The Pacman Game Using Java

A PROJECT REPORT

Submitted by

ABHISHEK PANDEY: 23BCS14112
SRISHTI MISHRA: 23BCS13857
AAKASH ARORA: 23BCS10919
UJJWAL PATHAK: 23BCS10936

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE
ENGINEERING

2025

BONAFIDE CERTIFICATE

Certified that this project report **"The Pacman Game Using Java"** is the Bonafide work of Srishti Mishra (23BCS13857), Abhishek Pandey (23BCS14112), Aakash Arora (23BCS10919), Ujjwal Pathak(23BCS10936) who carried out the project work under my/our supervision.

SIGNATURE
DR. SANDEEP SINGH

HEAD OF DEPARTMENT
BE-CSE

SIGNATURE
ARSHDEEP GROVER

SUPERVISOR
BE-CSE

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Introduction to Project

The **Pacman Game using Java** is a desktop-based application designed to recreate the classic Pacman arcade experience.
It is built entirely with **core Java**, utilizing **AWT and Swing** libraries for rendering the game window, handling user input, and controlling game logic.

The project focuses on implementing smooth character movement, collision detection, score tracking, and level progression all while maintaining simplicity and efficiency. The game provides a fun, interactive way to demonstrate **object-oriented programming (OOP)** principles such as encapsulation, inheritance, and event-driven design.

Key Technical Highlights:

- **Programming Language:** Java (JDK 8+)
- **Libraries Used:** Java AWT and Swing
- **IDE:** Visual Studio Code / IntelliJ / Eclipse
- **Assets:** PNG images for ghosts, Pacman, and collectibles
- **Database:** SQLite (for saving high scores and player data)

This project showcases how traditional 2D arcade mechanics can be recreated through logical structuring, event handling, and graphical rendering in Java.

1.2 Identification of Problem

The need for this project arises from the desire to understand and demonstrate **fundamental game development logic** without relying on advanced engines or frameworks.
Existing game engines abstract most of the core functionalities, leaving little room to explore how games actually function internally.

Problems addressed by this project:

- Lack of foundational understanding of how 2D games handle **movement, collision, and rendering**.
- Difficulty in visualizing real-time **player-object interaction** using raw Java code.
- Limited awareness among students of **how OOP principles apply to real-time systems**.
- Need for a project that combines **logic, creativity, and visual feedback**.

The Pacman Game bridges this gap by allowing users to interact with a visual, logic-driven environment coded purely in Java.

CHAPTER 2: BACKGROUND STUDY

2.1 Existing Solutions

Many existing Pacman implementations exist, ranging from simple command-line recreations to advanced Unity or Unreal Engine versions. However, those often:

- Depend heavily on third-party game engines or frameworks.
- Abstract away the actual logic of pathfinding, collisions, and rendering.
- Are complex and resource-intensive for beginners to understand.

This project differentiates itself by being **lightweight, engine-free, and educational**, giving learners direct control over every aspect of the gameplay logic.

2.2 Problem Definition

The problem is to design and implement a **fully functional Pacman clone** using **only core Java**, with all gameplay mechanics (ghost movement, pellet collection, score updates) coded manually.

The system should:

- Display a playable 2D grid-based map.
- Control Pacman movement via keyboard inputs.
- Detect collisions between Pacman, ghosts, and walls.
- Track and display the player's score dynamically.
- Maintain game state (running, paused, game over).

2.3 Goals/Objectives

The main objectives of this project are:

1. To develop an interactive 2D game entirely using Java.
2. To implement classic arcade gameplay mechanics without external libraries.
3. To apply OOP concepts in designing modular and reusable code.
4. To demonstrate event-driven programming via keyboard listeners.
5. To manage assets like sprites and icons effectively within Java's resource structure.
6. To store and retrieve player high scores using an SQLite database.

CHAPTER 3: DESIGN FLOW / PROCESS

3.1 Evaluation & Selection of Specifications/Features

Core Features:

- Player-controlled Pacman that can move in four directions.
- Intelligent ghosts with movement logic (random or semi-predictive).
- Score counter that increases when Pacman eats pellets.
- "Game Over" condition when a ghost collides with Pacman.
- Restart and Exit options for continuous gameplay.

Technology Stack:

- **Language:** Java
- **UI Framework:** AWT/Swing
- **Database:** SQLite (optional for saving high scores)
- **Graphics:** PNG image assets for characters and background

3.2 Analysis of Features and Finalization

After testing multiple implementations, the following design decisions were finalized:

- **AWT/Swing** chosen for its native rendering support and event handling.
- **Thread-based game loop** implemented for continuous frame updates.
- **2D grid system** used for managing movement and collision detection.
- **Class structure** divided into:
    - `App.java` – Main entry point launching the game window.
    - `PacMan.java` – Core logic controlling rendering, collisions, and events.
    - Inner classes (e.g., `Block`, `Ghost`) – Encapsulate specific behaviors.

3.3 Design Flow

User Interaction Flow:

1. User launches the game (`App.java` executes).
2. Game window opens, displaying Pacman and maze layout.
3. Player uses arrow keys to move Pacman.
4. Pacman collects pellets while avoiding ghosts.
5. Score increases with each pellet collected.
6. On collision with a ghost → "Game Over" screen displayed.
7. User can restart the game or exit the program.

This structure provides a seamless real-time experience while maintaining low CPU overhead through efficient thread management.

CHAPTER 4: RESULTS ANALYSIS AND VALIDATION

4.1 Implementation of Solution

Frontend (Graphics/UI):

- Built using **Java Swing** components such as `JFrame`, `JPanel`, and `Graphics2D`.
- Game assets (PNG images) loaded dynamically and scaled to fit the window.
- Double-buffering used for smooth animations and flicker-free rendering.
- Keyboard inputs captured using `KeyListener` for responsive controls.

Backend (Logic/State Management):

- `PacMan.java` handles game logic, collision detection, and entity management.
- Separate threads manage ghost movement and screen refresh rate.
- SQLite used optionally to store and display top scores.
- The game loop constantly checks player status, collisions, and updates the interface.

Outcome:

- Fully functional **2D Pacman Game** that runs independently.
- Clean graphics and responsive gameplay.
- Demonstrates practical OOP implementation in Java.
- Can be extended with new maps, sound effects, and smarter AI.

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 Conclusion

The **Pacman Game using Java** successfully replicates the core features of the classic arcade version while serving as a learning project for event-driven and graphical programming in Java.
It combines logical structuring, interactive gameplay, and real-time rendering to create a simple yet effective demonstration of Java's graphical capabilities.

This project highlights how **object-oriented design** and **thread management** can be used to build interactive systems even without specialized engines or external libraries.

5.2 Future Work

To further enhance the project, the following improvements can be implemented:

- **Add sound effects and background music** for better immersion.
- **Introduce multiple levels** with increasing difficulty and maps.
- *Implement AI pathfinding (A or BFS)** for smarter ghost movement.
- **Add power-ups** like "Super Pellet" enabling Pacman to eat ghosts temporarily.
- **Create a high-score leaderboard UI** integrated with SQLite.
- **Package as an executable JAR** for easier distribution.
- Accessibility for mobile devices.