Jacob Stewart

COP-2939-86403

Instructor: Dr. Banisakher

Date: March 22, 2024

Detailed Design

CAPSTONE: PACMAN IN JAVA

Table of Contents

[1. Overview 2](#_Toc161990428)

[1.1 Basic Structure 2](#_Toc161990429)

[1.1.1 Game Loop: 2](#_Toc161990430)

[1.1.2 Game State: 2](#_Toc161990431)

[1.1.3 Entities: 2](#_Toc161990432)

[1.1.4 Collision Detection: 3](#_Toc161990433)

[1.1.5 Input Handling: 3](#_Toc161990434)

[1.1.6 Rendering: 3](#_Toc161990435)

[1.1.7 Map Design: 3](#_Toc161990436)

[1.1.8 Game Rules: 3](#_Toc161990437)

[1.1.9 Score Keeping: 3](#_Toc161990438)

[1.1.10 Sound Effects: 3](#_Toc161990439)

[2. Program Packages 4](#_Toc161990440)

[2.1. Main Package 4](#_Toc161990441)

[2.2 Collision Package 4](#_Toc161990442)

[2.3 AI Package 4](#_Toc161990443)

[2.4 Actors Package 4](#_Toc161990444)

[2.5 Objects Package 5](#_Toc161990445)

[2.6 Tiles Package 6](#_Toc161990446)

[2.7 UI Package 6](#_Toc161990447)

[2.8 Utils Package 6](#_Toc161990448)

[2.8 Sound Package 6](#_Toc161990449)

[3. Frameworks and Software 6](#_Toc161990450)

[3.1 Frameworks 6](#_Toc161990451)

[3.2 Software 7](#_Toc161990452)

[4. Detailed User Interface 7](#_Toc161990453)

# Overview

The purpose of this document is to present the detailed design for pacman in java. The systems architecture revolves around its core game loop that is modified using a basic state machine to record which section of the game is currently in play (i.e menu, game, death, loading). Each entity in the game is responsible for its update and render loops when is then passed into the core game loop.

## Basic Structure

### 1.1.1 Game Loop:

The game loop controls the flow of the game, updating the game state and rendering it to the screen. It consists of a loop that repeatedly performs the following steps:

* Process user input.
* Update the game state based on the input and current game logic.
* Render the game state to the screen.

### 1.1.2 Game State:

Represents the current state of the game, including the positions and states of all entities (Pac-Man, ghosts, pellets, etc.). It also includes the current score, level, and lives.

### 1.1.3 Entities:

Each entity in the game (Pac-Man, ghosts, pellets, etc.) is represented by a class. Entities have properties such as position, direction, and state, as well as methods for updating their state and rendering them to the screen. Entities also have collision detection logic to detect collisions with other entities.

### 1.1.4 Collision Detection:

Handles collision detection between entities, such as between Pac-Man and a ghost, or between Pac-Man and a pellet. Uses rectangle objects to create algorithms to determine when entities collide notably using Rectangles built in Intersect method.

### Input Handling:

Captures user input, such as keypresses, to control Pac-Man's movements. Converts user input into commands that update the game state (e.g., moving Pac-Man up, down, left, or right).

### 1.1.6 Rendering:

Responsible for rendering the game graphics to the screen. Uses a rendering library (e.g., JavaFX, Swing) to draw the game state (entities, maze, score) to the screen.

### 1.1.7 Map Design:

Represents the game levels, including the layout of the maze, the placement of pellets and power pellets, and the starting positions of Pac-Man and ghosts. Uses data structures (e.g., arrays, lists) to represent the maze layout recorded from a text file.

### 1.1.8 Game Rules:

Enforces the game rules, such as what happens when Pac-Man eats a pellet or collides with a ghost. Controls the game progression, such as advancing to the next level when all pellets are eaten.

### 1.1.9 Score Keeping:

Tracks the player's score and updates it based on game events (e.g., eating pellets, eating ghosts). Displays the score to the player on the game screen.

### 1.1.10 Sound Effects:

Plays sound effects to enhance the gaming experience (e.g., eating pellets, colliding with ghosts). Uses a sound library (e.g., Java Sound API) to play the sound effects.

## Program Packages

### 2.1. Main Package

#### 2.1.1 Main.java

This class serves as the entry point into the application, window setup, thread starting, and game setup called here to initialize the game loop.

#### 2.1.2 InputHandler.java

This class is responsible for taking in key presses and updating boolean values to control UI navigation and Pacman movement.

#### 2.1.3 Engine.java

This class will act as the Game Engine, it is responsible for supplying the Jframe and controlling the update and render methods. Holds references and acts as the main hub for each class to interact with necessary components. The engine is also responsible for distinguishing what is updated and rendered depending on the current state of the game.

### 2.2 Collision Package

#### 2.2.1 CollisionManager.java

This class provides the methods required for each actor, object, and tile to control and check its collision within the game space. It provides methods for all actors to detect tile collisions, Pacman to detect object collisions, and Ghosts to detect collisions with Pacman.

### 2.3 AI Package

#### 2.3.1 Node.java

This class serves to act as a single node, it tracks its individual costs, position, and collision status. The node tracks if it was previously checked or open to provide accurate points for the A\* algorithm to process routes.

#### 2.3.2 PathFinder.java

This class handles pathfinding. It is responsible for setting nodes, finding the cost, and searching for the best path to follow towards the target destination. This class follows the standard implementation of the A\* algorithm with modifications to correctly identify the collision methods set.

### 2.4 Actors Package

#### 2.4.1 Actor.java

The superclass provides the shared variables that all actors need to contain. Simply put it is the blueprint for creating an Actor.

#### 2.4.2 Pacman.java

This class is responsible for rendering, updating, and utilizing controls set in the input handler to control Pacman and its collisions. Pacman also controls its death and loss conditions as well as provides the API to control ghost states depending on how many pellets have been picked up during the active level.

#### 2.4.3 RedGhost.java

This class is responsible for the rendering, updating and state control of the red ghost. This ghost has a base speed half the speed of Pacman. It’s main responsibility is to be the silent killer, it tracks Pacmans’ exact position and proceeds to slowly stalk the player.

#### 2.4.4 PinkGhost.java

This class is responsible for rendering, updating and state control of the pink ghost. This ghost has a base speed equivalent to Pacman. The speed makes it slightly more threatening as it also tracks Pacmans exact position however without care is prone to cut-off and corner the player.

#### 2.4.5 BlueGhost.java

This class is responsible for the rendering, updating and state control of the blue ghost. This ghost has a base speed half the speed of Pacman. Its main responsibility is to provide randomness to the game. Its search algorithm uses the same methods as the other ghosts however its target tile is found randomly. While not a threatening ghost, its random movement tends to cause havoc once all 4 ghosts are released.

#### 2.4.6 OrangeGhost.java

This class is responsible for the rendering, updating, and state control of the orange ghost. This ghost has a base speed half the speed of Pacman. Its main responsibility is to poke at the player. Instead of following Pacman directly, it follows 1 tile ahead of the player. This creates the illusion that if the player confronts the ghost in a game of chicken the ghost will back off, however not confronting the ghost tends to result in it cutting off the player.

### 2.5 Objects Package

#### 2.5.1 SuperObject.java

This superclass represents data shared between each object in the game.

#### 2.5.2 ObjectManager.java

This class is responsible for creating objects based on the map data and adding them to the engine's object array for rendering.

#### 2.5.3 OBJ\_Small\_Pellet.java

This class is used to create and instance of the small pellet, containing its unique name and buffered image.

#### 2.5.4 OBJ\_Large\_Pellet.java

This class is used to create and instance of the large pellet, containing its unique name and buffered image.

#### 2.5.5 OBJ\_TPLeft.java

This class is used to create and instance of the TP Left object, containing its unique name and buffered image and hitbox.

#### 2.5.6 OBJ\_TPRight.java

This class is used to create and instance of the TP Right object, containing its unique name and buffered image and hitbox.

### 2.6 Tiles Package

#### 2.6.1 Tile.java

This class is represents a single tile and will hold collision and image values.

#### 2.6.2 TileManager.java

This class is responsible for setting up tiles image and collision, reading data from a text file to a Tile array before rendering the map. It reads in the tile images and loads the map accordingly.

### 2.7 UI Package

#### 2.7.1 UI.java

This class will act as the updater and renderer for all UI elements. Including timers, text and images. Depending on the current state of the game it controls what elements of the UI are shown such as the tile screen, initial start intro sequences and the play state UI.

### 2.8 Utils Package

#### 2.8.1 ImageScaler.java

This class is responsible for taking any buffered image and properly scaling it to the specified resolution. It is used in this project to convert 16x16 or 48x48 sized images to a standard 32x32.

#### 2.8.2 HighscoreManager.java

This class is responsible for providing a methods to the engine that will read and write the highscore achieved to from and to a text file.

### 2.8 Sound Package

#### 2.8.1 Sounds.java

This class is responsible for holding a reference to the music files and sound effects needed to for the project.

#### 2.8.2 SoundManager.java

This class provides the methods to play music on loops and play a secondary sound instance for each sound effect i.e. (eating pellets, eating ghosts, Pacman dying, winning a level or losing).

## Frameworks and Software

### 3.1 Frameworks

#### 3.1.1 Java Development Kit (JDK)

Required for java development. Includes the Java Runtime Environment (JRE) aswell as tools that are used for debugging and monitoring java applications.

#### 3.1.2 Graphics Libraries

Uses Java2D and Swing components to create graphics for more advanced rendering.

#### 3.1.3 Sound Libraries

Utilizes the Java Sound API with .wav format sound files to play sound effects and music within the game.

### 3.2 Software

#### 3.2.1 Eclipse IDE

An integrated development environment (IDE) for Java development. Eclipse provides tools for writing, compiling, debugging, and running Java programs.

#### 3.2.2 JUnit

A unit testing framework for Java. Junit is used to write and run tests for your game code to ensure its correctness.

#### 3.2.3 Github

A version control system for tracking changes in your codebase. Git helps you manage and provide version control for the game project with ease.

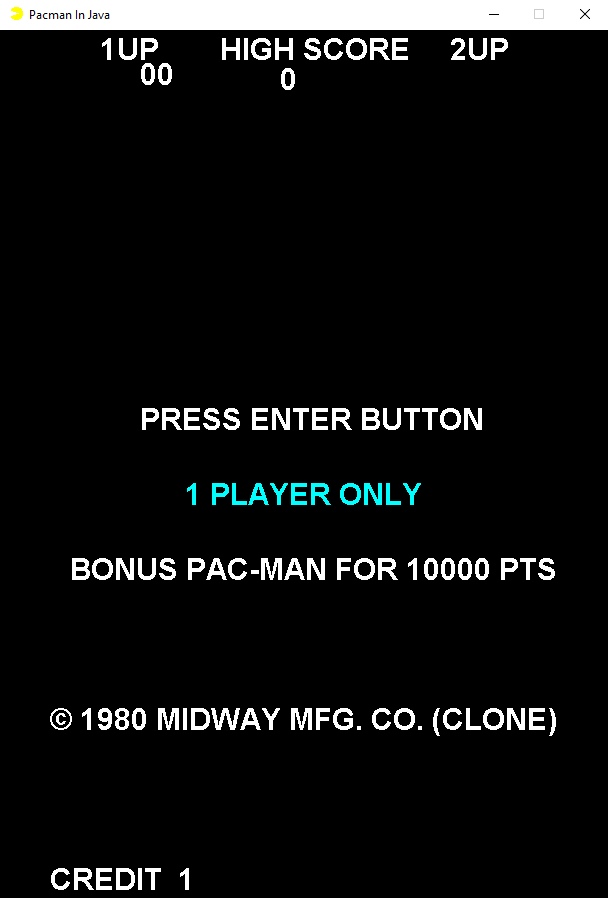
## Detailed User Interface

The user interface begins with the initial intro sequence followed by the prompt for the player to start the game. Then while in play state shows the score, high score, map, and lives data. Below are the current layouts of the UI.

A screenshot of a game

Description automatically generatedIntro Sequence (animated).

Title Screen Prompt



Play Screen

