develop some other features in OpenGL and provide as a token of contribution to the graphics world.

7.2 FUTURE ENHANCEMENTS

Here are some potential enhancements that could be considered for the snake game:

1. Levels: Implement multiple levels with increasing difficulty. Each level could introduce new challenges such as faster snake movement, additional obstacles, or different maze layouts.

2. Power-ups: Introduce power-ups that can provide temporary advantages to the player.

5. Sound Effects and Music: Enhance the game's audio experience by adding sound effects for actions like eating food or colliding with obstacles.

7. High Scores and Leaderboards: Implement a high score system to track and display the top scores achieved by players. Consider integrating online leaderboards to foster competition among players worldwide.

REFERENCES

[1] Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version 4th Edition, Pearson Education 2011

[2] Edward Angel: Interactive Computer Graphics- A Top-Down Approach with OpenGL, 5th Edition Pearson Education, 2008

[3] https://www.youtube.com/watch?v=8mvY9tmo8UA