VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI

Mini Project Report on

"PAC-MAN GAME"

Submitted in the partial fulfillment for the requirements of Computer Graphics & Visualization Laboratory of 6th semester CSE requirement in the form of the Mini Project work

Submitted By

PRATYUSH RAJ SHUKLA	USN: 1BY20CS139
PEEYUSH PARGANIHA	USN: 1BY20CS134
KUMAR KUSHAL	USN: 1BY20CS222

Under the guidance of

Mr. SHANKAR R
Assistant Professor, CSE,
BMSIT&M

Dr. SUNANDA DIXIT
Associate Professor, CSE,
BMSIT&M

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
YELAHANKA, BENGALURU - 560064.

VISVESWARAYA TECHNOLOGICAL UNIVERSITY
Jnana Sangama, Belagavi-590018.

BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
Avalahalli, Yelahanka, Bengaluru – 560064.

CERTIFICATE

This is to certify that the Project work entitled **"PAC-MAN GAME"** is a bonafide work carried out by **PRATYUSH RAJ SHIKLA (1BY20CS139), PEEYUSH PARGANIHA (1BY20CS134) and KUMAR KUSHAL (1BY20CS222)** in partial fulfillment for *Mini Project* during the year 2022-2023. It is hereby certified that this project covers the concepts of *Computer Graphics & Visualization*. It is also certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in this report.

Signature of the Guide with date	**Signature of the Guide with date**	**Signature of HOD with date**
Mr. Shankar R	Dr. Sunanda Dixit	Dr. Thippeswamy G
Assistant Professor	Associate Professor	Prof & Head
CSE, BMSIT&M	CSE, BMSIT&M	CSE, BMSIT&M

EXTERNAL VIVA – VOCE

Name of the Examiners	**Signature with Date**
1. _____	_____
2. _____	_____

INSTITUTE VISION

To emerge as one of the finest technical institutions of higher learning, to develop engineering professionals who are technically competent, ethical and environment friendly for betterment of the society.

INSTITUTE MISSION

Accomplish stimulating learning environment through high quality academic instruction, innovation and industry-institute interface.

DEPARTMENT VISION

To develop technical professionals acquainted with recent trends and technologies of computer science to serve as valuable resource for the nation/society.

DEPARTMENT MISSION

Facilitating and exposing the students to various learning opportunities through dedicated academic teaching, guidance and monitoring.

PROGRAM EDUCATIONAL OBJECTIVES

1. Lead a successful career by designing, analyzing and solving various problems in the field of Computer Science & Engineering.
2. Pursue higher studies for enduring edification.
3. Exhibit professional and team building attitude along with effective communication.
4. Identify and provide solutions for sustainable environmental development.

ACKNOWLEDGEMENT

We are happy to present this project after completing it successfully. This project would not have been possible without the guidance, assistance and suggestions of many individuals. We would like to express our deep sense of gratitude and indebtedness to each and every one who has helped us make this project a success.

We heartily thank our **Principal, Dr. MOHAN BABU G N**, BMS Institute of Technology &Management, for his constant encouragement and inspiration in taking up this project.

We heartily thank our Professor and **Head of the Department, Dr. THIPPESWAMY G, Department of Computer Science and Engineering**, BMS Institute of Technology &Management, for his constant encouragement and inspiration in taking up this project.

We gracefully thank our Project Guide**, Mr. Shankar R, Assistant Professor, and Dr. Sunanda Dixit, Associate Professor**, Department of Computer Science and Engineering for his intangible support and for being constant backbone for our project.

Special thanks to all the staff members of Computer Science Department for their help and kind co-operation.

Lastly, we thank our parents and friends for the support and encouragement given throughout in completing this precious work successfully.

PRATYUSH RAJ SHUKLA (1BY20CS139)

PEEYUSH PARGANIHA (1BY20CS134)

KUMAR KUSHAL (1BY20CS222)

ABSTRACT

The Pac-Man game is a classic arcade game that has gained immense popularity since its release in the 1980s. This abstract presents an implementation of the Pac-Man game using the C programming language with the GLUT (OpenGL Utility Toolkit) library.

The objective of the Pac-Man game is for the player to control the iconic character, Pac-Man, through a maze while eating all the pellets scattered throughout the level. The player must also avoid contact with four ghost enemies that roam the maze, as colliding with them will result in losing a life.

To create this game, the C programming language is used due to its versatility and low-level control over hardware resources. GLUT, a library that simplifies the process of creating graphical applications, is employed to handle window management, user input, and rendering.

The implementation involves several key components. Firstly, the maze layout is defined using a two-dimensional array, representing walls, pellets, and other game objects. The player's input is captured using event-driven programming techniques, allowing Pac-Man to move up, down, left, or right within the constraints of the maze. Collision detection mechanisms are employed to handle interactions between Pac-Man, the ghosts, and the various game objects.

The game's visual representation utilizes OpenGL functions provided by the GLUT library. These functions enable rendering of sprites, texturing, and animation. Additional features such as sound effects and score tracking can also be implemented to enhance the gaming experience.

By using the C++ language with GLUT, developers can create an efficient and interactive implementation of the Pac-Man game. This implementation allows players to relive the nostalgia of the original game while showcasing the power of the C language and the versatility of the GLUT library

Table of Contents

Chapter 1

INTRODUCTION

1.1 Overview

Interactive computer graphics provides us with the most natural means of communicating information through a computer. Over the years advancements in computer graphics have enabled us to not only make pictures of real-world objects but also visualize abstract, synthetic objects such as mathematical surfaces and of data that have no inherent geometry, such as survey results. Some topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, 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. The computer on receiving signals from the input device can modify the displayed picture appropriately. To the user it appears that the picture is changing instantaneously in response to his commands. He can give a series of commands, each one generating a graphical response from the computer. In this way he maintains a conversation, or dialogue, with the computer.

1.2 Problem Statement

Our main aim is to create a classic arcade-style Pac-Man game using the OpenGL Utility Toolkit (GLUT) in C++. The implementation of the Pac-Man game will involve various components, including rendering the maze and characters, handling user inputs for movement, implementing collision detection, managing game logic, and incorporating sound effects for an immersive gaming experience. GLUT, a popular and efficient graphics library, will be utilized to handle window management and provide a simple interface for creating interactive graphical applications.

1.3 Motivation

The Pac-Man game is a well-known 2D maze-based arcade game that has entertained players for decades. The primary objective of the game is to control the iconic Pac-Man character through a maze, consuming pellets while avoiding ghosts. The player must strategize to clear the entire maze and achieve the highest score possible.

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 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 help from 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, scientific visualization, information visualization, flight simulation, and video games. Since 2006 OpenGL has been managed by the non-profit technology consortium Khronos Group.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware. The API is defined as a set of functions which may be called by the client program, alongside a set of named integer constants. In addition to being language-independent, OpenGL is also cross-platform.

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 freeglut. GLFW is a newer alternative.

1.5.1 OpenGL API Architecture

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 primitive assembly, is the elimination of portions of geometry which 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. The results are clamped and then either written into texture memory or sent to the rasterization step.

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

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. More recently, software packages that allow interactive design of charts incorporating color, multiple data sets, and alternate plotting methods have become the norm. In fields such as architecture and mechanical design, hand drafting is being replaced by computer-based drafting systems using plotters and workstations. Medical imaging uses computer graphics in a number of exciting ways.

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.

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 own 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 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 of GUI on its 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.

The seventies, brought the Z-buffer for hidden surface removal, texture mapping, Phong's lighting model – all crucial components of the OpenGL API (Application Programming Interface) we'll be using soon – as well as keyframe-based animation. Photorealistic rendering of animated movie keyframes almost invariably deploys ray tracers, which were born in the seventies too.

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 cinema without film. 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 actually 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 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 of engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircrafts, Building, 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 (MDA) or computer-aided drafting (CAD), which includes the process of creating a technical drawing with the use of computer software.

- **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. 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 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 in a virtual environment. A person using virtual reality equipment is able to 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 a two- or three-dimensional video display device 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, desktop or handheld computing devices.

Chapter 3

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 (>4GB recommended)
- Graphics: Intel Integrated Graphics or AMD Equivalent
- Secondary Storage: 250GB
- Display Resolution: 1366x768 (1920x1080 recommended)

Chapter 4

SYSTEM DESIGN

4.1 Proposed System

User Interface:

Upon opening the Pac-Man game, a visually appealing and user-friendly game window will be presented to the user. The game window will display the labyrinth layout, food items, Pac-Man character, monsters, and other relevant game elements.

Game Initialization:

All necessary game assets, including graphics, sound effects, and level designs, will be loaded and prepared for gameplay. The initial game state will be set up, including initializing the score, level, remaining lives, and positions of Pac-Man and monsters. The user will be provided with clear instructions and an option to start playing the game.

Gameplay:

The user will control Pac-Man's movement using specified input controls such as arrow keys, WASD, or touch-based controls. Pac-Man's movement will be constrained by the labyrinth walls, preventing him from moving through them. As Pac-Man moves through the labyrinth, he will encounter food items. When Pac-Man consumes food, the score will be updated accordingly, and the food item will be removed.

Monsters within the game will exhibit predefined behavior patterns or AI, including chasing Pac-Man or following specific paths. Interactions between Pac-Man and monsters will be handled, such as detecting collisions and determining game over conditions. The game will provide feedback to the user through visual and auditory cues, such as sound effects for eating food, collision events, and level progression.

User Feedback and Progress:

The game window will display the user's current score, level, and remaining lives, allowing them to track their progress. Appropriate messages and visuals will be provided to indicate level completion, game over, or other significant events. Upon completing a level, the user may be given the option to proceed to the next level or start over.

Error Handling:

The game will handle potential errors or exceptional cases gracefully, such as unexpected user input or system failures. Appropriate error messages or prompts will be displayed to guide the user and prevent the game from crashing.

Replayability and Game Continuation:

The game will allow the user to replay the game after completing or losing a level. The option to save and load game progress may be provided, enabling users to continue from where they left off.

Visual and Audio Effects:

The game will incorporate high-quality graphics, animations, and sound effects to enhance the overall gaming experience. Background music and sound effects will accompany the gameplay, creating an immersive environment.

4.2 OpenGL Functions

glClearColor(0.1,0.8,0.1,1.0) :- glClearColor() specifies the red, green, blue, and alpha values used by glClear() to clear the color buffers. Values specified by glClearColor() are clamped to the range 0, 1.

glMatrixMode(GL_MODELVIEW) :- specify which matrix is the current matrix. 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.

gluOrtho2D(0,1000,0,500) :- define a 2D orthographic projection matrix. Left, right- Specify the coordinates for the left and right vertical clipping planes. Bottom, top - Specify the coordinates for the bottom and top horizontal clipping planes.

glRasterPos2f(x, y) :- The glRasterPos2 function uses the argument values for x and y while implicitly setting z and w to zero and one. The object coordinates presented by glRasterPos are treated just like those of a glVertex command. They are transformed by the current modelview and projection matrices and passed to the clipping stage.

glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,string[i]):- glutBitmapCharacter renders a bitmap character using OpenGL. Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font.

glutFullScreen() :- glutFullScreen requests that the current window be made full screen. The exact semantics of what full screen means may vary by window system. The intent is to make the window as large as possible and disable any window decorations or borders added the window system.

glFlush() :- Force execution of GL commands in finite time. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

glPointSize(4) :- specify the diameter of rasterized points. glPointSize specifies the rasterized diameter of points. The value written to the shading language built-in variable gl_PointSize will be used.

glBegin(GL_POINTS) :- delimit the vertices of a primitive or a group of like primitives. glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives. glBegin accepts a single argument that specifies in which of ten ways the vertices are interpreted.

glVertex2i(278,273) :- specify a vertex. glVertex commands are used within glBegin/glEnd pairs to specify point, line, and polygon vertices. The current color, normal, texture coordinates, and fog coordinate are associated with the vertex when glVertex is called.

glEnd() :- delimit the vertices of a primitive or a group of like primitives. glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives.

glutSwapBuffers() :- glutSwapBuffers swaps the bufferss of the current window if double buffered. Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. An implicit glFlush is done by glutSwapBuffers before it returns. If the layer in use is not double buffered, glutSwapBuffers has no effect.

glPushMatrix() push and pop the current matrix stack. There is a stack of matrices for each of the matrix modes. The current matrix in any mode is the matrix on the top of the stack for that mode.glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical to the one below it.

glPopMatrix() :- push and pop the current matrix stack. glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack.

glTranslate(a2,0,0) :- multiply the current matrix by a translation matrix. glTranslate produces a translation by x y z . The current matrix is multiplied by this translation matrix, with the product replacing the current matrix.

glutInit(&argc,argv) :- glutInit is used to initialize the GLUT library. glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. glutInit also processes command line options, but the specific options parse are window system dependent.

glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB) :- sets the initial display mode. The initial display mode is used when creating top-level windows, subwindows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

glutInitWindowSize(1000,500) :- set the initial window size . Windows created by glutCreateWindow will be requested to be created with the current initial window position and size. The initial value of the initial window size GLUT state is 300 by 300. The initial window size components must be greater than zero.The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size and position. The window system is not obligated to use this information. A GLUT program should use the windows reshape callback to determine the true size of the window.

glutCreateWindow("Pac-Man Game") :- 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.

glutPositionWindow(50,50) :- requests a change to the position of the current window. . For top-level windows, the x and y parameters are pixel offsets from the screen origin. For subwindows, the x and y parameters are pixel offsets from the window's parent window origin.

glutDisplayFunc(display1) :- registers the callback function (or event handler) for handling window-paint event. The OpenGL graphic system calls back this handler when it receives a window re-paint request. In the example, we register the function display() as the handler.

glutKeyboardFunc(NormalKey) :- glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data.

glutMainLoop() :- 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.

Chapter 5

IMPLEMENTATION

5.1 USER DEFINED FUNCTIONS

init(): This function initializes the game by setting up various variables and components. Here's what it does in more detail:

- It initializes the score variable to 0, which will keep track of the player's score.
- It sets the level variable to 1, indicating the current level of the game.
- It initializes the graphics and sound components required for the game. This might involve setting up a window or canvas to display the game, loading and rendering graphics assets, and initializing audio resources.
- It sets the initial positions of the pacman and monsters on the screen.
- It sets up any other necessary variables or states required for the game.

drawLaberynth(): This function is responsible for drawing the labyrinth on the screen. Here's what it does in more detail:

- It uses graphics functions or libraries to render the walls, paths, and any other elements of the maze on the screen. This might involve drawing lines, rectangles, or sprites to represent the labyrinth's structure.
- It may include additional logic to handle any dynamic elements in the labyrinth, such as doors that open and close, or obstacles that move.

foodEaten(): This function is called when the pacman consumes a food item. Here's what it does in more detail:

- It updates the score variable by incrementing it based on the value assigned to the food item that was eaten. For example, if each food item is worth 10 points, it would increase the score by 10.
- It checks if all the food items in the labyrinth have been eaten. This could be done by maintaining a separate count of the remaining food items or by checking against a predefined total.
- If all the food items have been eaten, it might trigger a win condition, display a victory message, or transition to the next level of the game.

drawFood(): This function is responsible for drawing the food items in the labyrinth. Here's what it does in more detail:

- It uses graphics functions or libraries to render the food items on the screen. This might involve drawing small circles, dots, or sprites at the respective positions of the food items.
- It may include additional logic to handle any animations or effects associated with the food items, such as blinking or disappearing after being eaten.

drawPacman(): This function is used to draw the pacman character on the screen. Here's what it does in more detail:

- It uses graphics functions or libraries to render the pacman character at its current position on the screen. This might involve drawing a circle or a sprite representing the pacman.
- It may include additional logic to handle the direction and orientation of the pacman based on user input, such as moving left, right, up, or down.
- It updates the position of the pacman based on user input or predefined game logic, such as collision detection with walls or food items.

drawMonster(): This function is responsible for drawing the monsters in the labyrinth. Here's what it does in more detail:

- It uses graphics functions or libraries to render the monster sprites at their respective positions on the screen.
- It may include additional logic to handle the movement and behavior of the monsters. For example, the monsters might move randomly, chase the pacman, or follow predefined paths.
- It updates the positions of the monsters based on their movement patterns or predefined game logic.
- It may include additional logic to handle collisions between the monsters and the pacman, such as checking if the pacman is caught by a monster.

5.2 User Implementation

For Player:

“W”	To Move UP
“S”	To Move Down
“D”	To Move Right
“A”	To Move Left

FIGURE 5.2.1: PAC MAN (A scenario)

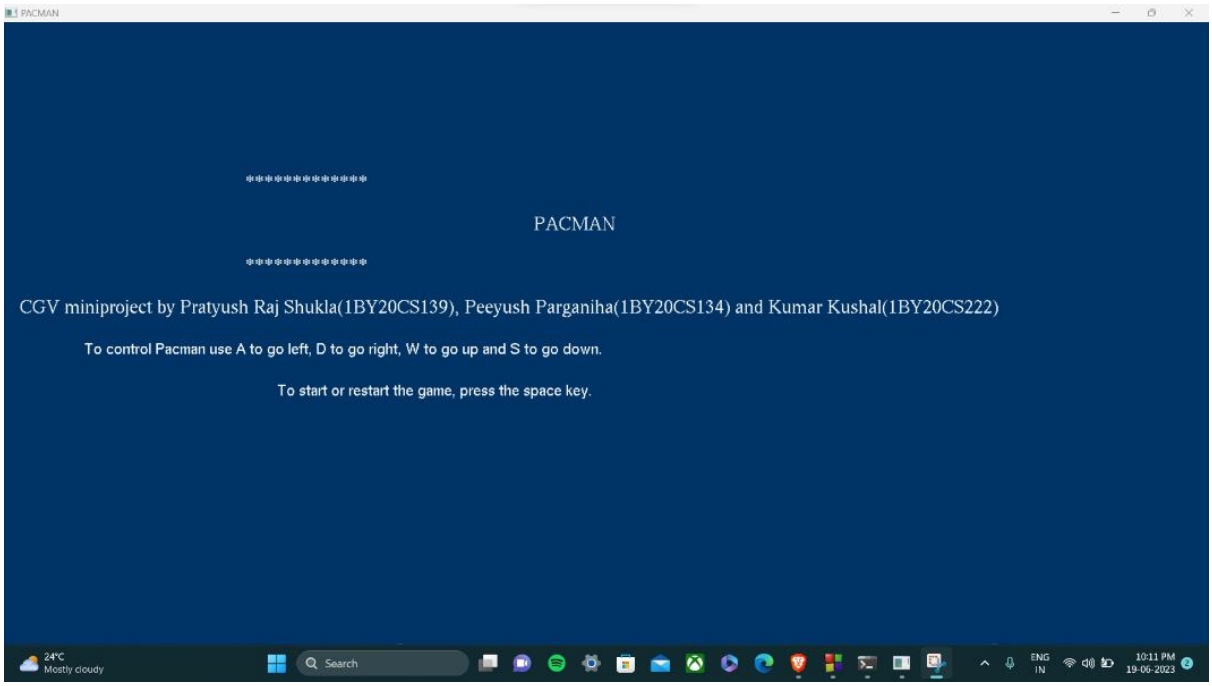

Fig 5.2.2 Startup page

Fig 5.2.3: Result of the game

Chapter 6

CONCLUSION

In conclusion, the Pac-Man game developed using the C language and GLUT library successfully recreates the beloved arcade experience. By addressing challenges such as maze construction, character movement, collision detection, game logic, graphics rendering, user interface, and sound effects, the project achieves an immersive and nostalgic gaming environment.

The project demonstrates the capabilities of the C language and the versatility of the GLUT library for game development. It serves as a valuable learning experience for developers, enhancing their skills in programming, game mechanics, graphics rendering, and user interface design.

Chapter 7

FUTURE ENHANCEMENTS

Some of the future enhancements would be:

- Support for advanced 3D representation of the entire scenario.
- Support for transparency of layers and originality that is, simulating the objects in a more realistic way.
- Making the user interface of this project more user friendly which will certainly be more effectively and efficiently narrated

BIBLIOGRAPHY

[1] Edward Angel: Interactive Computer Graphics: A Top Down Approach 5th Edition, Addison – Wesley, 2008

[2] Donald Hearn and Pauline Baker: OpenGL, 3rd Edition, Pearson Education, 2004

[3] Wikipedia: Computer Graphics – https://en.wikipedia.org/wiki/ComputerGraphics

[4] GeeksForGeeks: Data Communication - Definition, Components, Types, Channels - GeeksforGeeks