

RV COLLEGE OF ENGINEERING®
BENGALURU – 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



Pac-Man Game

COMPUTER GRAPHICS LAB (16CS73)

OPEN ENDED EXPERIMENT REPORT

VII SEMESTER

2020-2021

Submitted by

Sai Venkata Varun D	1RV17CS134
S B RAKSHITH	1RV17CS128

Under the Guidance of

Mamatha T
Department of CSE, R.V.C.E.,
Bengaluru - 560059

RV COLLEGE OF ENGINEERING[®], BENGALURU - 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**



CERTIFICATE

Certified that the **Open-Ended Experiment** titled “Pac-Man Game” has been carried out by **Sai Venkata Varun/1RV17CS134** , **S B RAKSHITH/1RV17CS128**, bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73)** during the year 2020-2021. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report.

Mamatha T
Faculty Incharge,
Department of CSE,
R.V.C.E., Bengaluru –59

Dr. Ramakanth Kumar P
Head of Department,
Department of CSE,
R.V.C.E., Bengaluru–59

RV COLLEGE OF ENGINEERING® , BENGALURU - 560059
(Autonomous Institution Affiliated to VTU)

**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**

DECLARATION

We, **Sai Venkata Varun D/1RV17CS134 , S B RAKSHITH/1RV17CS128** the students of Seventh Semester B.E., Computer Science and Engineering, R.V. College of Engineering, Bengaluru hereby declare that the mini-project titled “Pac-Man Game” has been carried out by us and submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73) - Open-Ended Experiment** during the year 2020-2021. We do declare that matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

Place: Bengaluru

Date:

S B Rakshith

Sai Venkata Varun D

Signature

Abstract

The main aim of this Mini Project is to illustrate the concepts and usage of pre-built functions in OpenGL, to create objects and games like PacMan using OpenGL library. Score is calculated based on the number of pebbles Pac-Man eats. When the location of PacMan is within ten pixels range of round spears, it creates an illusion as if PacMan has eaten that object by decrementing the life of PacMan and restarting from the initial positions. We have used input device like keyboard to interact with the program.

Contents

1	Introduction	2
1.1	Computer Graphics	2
1.2	OpenGL	2
1.2.1	OpenGL Graphics Architecture	3
1.2.2	Display Lists	4
1.3	Project Goal	5
1.4	Scope	5
1.4.1	Pac-Man	5
2	Hardware and Software Requirements	6
2.1	Hardware Requirements	6
2.2	Software Requirements	6
3	Design	7
3.1	Proposed System	7
3.1.1	Detailed Design	7
4	Implementation	9
4.1	Modular Functions	9
4.2	Algorithm	13
4.3	Data Flow Diagram	13
5	Results and Snapshots	15
6	Conclusion	19
A	Appendix A - Source Code	21

Chapter 1

Introduction

1.1 Computer Graphics

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.

1.2 OpenGL

OpenGL (Open Graphics Library) 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 3D scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization and flight simulation. It is also used in video games, where it competes with direct 3D on Microsoft Windows Platforms. OpenGL is managed by the non profit technology consortium, the Khronos group Inc.

OpenGL serves two main purposes :

- To hide the complexities of interfacing with different 3D accelerators, by presenting 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.

OpenGL has historically been influential on the development of 3D accelerator, promoting a base level of functionality that is now common in consumer level hardware:

- Rasterized points, lines and polygons are basic primitives
- A transform and lighting pipeline
- Z buffering
- Texture Mapping
- Alpha Blending

1.2.1 OpenGL Graphics Architecture

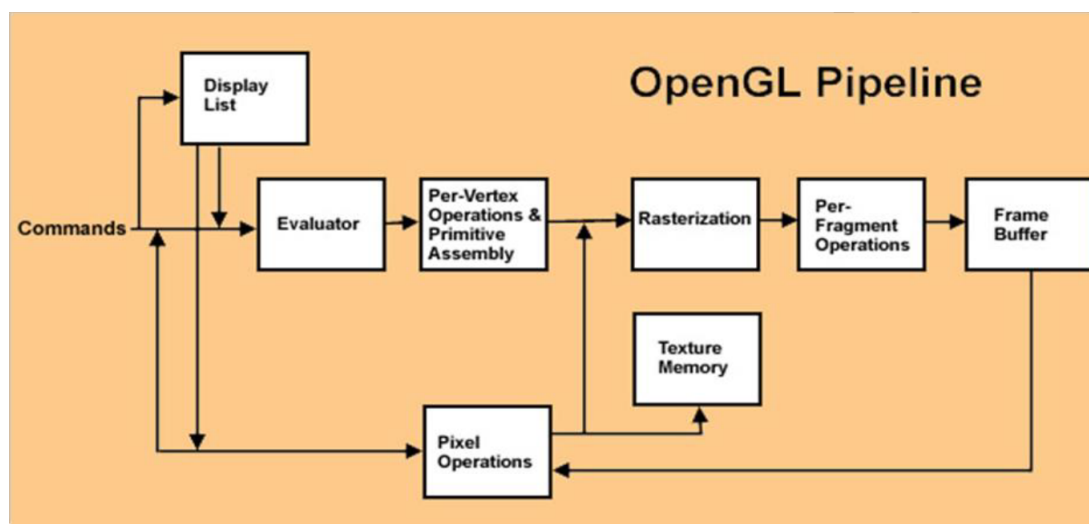


Figure 1.1: OpenGL Graphics Architecture

1.2.2 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 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 framebuffer. Color 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.

1.3 Project Goal

The aim of this project is to develop a 3D Game which supports basic operations which include Movement, Artificial Intelligence, collision Detection and also transformation operations like translation, rotation, scaling etc on objects. The package must also have a user friendly interface .

1.4 Scope

It is developed in ECLIPSE. It has been implemented on UBUNTU platform. The 3-D graphics package designed here provides an interface for the users for handling the display and manipulation of Pac-Man Movements. The Keyboard is the main input device used.

1.4.1 Pac-Man

The game was developed primarily by a young Namco employee Tōru Iwatani, over a year, beginning in April of 1979, employing a nine-man team. The original title was pronounced pakku-man and was inspired by the Japanese onomatopoeic phrase paku-paku taberu where paku-paku describes (the sound of) the mouth movement when widely opened and then closed in succession.

Although it is often cited that the character's shape was inspired by a pizza missing a slice, he admitted in a 1986 interview that it was a half-truth and the character design also came from simplifying and rounding out the Japanese character for mouth, kuchi as well as the basic concept of eating. Iwatani's efforts to appeal to a wider audience beyond the typical demographics of young boys and teenagers eventually led him to add elements of a maze. The result was a game he named Puck Man.

Chapter 2

Hardware and Software Requirements

2.1 Hardware Requirements

- Pentium or higher processor.
- 512 MB or more RAM.

2.2 Software Requirements

This graphics package has been designed for Windows Platform and uses Visual Studio integrated environment.

Development Platform : Windows 10

Development tool : Visual Studio

Language Used In Coding : C++

Chapter 3

Design

3.1 Proposed System

To achieve three dimensional effects, OpenGL software is proposed. It is software which provides a graphical interface. It is an interface between application program and graphics hardware. The advantages are:

- OpenGL is designed as a streamlined.
- It is a hardware independent interface, it can be implemented on many different hardware platforms.
- With OpenGL, we can draw a small set of geometric primitives such as points, lines and polygons etc.
- It provides double buffering which is vital in providing transformations.
- It is event driven software.
- It provides call back function.

3.1.1 Detailed Design

Translation : Translation is done by adding the required amount of translation quantities to each of the points of the objects in the selected area. If $P(x,y)$ be the a point and (tx, ty) translation quantities then the translated point is given by $glTranslatef(dx,dy,dz)$;

Rotation : The rotation of an object by an angle 'a' is accomplished by rotating each of the points of the object. The rotated points can be obtained using the OpenGL functions `glRotatef(angle, vx,vy,vz);`

Scaling : The scaling operation on an object can be carried out for an object by multiplying each of the points (x,y,z) by the scaling factors sx, sy and sz, using `glScalef(sx,sy,sz);`

Chapter 4

Implementation

4.1 Modular Functions

Function for Pac-Man :

- Pacman_Move () - Both the x and y updated coordinates are calculate using the speed of the pacman and trigonometric functions.
- Pacman_Draw () - Here, the pacman is drawn using the glutSolidSphere () function And the eyes of the pacman is rendered using a combination of Random coloring and glutSolidSphere () function.

Collision Detection For Pac-Man And Ghost :

- bool open () - Here, the condition to check if the board is open is given. This is used for collision detection.

Function for Monsters :

- Monster_init () - Here, all the variable values are initialized at the beginning of the Game.
- Monster_Move () - Both the x and y updated coordinates are calculate using the speed of the Monster and trigonometric functions.
- Monster_Updation () - Here, the state of the monster is updated. The edibility condition is checked and the flag is set. If the monster is eaten, then the jail timer starts and the monster is sent to jail.

- `Monster_Vulnerable ()` - Checks the edible condition for the monsters.
- `Monster_Chase ()` - Here, depending on the edible condition for the monsters, they are set to chase pacman or escape from it. This is done by using the x-y coordinates of the pacman. The random movement of the monsters is also handled.
- `Monster_Draw ()` - Here, the pacman is drawn using the `glutSolidSphere ()` function. The color of the monsters is changed depending on the edible condition.

Function for Board:

- `Board_draw ()` - Here, the board is rendered. Its done in 2 steps to avoid complication in depth. Depending on the x-y coordinates, the board is rendered Using different walls. The pebbles are also rendered here. Using the random `f()`, the color of the pebbles is changed To give it a flicker effect.
- `Render_scene ()` - This is the default display function. Here, the collision detection for pacman, the conditions for Normal and super pills consumption, with monster movements Are covered. Options are provided for game control.
- `Create_list ()` - This function is used to create the basic primitive walls using Display lists. Based on the position, the appropriate list are called.

In Built Functions Used:

PushMatrix And PopMatrix

Syntax: `glPushMatrix(); glPopMatrix();`

Description :

Pushes the current transformation matrix onto the matrix stack. The `glPushMatrix()` function saves the current coordinate system to the stack and `glPopMatrix()` restores the prior coordinate system.

Solid Sphere

Syntax: `void glutSolidSphere (GLdouble radius, GLint slices, GLint stacks);`

Description :

Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

get Async KeyState Function

Syntax: *short GetAsyncKeyState(int vKey);*

Description :

The GetAsyncKeyState function determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to GetAsyncKeyState.

Post Redisplay

Syntax: *void glutPostRedisplay();*

Description :

glutPostRedisplay marks the normal plane of current window as needing to be redisplayed. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

Timer Function

Syntax: *void glutTimerFunc(unsigned int msec, void(*func), int value);*

Description :

glutTimerFunc registers the timer callback func to be triggered in at least msec milliseconds. The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc.

Bitmap Character

Syntax: *void glutBitmapCharacter(void *font , int character);*

Description :

Without using any display lists, glutBitmapCharacter renders the character in the named bitmapfont. The available fonts are:

GLUT_BITMAP_TIMES_ROMAN_24 : A 24-point proportional spaced Times Roman font.

GLUT_BITMAP_HELVETICA_18 : A 18-point proportional spaced Helvetica font.

Raster Position

Syntax: *void glRasterPos3f(GLfloat x, GLfloat y, GLfloat z);*

Description :

OpenGL maintains a 3-D position in window coordinates. This position, called the raster position, is maintained with subpixel accuracy. It is used to position pixel and bitmap write operations.

Color Function

Syntax: *void glColor3ub(GLubyte red, GLubyte green, GLubyte blue);*

Description :

This function randomly generates different color based on the rand() function.

Keyboard Function

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

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.

ShadeModel

Syntax: *void glShadeModel(GLenum mode);*

Description :

GL primitives can have either flat or smooth shading. Smooth shading, the default, causes the computed colors of vertices to be interpolated as the primitive is rasterized typically assigning different colors to each resulting pixel fragment. Flat shading selects the computed color of just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive.

4.2 Algorithm

Step 1 : Initialize the graphics window and its size using GLUT functions.

Step 2 : Register the keyboard and display call backs in main function.

Step 3 : When arrow keys are pressed ghosts are released from jail.

Step 4 : If left arrow is pressed the Pac-man move towards left in the maze eating the pebbles simultaneously points are incremented, when points becomes 260 the game is restored.

Step 5 : If right, up, down arrows is pressed Pac-man moves in respective direction eating pebbles.

Step 6 : If Pac-man eats super pebbles ghosts become edible and vulnerable function is called.

Step 7 : If Pac-man collides the ghosts in vulnerable state ghosts go to jail. If the ghosts are uneaten in vulnerable state update function is called.

Step 8 : If Pac-man collides with ghosts provided ghosts are not in vulnerable state Pac man becomes edible and lives are decremented by one. If lives becomes zero the game is over.

4.3 Data Flow Diagram

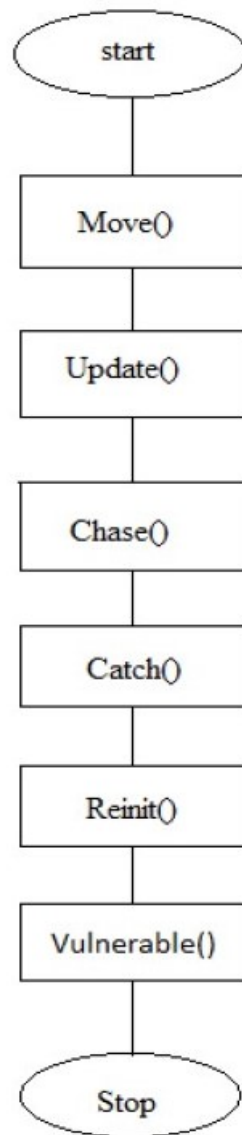


Figure 4.1: Data flow diagram

Chapter 5

Results and Snapshots



Figure 5.1: Initial View Of Pac-Man

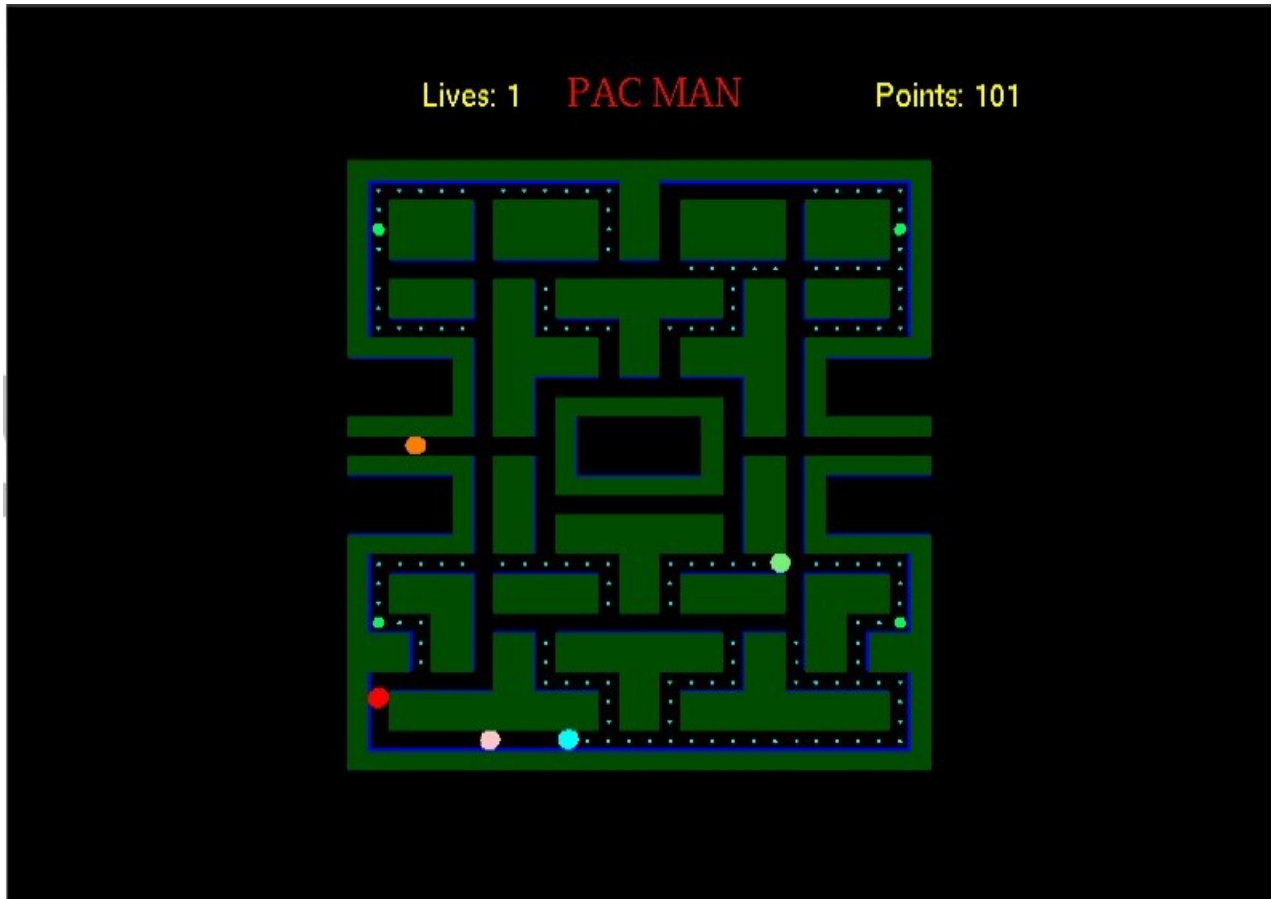


Figure 5.2: Ghosts Chasing The Pac-Man

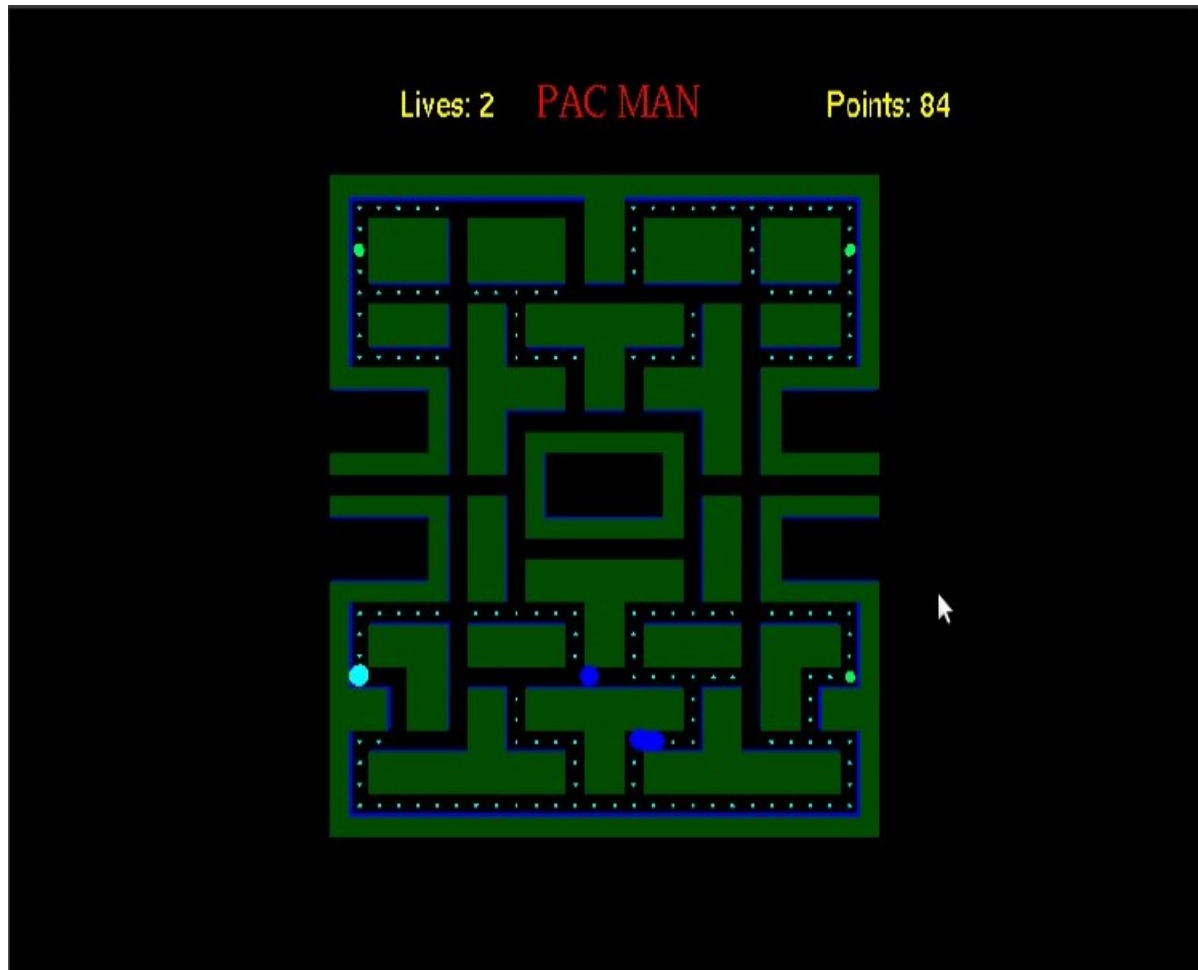


Figure 5.3: Ghosts In The Vulnerable State



Figure 5.4: Game Over

Chapter 6

Conclusion

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

In future the following enhancements could be done:

- Providing Camera Movement.
- Providing More Number of Levels.
- Providing High Quality Graphics.
- Implementing Shortest Path Algorithm for Ghosts.

Bibliography

- [1] The Red Book-OpenGL programming Guide,. *6th edition*.
- [2] Edward Angel. *Interactive Computer Graphics A Top-Down Approach with OpenGL, 5th edition*. Addison, Wesley.
- [3] <http://www.opengl.org/registry/>.
- [4] <http://www.openglforum.com>.
- [5] <http://www.codeguru.com>.

Chapter A

Appendix A - Source Code

```
#include <ctype.h>
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
#define M_PI
    3.14159265358979323846264338327950288419716939937510
#define false 0
#define true 1
const int BOARD_X = 31;
const int BOARD_Y = 28;
int board_array[BOARD_X][BOARD_Y] =
    {{8, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 1, 5, 5, 5,
      5, 5, 5, 5, 5, 5, 5, 5, 5, 7},
     {6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 4, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 6},
     {6, 0, 8, 1, 1, 7, 0, 8, 1, 1, 1, 7, 0, 2, 4, 0, 8, 1,
      1, 1, 7, 0, 8, 1, 1, 7, 0, 6},
     {6, 0, 2, 11, 11, 4, 0, 2, 11, 11, 11, 4, 0, 2, 4, 0, 2,
      11, 11, 11, 4, 0, 2, 11, 11, 4, 0, 6},
     {6, 0, 9, 3, 3, 10, 0, 9, 3, 3, 3, 10, 0, 9, 10, 0, 9,
      3, 3, 3, 10, 0, 9, 3, 3, 10, 0, 6},
```

22

```

    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6},
{6, 0, 8, 1, 1, 7, 0, 8, 1, 1, 1, 7, 0, 2, 4, 0, 8, 1,
    1, 1, 7, 0, 8, 1, 1, 7, 0, 6},
{6, 0, 9, 3, 11, 4, 0, 9, 3, 3, 3, 10, 0, 9, 10, 0, 9,
    3, 3, 3, 10, 0, 2, 11, 3, 10, 0, 6},
{6, 0, 0, 0, 2, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 2, 4, 0, 0, 0, 6},
{2, 1, 7, 0, 2, 4, 0, 8, 7, 0, 8, 1, 1, 1, 1, 1, 1, 7,
    0, 8, 7, 0, 2, 4, 0, 8, 1, 4},
{2, 3, 10, 0, 9, 10, 0, 2, 4, 0, 9, 3, 3, 11, 11, 3, 3,
    10, 0, 2, 4, 0, 9, 10, 0, 9, 3, 4},
{6, 0, 0, 0, 0, 0, 0, 2, 4, 0, 0, 0, 0, 2, 4, 0, 0, 0,
    0, 2, 4, 0, 0, 0, 0, 0, 0, 6},
{6, 0, 8, 1, 1, 1, 1, 11, 11, 1, 1, 7, 0, 2, 4, 0, 8, 1,
    1, 11, 11, 1, 1, 1, 1, 7, 0, 6},
{6, 0, 9, 3, 3, 3, 3, 3, 3, 3, 3, 10, 0, 9, 10, 0, 9, 3,
    3, 3, 3, 3, 3, 3, 3, 10, 0, 6},
{6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 6},
{9, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
    5, 5, 5, 5, 5, 5, 5, 5, 10}}};

int pebble_array[BOARD_X][BOARD_Y] =
{{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 0},
{0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
    0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
{0, 3, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
    0, 0, 0, 1, 0, 0, 0, 0, 3, 0},
{0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,

```

$0, 0, 0, 1, 0, 0, 0, 0, 1, 0\},$
 $\{0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,$
 $1, 1, 1, 1, 1, 1, 1, 1, 1, 0\},$
 $\{0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,$
 $1, 0, 0, 1, 0, 0, 0, 0, 1, 0\},$
 $\{0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,$
 $1, 0, 0, 1, 0, 0, 0, 0, 1, 0\},$
 $\{0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1,$
 $1, 0, 0, 1, 1, 1, 1, 1, 1, 0\},$
 $\{0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 1, 0, 0, 0, 0, 0, 0\},$
 $\{0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 1, 0, 0, 0, 0, 0, 0\},$
 $\{0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 1, 0, 0, 0, 0, 0, 0\},$
 $\{0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 1, 0, 0, 0, 0, 0, 0\},$
 $\{0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 1, 0, 0, 0, 0, 0, 0\},$
 $\{0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 1, 0, 0, 0, 0, 0, 0\},$
 $\{0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 1, 0, 0, 0, 0, 0, 0\},$
 $\{0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 1, 0, 0, 0, 0, 0, 0\},$
 $\{0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
 $0, 0, 0, 1, 0, 0, 0, 0, 0, 0\},$

```

{0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
{0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
{0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
{0, 3, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
 1, 1, 1, 1, 0, 0, 1, 1, 3, 0},
{0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
 1, 0, 0, 1, 0, 0, 1, 0, 0, 0},
{0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
 1, 0, 0, 1, 0, 0, 1, 0, 0, 0},
{0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1,
 1, 0, 0, 1, 1, 1, 1, 1, 1, 0},
{0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
{0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
{0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};

```

```

GLubyte list[5];

```

```

int tp_array[31][28];

```

```

int pebbles_left;

```

```

double speed1 = 0.1;

```

```

double angle1 = 90;

```

```

double a = 13.5, b = 23;

```

```

bool animate = false;

```

```

int lives = 3;

```

```

int points = 0;

```

```

void keys();
unsigned char ckey = 'w';
void mykey(unsigned char key, int x, int y);
bool Open(int a, int b);
void Move()
{
    a += speed1 * cos(M_PI / 180 * angle1);
    b += speed1 * sin(M_PI / 180 * angle1);
    if (animate && ckey == GLUT_KEY_UP && (int)a - a > -0.1
        && angle1 != 270)
    {
        if (Open(a, b - 1))
        {
            //animate = true;
            angle1 = 270;
        }
    }
    else if (animate && ckey == GLUT_KEY_DOWN && (int)a - a >
        -0.1 && angle1 != 90) // s
    {
        if (Open(a, b + 1))
        {
            animate = true;
            angle1 = 90;
        }
    }
    else if (animate && ckey == GLUT_KEY_LEFT && (int)b - b >
        -0.1 && angle1 != 180) //a
    {
        if (Open(a - 1, b))
        {

```

```

        animate = true;
        angle1 = 180;
    }
}
else if (animate && ckey == GLUT_KEY_RIGHT && (int)b - b
    > -0.1 && angle1 != 0) //d
{
    if (Open(a + 1, b))
    {
        animate = true;
        angle1 = 0;
    }
}
}
void Pac(void)
{
    //Draw Pacman
    glColor3f(0, 1, 1);
    glPushMatrix();
    glTranslatef(a, -b, 0);
    glTranslatef(0.5, 0.6, 0);
    glTranslatef((float)BOARDX / -2.0f, (float)BOARDY / 2.0
        f, 0.5);
    glutSolidSphere(0.5, 15, 10);
    glPopMatrix();
}
//Monster Drawing And Moving Begins
bool open_move[4];
bool gameover = false;
int num_ghosts = 4;
int start_timer = 3;

```

```

class Ghost
{
private:
public:
    bool edible;
    int edible_max_time;
    int edible_timer;
    bool eaten;
    bool transporting;
    float color[3];
    double speed;
    double max_speed;
    bool in_jail;
    int jail_timer;
    double angle;
    double x, y;
    Ghost(double, double);
    ~Ghost(void);
    void Move(); //Move the Monster
    void Update(void); //Update Monster
        State
    void Chase(double, double, bool *); //Chase Pacman
    bool Catch(double, double);
    //collision detection
    void Reinit(void);
    void Vulnerable(void);
    void Draw(void); //Draw the Monster
    void game_over(void);
};

Ghost *ghost[4];

Ghost::~~Ghost(void) {}

```



```

Ghost::Ghost(double tx, double ty)
{
    tx = x;
    ty = y;
    angle = 90;
    speed = max_speed = 1;
    color[0] = 1;
    color[1] = 0;
    color[2] = 0;
    eaten = false;
    edible_max_time = 300;
    edible = false;
    in_jail = true;
    jail_timer = 30;
}

void Ghost::Reinit(void)
{
    edible = false;
    in_jail = true;
    angle = 90;
}

//Move Monster
void Ghost::Move()
{
    x += speed * cos(M_PI / 180 * angle);
    y += speed * sin(M_PI / 180 * angle);
}

void Ghost::game_over()
{
}

void Ghost::Update(void)

```

```

{
    if ((int)x == 0 && (int)y == 14 && !(transporting))
    {
        angle = 180;
    }
    if (x < 0.1 && (int)y == 14)
    {
        x = 26.9;
        transporting = true;
    }
    if ((int)x == 27 && (int)y == 14 && !(transporting))
    {
        angle = 0;
    }
    if (x > 26.9 && (int)y == 14)
    {
        x = 0.1;
        transporting = true;
    }
    if ((int)x == 2 || (int)x == 25)
        transporting = false;
    if (((int)x < 5 || (int)x > 21) && (int)y == 14 && !
        edible && !eaten)
        speed = max_speed / 2;
    speed = max_speed;
    //edibility
    if (edible_timer == 0 && edible && !eaten)
    {
        edible = false;
        speed = max_speed;
    }
}

```

```

if (edible)
    edible_timer--;
//JAIL
if (in_jail && (int)(y + 0.9) == 11)
{
    in_jail = false;
    angle = 180;
}
if (in_jail && ((int)x == 13 || (int)x == 14))
{
    angle = 270;
}
//if time in jail is up, position for exit
if (jail_timer == 0 && in_jail)
{
    //move right to exit
    if (x < 13)
        angle = 0;
    if (x > 14)
        angle = 180;
}
//decrement time in jail counter
if (jail_timer > 0)
    jail_timer--;
//EATEN GHOST SEND TO JAIL
if (eaten && ((int)x == 13 || (int)(x + 0.9) == 14) && ((
    int)y > 10 && (int)y < 15))
{
    in_jail = true;
    angle = 90;
    if ((int)y == 14)

```

```

        {
            eaten = false;
            speed = max_speed;
            jail_timer = 66;
            x = 11;
        }
    }
}

bool Ghost::Catch(double px, double py)
{
    // Collision Detection
    if (px - x < 0.2 && px - x > -0.2 && py - y < 0.2 && py -
        y > -0.2)
    {
        return true;
    }
    return false;
}

//called when pacman eats a super pebble
void Ghost::Vulnerable(void)
{
    if (!(edible))
    {
        angle = ((int)angle + 180) % 360;
        speed = max_speed;
    }
    edible = true;
    edible_timer = edible_max_time;
    //speed1=0.15;
}

void Ghost::Chase(double px, double py, bool *open_move)

```

```

{
    int c;
    if (edible)
        c = -1;
    else
        c = 1;
    bool moved = false;
    if ((int)angle == 0 || (int)angle == 180)
    {
        if ((int)c * py > (int)c * y && open_move[1])
            angle = 90;
        else if ((int)c * py < (int)c * y && open_move[3])
            angle = 270;
    }
    else if ((int)angle == 90 || (int)angle == 270)
    {
        if ((int)c * px > (int)c * x && open_move[0])
            angle = 0;
        else if ((int)c * px < (int)c * x && open_move[2])
            angle = 180;
    }
    //Random Moves Of Monsters
    if ((int)angle == 0 && !open_move[0])
        angle = 90;
    if ((int)angle == 90 && !open_move[1])
        angle = 180;
    if ((int)angle == 180 && !open_move[2])
        angle = 270;
    if ((int)angle == 270 && !open_move[3])
        angle = 0;
    if ((int)angle == 0 && !open_move[0])

```

```

        angle = 90;
    }
    void Ghost::Draw(void)
    {
        if (!edible)
            glColor3f(color[0], color[1], color[2]);
        else
        {
            if (edible_timer < 150)
                glColor3f((edible_timer / 10) % 2, (edible_timer
                    / 10) % 2, 1);
            if (edible_timer >= 150)
                glColor3f(0, 0, 1);
        }
        if (eaten)
            glColor3f(1, 1, 0); //When Eaten By PacMan Change
                                Color To Yellow
        glPushMatrix();
        glTranslatef(x, -y, 0);
        glTranslatef(0.5, 0.6, 0);
        glTranslatef((float)BOARDX / -2.0f, (float)BOARDY / 2.0
            f, 0.5);
        glutSolidSphere(.5, 10, 10);
        glPopMatrix();
    }
    void tp_restore(void)
    {
        for (int ISO = 0; ISO < BOARDX; ISO++)
        {
            for (int j = 0; j < BOARDY; j++)
            {

```

```

        tp_array[ISO][j] = pebble_array[ISO][j];
    }
}
pebbles_left = 244;
}
void Draw(void)
{
    glColor3f(1, 0, 1);
    //split board drawing in half to avoid issues with depth
    for (int ISO = 0; ISO < BOARD_X; ISO++)
    {
        for (int j = 0; j < BOARD_Y / 2; j++)
        {
            glColor3f(0, 0, 1);
            int call_this = 0;
            glPushMatrix();
            glTranslatef(-(float)BOARD_X / 2.0f, -(float)
                BOARD_Y / 2.0f, 0);
            glTranslatef(j, BOARD_Y - ISO, 0);
            glPushMatrix();
            glTranslatef(0.5, 0.5, 0);
            switch (board_array[ISO][j])
            {
                case 4:
                    glRotatef(90.0, 0, 0, 1);
                case 3:
                    glRotatef(90.0, 0, 0, 1);
                case 2:
                    glRotatef(90.0, 0, 0, 1);
                case 1:
                    call_this = 1;

```

```

        break;
    case 6:
        glRotatef(90.0, 0, 0, 1);
    case 5:
        call_this = 2;
        break;
    case 10:
        glRotatef(90.0, 0, 0, 1);
    case 9:
        glRotatef(90.0, 0, 0, 1);
    case 8:
        glRotatef(90.0, 0, 0, 1);
    case 7:
        call_this = 3;
        break;
}
glScalef(1, 1, 0.5);
glTranslatef(-0.5, -0.5, 0);
glCallList(list[call_this]);
glPopMatrix();
//now put on the top of the cell
if (call_this != 0 || board_array[ISO][j] == 11)
{
    glTranslatef(0, 0, -0.5);
    glCallList(list[4]);
}
glPopMatrix();
if (tp_array[ISO][j] > 0)
{
    glColor3f(0, 300, 1 / (float)tp_array[ISO][j]
    );
}

```



```

        glPushMatrix();
        glTranslatef(-(float)BOARDX / 2.0f, -(float)
            BOARD_Y / 2.0f, 0);
        glTranslatef(j, BOARD_Y - ISO, 0);
        glTranslatef(0.5, 0.5, 0.5);
        glutSolidSphere(0.1f * ((float)tp_array[ISO][
            j]), 6, 6);
        glPopMatrix();
    }
}

int ISO;
for (ISO = 0; ISO < BOARDX; ISO++)
{
    for (int j = BOARD_Y - 1; j >= BOARD_Y / 2; j--)
    {
        glColor3f(0, 0, 1);
        int call_this = 0;
        glPushMatrix();
        glTranslatef(-(float)BOARDX / 2.0f, -(float)
            BOARD_Y / 2.0f, 0);
        glTranslatef(j, BOARD_Y - ISO, 0);
        glPushMatrix();
        glTranslatef(0.5, 0.5, 0);
        switch (board_array[ISO][j])
        {
            case 4:
                glRotatef(90.0, 0, 0, 1);
            case 3:
                glRotatef(90.0, 0, 0, 1);
            case 2:

```

```

        glRotatef(90.0, 0, 0, 1);
case 1:
        call_this = 1;
        break;
case 6:
        glRotatef(90.0, 0, 0, 1);
case 5:
        call_this = 2;
        break;
case 10:
        glRotatef(90.0, 0, 0, 1);
case 9:
        glRotatef(90.0, 0, 0, 1);
case 8:
        glRotatef(90.0, 0, 0, 1);
case 7:
        call_this = 3;
        break;
    }
    glScalef(1, 1, 0.5);
    glTranslatef(-0.5, -0.5, 0);
    glCallList(list[call_this]);
    glPopMatrix();
    //now put on top
    if (call_this != 0 || board_array[ISO][j] == 11)
    {
        glTranslatef(0, 0, -0.5);
        glCallList(list[4]);
    }
    glPopMatrix();
    if (tp_array[ISO][j] > 0)

```

```

        {
            glColor3f(0, 300, 1 / (float)tp_array[ISO][j]
                );
            glPushMatrix();
            glTranslatef(-(float)BOARDX / 2.0f, -(float)
                BOARDY / 2.0f, 0);
            glTranslatef(j, BOARDY - ISO, 0);
            glTranslatef(0.5, 0.5, 0.5);
            glutSolidSphere(0.1f * ((float)tp_array[ISO][
                j]), 6, 6);
            glPopMatrix();
        }
    }
}
Pac();
}
bool Open(int a, int b)
{
    if (board_array[b][a] > 0)
    {
        return false;
    }
    return true;
}
void RenderScene();
void mykey(unsigned char key, int x, int y)
{
    if (start_timer > 0)
    {
        start_timer--;
    }
}

```

```

}
void specialDown(int key, int x, int y)
{
    if (start_timer > 0)
        start_timer--;
    ckey = key;
    if (key == GLUT_KEY_UP && (int)a - a > -0.1 && angle1 !=
        270)
    {
        if (Open(a, b - 1))
        {
            //w
            animate = true;
            angle1 = 270;
        }
    }
    else if (key == GLUT_KEY_DOWN && (int)a - a > -0.1 &&
        angle1 != 90) // s
    {
        if (Open(a, b + 1))
        {
            animate = true;
            angle1 = 90;
        }
    }
    else if (key == GLUT_KEY_LEFT && (int)b - b > -0.1 &&
        angle1 != 180) //a
    {
        if (Open(a - 1, b))
        {
            animate = true;

```

```

        angle1 = 180;
    }
}
else if (key == GLUT_KEY_RIGHT && (int)b - b > -0.1 &&
        angle1 != 0) //d
{
    if (Open(a + 1, b))
    {
        animate = true;
        angle1 = 0;
    }
}
}
void specialUp(int key, int x, int y)
{
}
void P_Reinit()
{
    a = 13.5;
    b = 23;
    angle1 = 90;
    animate = false;
    Pac();
}
void G_Reinit(void)
{
    start_timer = 3;
    //ghost initial starting positions
    int start_x[4] = {11, 12, 15, 16};
    float ghost_colors[4][3] = {{255, 0, 0}, {120, 240, 120},
                                {255, 200, 200}, {255, 125, 0}};

```

```

for (int i = 0; i < num_ghosts; i++)
{
    ghost[i]→Reinit();
    ghost[i]→x = start_x[i];
    ghost[i]→y = 14;
    ghost[i]→eaten = false;
    ghost[i]→jail_timer = i * 33 + 66;
    ghost[i]→max_speed = 0.1 - 0.01 * (float)i;
    ghost[i]→speed = ghost[i]→max_speed;
    //colorize ghosts
    for (int j = 0; j < 3; j++)
        ghost[i]→color[j] = ghost_colors[i][j] / 255.0f;
}
}

void renderBitmapString(float x, float y, void *font, char *
    string)
{
    char *c;
    glRasterPos2f(x, y);
    for (c = string; *c != '\0'; c++)
    {
        glutBitmapCharacter(font, *c);
    }
}

void Write(char *string)
{
    while (*string)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *string
            ++);
}

void print(char *string)

```

```

{
    while (*string)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *
            string++);
}

//Display Function->This Function Is Registered in
    glutDisplayFunc
void RenderScene()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    //Through Movement->From One End To The Other
    if ((int)a == 27 && (int)b == 14 && angle1 == 0)
    {
        a = 0;
        animate = true;
    }
    else if ((int)(a + 0.9) == 0 && (int)b == 14 && angle1 ==
        180)
    {
        a = 27;
        animate = true;
    }
    //Collision Detection For PacMan
    if (animate)
        Move();
    if (!(Open((int)(a + cos(M_PI / 180 * angle1)),
        (int)(b + sin(M_PI / 180 * angle1)))) &&
        a - (int)a < 0.1 && b - (int)b < 0.1)
        animate = false;
    if (tp_array[(int)(b + 0.5)][(int)(a + 0.5)] == 1)
    {

```

```

        tp_array[(int)(b + 0.5)][(int)(a + 0.5)] = 0;
        pebbles_left--;
        points += 1;
    }
    //Super Pebble Eating
    else if (tp_array[(int)(b + 0.5)][(int)(a + 0.5)] == 3)
    {
        tp_array[(int)(b + 0.5)][(int)(a + 0.5)] = 0;
        pebbles_left--;
        points += 5;
        for (int i = 0; i < 4; i++)
        {
            if (!ghost[i]→eaten)
                ghost[i]→Vulnerable(); //Calls A Function To
                Make Monster Weak
        }
    }
    //All The Pebbles Have Been Eaten
    if (pebbles_left == 0)
    {
        G_Reinit();
        P_Reinit();
        tp_restore();
        points = 0;
        lives = 3;
    }
    if (!gameover)
        Draw();
    for (int d = 0; d < num_ghosts; d++)
    {
        if (!gameover && start_timer == 0)

```



```

ghost[d]→Update();
if (!ghost[d]→in_jail &&
ghost[d]→x - (int)ghost[d]→x < 0.1 && ghost[d]
    ]→y - (int)ghost[d]→y < 0.1)
{
    bool open_move[4];
    //Finding Moves
    for (int ang = 0; ang < 4; ang++)
    {
        open_move[ang] = Open((int)(ghost[d]→x + cos
            (M_PI / 180 * ang * 90)),
                                (int)(ghost[d]→y + sin
                                    (M_PI / 180 * ang *
                                        90)));
    }
    //Chase Pac Man
    if (!ghost[d]→eaten)
    {
        if (ghost[d]→x - (int)ghost[d]→x < 0.1 &&
            ghost[d]→y - (int)ghost[d]→y < 0.1)
            ghost[d]→Chase(a, b, open_move);
    }
    else
    {
        if (ghost[d]→x - (int)ghost[d]→x < 0.1 &&
            ghost[d]→y - (int)ghost[d]→y < 0.1)
            ghost[d]→Chase(13, 11, open_move);
    }
}
if (ghost[d]→in_jail && !(Open((int)(ghost[d]→x +
    cos(M_PI / 180 * ghost[d]→angle)), (int)(ghost[d]

```

```

]->y + sin(M_PI / 180 * ghost[d]->angle))) &&
ghost[d]->jail_timer > 0 && ghost[d]->x - (int)
ghost[d]->x < 0.1 && ghost[d]->y - (int)ghost[d]->
y < 0.1)
{
    ghost[d]->angle = (double)(((int)ghost[d]->angle
        + 180) % 360);
}
if (!gameover && start_timer == 0)
    ghost[d]->Move();
ghost[d]->Draw();
if (!(ghost[d]->eaten))
{
    bool collide = ghost[d]->Catch(a, b);
    //Monster Eats PacMan
    if (collide && !(ghost[d]->edible))
    {
        if (lives > 0)
            lives--;
        if (lives == 0)
        {
            gameover = true;
            lives = 0;

            ghost[d]->game_over();
        }
        else
            P_Reinit();

        d = 4;
    }
}

```

```

        //PacMan Eats Monster And Sends It To Jail
        else if (collide && ((ghost[d]->edible)))
        {
            ghost[d]->edible = false;
            ghost[d]->eaten = true;
            ghost[d]->speed = 1;
        }
    }
}
if (gameover == true)
{
    glColor3f(1, 0, 0);
    renderBitmapString(-5, 0.5, GLUT_BITMAP_HELVETICA_18,
        "GAMEOVER");
}
char tmp_str[40];
glColor3f(1, 1, 0);
glRasterPos2f(10, 18);
sprintf(tmp_str, "Points: %d", points);
Write(tmp_str);
glColor3f(1, 0, 0);
glRasterPos2f(-5, 18);
sprintf(tmp_str, "PACMAN");
print(tmp_str);
glColor3f(1, 1, 0);
glRasterPos2f(-12, 18);
sprintf(tmp_str, "Lives: %d", lives);
Write(tmp_str);
glutPostRedisplay();
glutSwapBuffers();
}

```

```

void create_list_lib ()
{
    //Set Up Maze Using Lists
    list [1] = glGenLists (1);
    glNewList (list [1], GL_COMPILE);
    //North Wall
    glBegin (GL_QUADS);
    glColor3f (0, 0, 1);
    glNormal3f (0.0, 1.0, 0.0);
    glVertex3f (1.0, 1.0, 1.0);
    glVertex3f (1.0, 1.0, 0.0);
    glVertex3f (0.0, 1.0, 0.0);
    glVertex3f (0.0, 1.0, 1.0);
    glEnd ();
    glEndList ();
    list [2] = glGenLists (1);
    glNewList (list [2], GL_COMPILE);
    glBegin (GL_QUADS);
    //North Wall
    glColor3f (0, 0, 1);
    glNormal3f (0.0, 1.0, 0.0);
    glVertex3f (1.0, 1.0, 1.0);
    glVertex3f (1.0, 1.0, 0.0);
    glVertex3f (0.0, 1.0, 0.0);
    glVertex3f (0.0, 1.0, 1.0);
    //South Wall
    glColor3f (0, 0, 1);
    glNormal3f (0.0, -1.0, 0.0);
    glVertex3f (1.0, 0.0, 0.0);
    glVertex3f (1.0, 0.0, 1.0);
    glVertex3f (0.0, 0.0, 1.0);

```

```

glVertex3f(0.0, 0.0, 0.0);
glEnd();
glEndList();
list[3] = glGenLists(1);
glNewList(list[3], GL_COMPILE);
glBegin(GL_QUADS);
//North Wall
glColor3f(0, 0, 1);
glNormal3f(0.0f, 1.0f, 0.0f);
glVertex3f(1.0, 1.0, 1.0);
glVertex3f(1.0, 1.0, 0.0);
glVertex3f(0.0, 1.0, 0.0);
glVertex3f(0.0, 1.0, 1.0);
//East Wall
glColor3f(0, 0, 1);
glNormal3f(1.0, 0.0, 0.0);
glVertex3f(1.0, 1.0, 0.0);
glVertex3f(1.0, 1.0, 1.0);
glVertex3f(1.0, 0.0, 1.0);
glVertex3f(1.0, 0.0, 0.0);
glEnd();
glEndList();
list[4] = glGenLists(1);
glNewList(list[4], GL_COMPILE);
glBegin(GL_QUADS);
//Top Wall
glColor3f(-1, 0.3, 0);
glNormal3f(1.0, 0.0, 1.0);
glVertex3f(1, 1, 1.0);
glVertex3f(0, 1, 1.0);
glVertex3f(0, 0, 1.0);

```

```

        glVertex3f(1, 0, 1.0);
        glEnd();
        glEndList();
    }
    void init()
    {

        glEnable(GL_NORMALIZE);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(60, 1.33, 0.005, 100);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(-1.5, 0, 40, -1.5, 0, 0, 0.0f, 1.0f, 0.0f);
    }
    void erase()
    {

        glColor3f(0.1, 0.0, 0.0);
        glBegin(GL_POLYGON);
        glVertex2f(0, 0);
        glVertex2f(0.5, 0);
        glVertex2f(0.25, 0.5);
        glEnd();
    }
    int main(int argc, char **argv)
    {

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
            ;
        glutInitWindowSize(1200, 780);
        glutInitWindowPosition(0, 0);
    }

```

```

glutCreateWindow("Pac_GL_3D");
init();
glutDisplayFunc(RenderScene);
create_list_lib();
glutKeyboardFunc(mykey);
glutSpecialFunc(specialDown);
glutSpecialUpFunc(specialUp);
glEnable(GL_DEPTH_TEST);
int start_x[4] = {11, 12, 15, 16};
for (int ISO = 0; ISO < num_ghosts; ISO++)
{
    ghost[ISO] = new Ghost(start_x[ISO], 14);
}
float ghost_colors[4][3] = {{255, 0, 0}, {120, 240, 120},
    {255, 200, 200}, {255, 125, 0}};
int ISO;
for (ISO = 0; ISO < num_ghosts; ISO++)
{
    ghost[ISO]->x = start_x[ISO];
    ghost[ISO]->y = 14;
    ghost[ISO]->eaten = false;
    ghost[ISO]->max_speed = 0.1 - 0.01 * (float)ISO;
    ghost[ISO]->speed = ghost[ISO]->max_speed;
    //colorize ghosts
    for (int j = 0; j < 3; j++)
        ghost[ISO]->color[j] = ghost_colors[ISO][j] /
            255.0f;
}
for (ISO = 0; ISO < BOARD_X; ISO++)
{
    for (int j = 0; j < BOARD_Y; j++)

```

```

        {
            tp_array[ISO][j] = pebble_array[ISO][j];
        }
    }
    pebbles_left = 244;
    glShadeModel(GLSMOOTH);
    glutMainLoop();
    return 0;
}

```