

VISVESVARAYA TECHNOLOGICAL UNIVERSITY  
BELAGAVI- 590018



**A COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT (18CSL67)**  
**SYNOPSIS ON**

**“Paccman Game”**

Submitted By

<b>Mr. Swapnadeep Kapuri</b>	<b>[2BU20CS098]</b>
<b>Mr. Sushant Hakare</b>	<b>[2BU20CS097]</b>
<b>Mr. Sunil Biradar</b>	<b>[2BU20CS096]</b>
<b>Mr. Sammed B.</b>	<b>[2BU20CS071]</b>

Under the Guidance of

**Ms. Nalinakshi B. G.**

**Assistant Professor**

Department of Computer Science and Engineering  
S.G.Balekundri Institute of Technology, Shivabasav Nagar  
Belagavi-10



**S.S EDUCATION TRUST's**

**S.G Balekundri Institute of Technology, Shivabasava Nagar Belagavi-10**

*An ISO 21001:2018 certified institution,*

**Department of Computer Science and Engineering.**

*Accredited by NBA*

*2022-23*

**S.S.E.T'S**

**S.G. BALEKUNDRI INSTITUTE OF TECHNOLOGY**

*An ISO 21001:2018 certified institution*

**SHIVABASAVA NAGAR, BELAGAVI-590010**



**DEPARTMENT OF  
COMPUTER SCIENCE & ENGINEERING**

**Accredited by NBA**

**CERTIFICATE**

This is to certify that the project work titled **Paccman Game** is a Bonafede work satisfactorily completed by **Mr. Swapnadeep Kapuri (USN:2BU20CS098)**, **Mr. Sushant Hakare (USN:2BU20CS097)**, **Mr. Sunil Biradar (USN:2BU20CS096)** and **Mr. Sammed B. (USN:2BU20CS071)**, in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering under Visvesvaraya Technological University, Belagavi, for the year 2022-2023. 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.

Course Coordinator  
**Ms. Nalinakshi B. G.**

HOD  
**Dr. B. S. Halakarnimath**

Principal  
**Dr. B. R. Patagundi**

**External Viva**

Name of the Examiner's

Signature with Date

- 1)
- 2)

## **ACKNOWLEDGEMENT**

It is our proud privilege and duty to acknowledge the kind help and guidance received from several persons in preparation of this Mini Project Report. It would not have been possible to prepare this report in this form without their valuable help, cooperation and guidance.

First and the foremost, we wish to record our sincere gratitude to Management of this college and to our beloved Professor, Dr. B.R. Patagundi, Principal, S. G. Balekundri Institute of Technology, Belagavi for his constant support and encouragement in preparation of this report and for making available library and laboratory facilities needed to prepare this report.

Our sincere thanks are also due to Dr. B.S. Halakarnimath. Head, Department of Computer Science and Engineering in S.G.B.I.T. for the valuable suggestions and guidance through the period of preparation of this report.

We express our sincere gratitude to our beloved guide Mrs. Nalinakshi B. G, Assistant Professor in the Dept. of Computer Science and Engineering S.G.B.I.T., Belagavi for guiding us in investigations for this mini project. Our numerous discussions with him were extremely helpful. We hold her in esteem for guidance, encouragement and inspiration received from her.

Belagavi

Date: 08/07/2023

Mr. Swapnadeep Kapuri

Mr. Sushant Hakare

Mr. Sunil Biradar

Mr. Sammed B.

# **Content**

## **Chapter 1: Introduction**

1.1 Objectives of the Project

## **Chapter 2: System Requirements**

2.1 Software requirements

2.2 Hardware requirements

## **Chapter 3: Design**

3.1 Flow Chart

3.2 Working Principle

## **Chapter 4: Implementation**

4.1 Coding

## **Chapter 5: Snapshots**

**Conclusion**

**References**

## LIST OF FIGURES

<b>Figure Number</b>	<b>Figure Names</b>	<b>Page Number</b>
3.1	Data Flow Diagram	4
4.2.1	Terminal Input	37
4.2.2	Start of the game	37
4.2.3	Game Flow	38
4.2.4	End of the Game	38

## Chapter 1

# INTRODUCTION

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 paku-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 and taberu means it is eating in Japanese. 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.

In North America, the game was released by Midway Manufacturing as part of its licensing agreement with Namco America. The player controls Pac-Man, who must eat all the dots inside an enclosed maze while avoiding four colored ghosts. Eating large flashing dots called "Power Pellets" causes the ghosts to temporarily turn blue, allowing Pac-Man to eat them for bonus points.

The project helps to emulate the once popular game, using OpenGL and using C++ compiler technology. Pac-Man is an action maze chase video game; the player controls the eponymous character through an enclosed maze. The objective of the game is to eat all of the dots placed in the maze while avoiding four colored ghosts — Blinky (red), Pinky (pink), Inky (cyan), and Clyde (orange) — that pursue Pac-Man. The game remains one of the highest-grossing and best-selling games, generating more than \$14 billion in revenue (as of 2016) and 43 million units in sales combined, and has an enduring commercial and cultural legacy, commonly listed as one of the greatest video game of all time.

## 1.1 Objective of Mini project

The objective of the Pac-Man game is to control a character, known as Pac-Man, and guide it through a maze-like environment. The primary goal is to eat all the small white dots, called Pac-Dots, that are scattered throughout the maze while avoiding four colorful ghosts that roam the maze.

Additional objectives in the game include:

1. Eating larger flashing dots, called Power Pellets, which temporarily grant Pac-Man the ability to turn the tables on the ghosts. When Pac-Man eats a Power Pellet, the ghosts turn blue and become vulnerable, allowing Pac-Man to eat them for extra points.
2. Accumulating points by eating Pac-Dots, ghosts (when vulnerable), fruit, and other bonus items that may appear in the maze.
3. Navigating the maze strategically to avoid getting trapped by the ghosts and losing one of Pac-Man's lives. Pac-Man starts with a limited number of lives, typically three, and loses one each time it is caught by a ghost.
4. Advancing to the next level by eating all the Pac-Dots in the current maze. As the levels progress, the difficulty increases, with faster ghosts and more complex mazes.
5. Achieving a high score by accumulating points throughout the game. Pac-Man has a scoring system that rewards various actions, such as eating ghosts, eating fruit, and completing levels quickly.

The ultimate objective is to maximize your score and progress as far as possible through the game's increasingly challenging levels while enjoying the classic arcade experience of Pac-Man.

## Chapter 2

# SYSTEM REQUIREMENTS

### 2.1 Hardware Requirements

- Computer processor capable of running Open GL application. Minimum 2 GB RAM.
- Sufficient RAM to handle the execution of the program.
- Graphic Card capable of supporting Open GL graphics rendering.
- Display Monitor which supports at least HD quality image rendering.

### 2.2 Software Requirements

- Operating System: WINDOWS and LINUX.
- Open GL Libraries: Open GL utility tool kit (GLUT) and free GLUT3.
- C and C++ Compiler: GCC (GNU Compiler Collection) for LINUX, MinGW for Windows.
- IDE: CodeBlocks (For Windows) and GEDIT (For Linux)



## Chapter 3

# DESIGN

### 3.1 Flow Chart

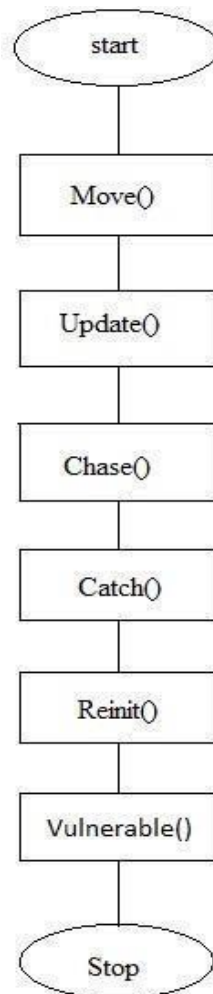


Figure 3.1 Dataflow diagram

### 3.2 Working Principle

The working principle of the Pac-Man game involves several key components and mechanics that work together to create an interactive and entertaining gameplay experience. Here's a breakdown of the main elements and their functioning:

1. **Maze Generation:** The game starts by generating a maze, which consists of walls that form a grid-like structure. The maze layout is typically predefined or procedurally generated, providing a structured environment for Pac-Man and the ghosts to navigate.
2. **Game Objects:**
  - **Pac-Man:** The player controls Pac-Man, a circular character that moves through the maze. Pac-Man's movement is controlled by user input, typically using arrow keys or a joystick.
  - **Ghosts:** The game features multiple ghosts that move independently and have distinct behaviors. Each ghost may have different movement patterns, such as chasing Pac-Man, patrolling specific areas, or attempting to surround Pac-Man strategically.
  - **Pac-Dots:** Small white dots, scattered throughout the maze, represent Pac-Dots. Pac-Man's objective is to eat all Pac-Dots to complete the level.
  - **Power Pellets:** Larger flashing dots, also located in the maze, are Power Pellets. When Pac-Man eats a Power Pellet, the ghosts temporarily become vulnerable, allowing Pac-Man to eat them for extra points.
  - **Fruit and Bonus Items:** Occasionally, bonus items such as fruit or other objects may appear in the maze, offering additional points when collected.
3. **Collision Detection:** The game checks for collisions between different objects to determine the outcome of interactions. For example:
  - When Pac-Man collides with a Pac-Dot, it is removed from the maze, and Pac-Man's score increases.
  - If Pac-Man collides with a ghost while it is vulnerable (after eating a Power Pellet), the ghost is captured, and Pac-Man's score increases.
  - If Pac-Man collides with a ghost while it is in a non-vulnerable state, Pac-Man loses a life, and the game may either end if no lives remain or reset Pac-Man and the ghosts to their initial positions.

4. **AI Behavior:** Each ghost typically has its own AI behavior to determine its movement and decision-making. The AI algorithms may involve strategies such as chasing Pac-Man, predicting Pac-Man's movements, or attempting to trap Pac-Man strategically.
5. **Scoring and Levels:** The game keeps track of the player's score, which increases when Pac-Man eats Pac-Dots, captures ghosts, collects bonus items, or achieves specific milestones. As the player progresses, they advance through different levels, which may feature faster ghosts, more complex mazes, and additional challenges.
6. **User Input and Controls:** The player controls Pac-Man's movement using input devices such as arrow keys, a joystick, or touch controls. The game interprets the input and updates Pac-Man's position accordingly.
7. **Rendering and Display:** The game renders the maze, Pac-Man, ghosts, and other objects onto the screen. This visual representation provides real-time feedback to the player, allowing them to navigate the maze and observe the game's state.
8. **Game Loop:** The game continuously runs a loop that handles updating the game's state, detecting collisions, processing user input, updating object positions, and rendering the display. This loop ensures smooth gameplay and interactivity.

By combining these elements and principles, the Pac-Man game creates an engaging experience where players navigate a maze, eat Pac-Dots, avoid ghosts, and strive to achieve high scores.

## Chapter 4

# IMPLEMENTATION

### 4.1 Coding

```
#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] =

{{8,5,5,5,5,5,5,5,5,5,5,5,1,1,5,5,5,5,5,5,5,5,5,7},

{6,0,0,0,0,0,0,0,0,0,0,0,2,4,0,0,0,0,0,0,0,0,0,6},

{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},

{6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6},

{6,0,8,1,1,7,0,8,7,0,8,1,1,1,1,1,7,0,8,7,0,8,1,1,7,0,6},

{6,0,9,3,3,10,0,2,4,0,9,3,3,11,11,3,3,10,0,2,4,0,9,3,3,10,0,6},

{6,0,0,0,0,0,0,2,4,0,0,0,0,2,4,0,0,0,0,2,4,0,0,0,0,0,6},

{9,5,5,5,5,7,0,2,11,1,1,7,0,2,4,0,8,1,1,11,4,0,8,5,5,5,10},
```

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

```
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},
```



```
{0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0},  
{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},  
{0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0},  
{0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0},  
{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)) {

        //wanimate = 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 ifanimate&&ckey==GLUT_KEY_RIGHT&& (int) b - b > -0.1 && angle1 != 0) {

    if (Open(a+1,b)) {

        animate = true;

        angle1 = 0;

    } } }

void Pac(void) {

    glColor3f(0,1,1);

    glPushMatrix();
```



```
    glTranslatef(a,-b,0);

    glTranslatef(0.5,0.6,0);

    glTranslatef((float)BOARD_X/-2.0f,(float)BOARD_Y/2.0f,0.5);

    glutSolidSphere(0.5,15,10);

    glPopMatrix();

}

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;

    }

    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--;

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 (jail_timer == 0 && in_jail) {

        if (x < 13)

            angle = 0;

        if (x > 14)

            angle = 180;

    }

    if (jail_timer > 0)

        jail_timer--;

    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) {  
  
    if (px - x < 0.2 && px - x > -0.2 && py - y < 0.2 && py - y > -0.2) {  
  
        return true;  
  
    }  
  
    return false;  
  
}
```

```
void Ghost::Vulnerable(void) {  
  
    if (!(edible)) {  
  
        angle = ((int)angle + 180)%360;  
  
        speed = max_speed;  
  
    }  
  
    edible = true;  
  
    edible_timer = edible_max_time;  
  
}
```

```
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;

    }

    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)BOARD_X/-2.0f, (float)BOARD_Y/2.0f,0.5);

    glutSolidSphere(.5,10,10);

    glPopMatrix();

}

void tp_restore(void) {

    for (int 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;

}
```



```
void Draw(void) {

    glColor3f(1,0,1);

    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();

    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) BOARD_X / 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< BOARD_X; ISO++)  {

    for (int j = BOARD_Y-1; 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();

    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) BOARD_X / 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();

    } } }

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)) {

            animate = true;

            angle1 = 270;

        } }

    else if(key==GLUT_KEY_DOWN&& (int) a - a > -0.1 && angle1 != 90) {

        if (Open(a,b + 1)) {

            animate = true;

            angle1= 90;

        } }

}
```

```
else if(key==GLUT_KEY_LEFT&& (int) b - b > -0.1 && angle1 != 180) {

    if (Open(a-1,b)) {

        animate = true;

        angle1 = 180;

    } }

else if(key==GLUT_KEY_RIGHT&& (int) b - b > -0.1 && angle1 != 0) {

    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;

    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;

    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++);

}

void RenderScene() {

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

    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;

    }

    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;

    }

}
```



```
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

    } }

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];
```

```
        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)));

        }

        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);

    if (collide && !(ghost[d]->edible)) {
```

```
        lives--;

        if (lives == 0) {

            gameover = true;

            lives=0;

            ghost[d]->game_over();

        }

        P_Reinit();

        d = 4;

    }

    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 ,"GAME OVER");

}

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, "PAC MAN");

    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() {

    list[1] = glGenLists(1);

    glNewList(list[1], GL_COMPILE);

    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);

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);

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);

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);

        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);

        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()
```

```
{

    float color[4];

    glEnable(GL_LIGHT0);

    glEnable(GL_LIGHTING);

    glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

    glEnable(GL_COLOR_MATERIAL);

    color[0] = 1.0f; color[1] = 1.0f; color[2] = 0.0f; color[3] = 0.0f;

    glLightfv(GL_LIGHT0, GL_DIFFUSE, color);

    color[0] = 1.0f; color[1] = 0.0f; color[2] = 1.0f; color[3] = 1.0f;

    glLightfv(GL_LIGHT0, GL_AMBIENT, color); //

    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);

    }

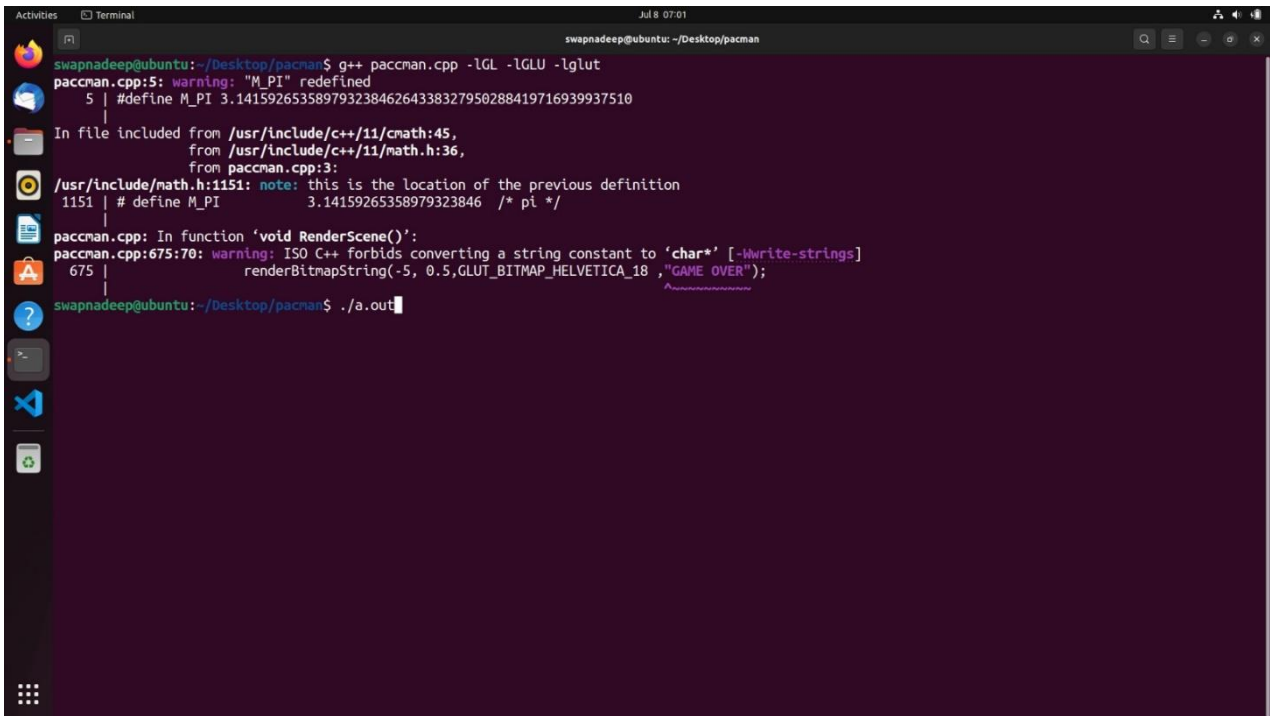
    float ghost_colors[4][3] = {{ 255,0,0},{ 120,240,120},{ 255,200,200},{ 255,125,0} };

    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;  
  
        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(GL_SMOOTH);  
  
    glutMainLoop();  
  
    return 0;  
  
}
```

## 4.2 Snapshot



```
swapnadeep@ubuntu: ~/Desktop/pacman$ g++ pacman.cpp -lGL -lGLU -lglut
pacman.cpp:5: warning: "M_PI" redefined
5 | #define M_PI 3.14159265358979323846264338327950288419716939937510
  |
In file included from /usr/include/c++/11/cmath:45,
                 from /usr/include/c++/11/math.h:36,
                 from pacman.cpp:3:
/usr/include/math.h:1151: note: this is the location of the previous definition
1151 | # define M_PI 3.14159265358979323846 /* pi */
     |
pacman.cpp: In function 'void RenderScene()':
pacman.cpp:675:70: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
675 |         renderBitmapString(-5, 0.5, GLUT_BITMAP_HELVETICA_18, "GAME OVER");
     |
swapnadeep@ubuntu: ~/Desktop/pacman$ ./a.out
```

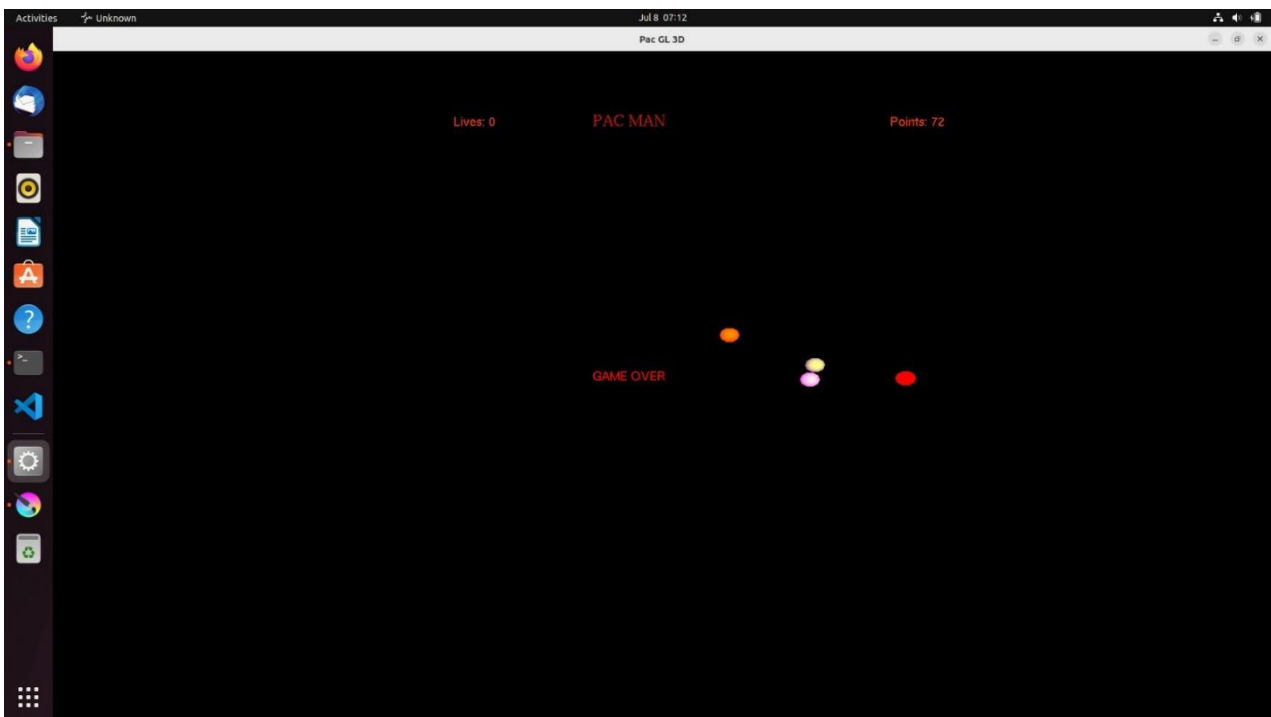
Figure 4.2.1: Terminal Input showing the necessary keywords to input the program



Figure 4.2.2: Starting of the game.



**Figure 4.2.3:** Game flow of the game. The Pac-Man must avoid the four Ghosts and collect the stones to gain points.



**Figure 4.2.4:** Game gets over when Pac-Man gets in contact with the Ghosts for more than 3 times.

## Chapter 5

# CONCLUSION

In conclusion, the Pac-Man project developed in OpenGL using the GNU-based C++ compiler in Ubuntu has successfully implemented a playable version of the classic arcade game. The project involved creating a graphical environment, designing game objects, handling user input, implementing game logic, and providing visual and audio feedback.

Throughout the project, various OpenGL features and functionalities were utilized to render the game's graphics and animations. The game environment was built using 2D shapes and textures, allowing for a visually appealing representation of the maze, Pac-Man, and the ghosts. Interaction with the game was achieved through keyboard input, enabling players to control Pac-Man's movement and navigate through the maze.

The game's logic included the implementation of ghost AI, collision detection, and score tracking. The ghosts' behavior followed specific patterns and strategies, creating an engaging and challenging gameplay experience. Collision detection algorithms ensured that Pac-Man and the ghosts interacted appropriately when colliding with each other or with the maze's walls. The game also tracked the player's score, updating it whenever Pac-Man collected pellets or ate the ghosts.

Overall, the Pac-Man project successfully combined the power of OpenGL with C++ programming to recreate the beloved classic game in a modern environment. It showcased the ability to create a functional and enjoyable gaming experience while leveraging the GNU-based C++ compiler in Ubuntu. This project serves as a testament to the capabilities of OpenGL and the flexibility of the GNU toolchain for game development purposes.

## REFERENCES

1. The Red Book-OpenGL programming Guide, 6<sup>th</sup> edition
2. Edward Angel Interactive Computer Graphics A Top-Down Approach with OpenGL, 5<sup>th</sup> edition, Addison and Wesley
3. Website: <https://en.wikipedia.org/wiki/Pac-Man>
4. Website: <https://learnopengl.com/Getting-started/Introduction>
5. Website: <https://www.geeksforgeeks.org/computer-graphics-programming-in-opengl-with-c/>