MEGAMAN GAME

COURSE: OBJECT-ORIENTED PROGRAMMING

GROUP: Billie English

INTERNATIONAL UNIVERSITY – VIETNAM NATIONAL UNIVERSITY HCMC School of Computer Science and Engineering

Game Developer:

Hoàng Xuân Dũng-ITITWE22009

Nguyễn Quang Thái- ITCSIU22227

Nguyễn Nhật Nam- ITITWE22419

Nguyễn Trung Anh- ITITWE22122

**Table of content:**

1. Introduction:
2. Project objectives:
3. Tools and Frameworks Used
4. Package and Systems Components:
5. Control
6. GameEffect
7. GameObject
8. UserInterface
9. UML Diagram
10. Results, Limitations, and Future Development
11. Conclusion

1. **Introduction**

The MegaMan game project is a modularly designed application inspired by the classic MegaMan series. This report covers the development methodology, tools, UML design, and future improvements for a 2D platformer game created using object-oriented programming principles and the LibGDX framework.

1. **Project Objectives**

The MegaMan game project aims to showcase the practical application of Object-Oriented Programming (OOP) principles in game development. A central goal is to create a fully functional 2D platformer game where players control the protagonist, Megaman, navigating through various levels filled with challenges. The game mechanics allow the player to jump, attack, and interact with the environment, including defeating enemies, collecting items, and achieving level objectives. These features are designed to demonstrate the versatility and effectiveness of OOP techniques.

To achieve cross-platform compatibility, the LibGDX framework is employed. This framework enables the game to run seamlessly on multiple platforms, including desktop and web environments, by leveraging its core and desktop modules. The project's architecture is intentionally modular, with well-defined packages such as Control, GameEffect, GameObject, and UserInterface, ensuring scalability and maintainability. This structure supports the addition of new features, like additional levels, enemies, and characters, with minimal disruption to the existing codebase.

The project emphasizes the application of OOP concepts, including inheritance, polymorphism, and encapsulation, to foster code reusability and clarity. Advanced design patterns, such as Bridge, Command, and Singleton, are implemented to streamline the development process and enhance code organization. Furthermore, the game includes a user-friendly interface with menus, heads-up displays (HUD), and visually engaging transitions, supported by sound effects and animations to provide an immersive player experience.

1. **Tools and Frameưorks Used**

The MegaMan project leverages a robust set of tools and frameworks to ensure a smooth and scalable development process. The primary programming language used is **Java**, chosen for its versatility and extensive library support, making it ideal for game development. The **LibGDX framework** plays a crucial role by enabling cross-platform functionality, allowing the game to run seamlessly on both desktop and web environments. This framework facilitates the development of a high-performance 2D game engine, simplifying tasks like rendering, animation, and input handling.

For development, tools like **IntelliJ IDEA** and **NetBeans IDE** provide an integrated environment for writing, debugging, and testing code. These IDEs enhance productivity with features like syntax highlighting, version control integration, and efficient refactoring options. Version control is managed using **Git**, ensuring collaborative work and codebase tracking across team members, while also enabling the rollback of changes if needed.

The build process relies on **Apache Ant**, a flexible build automation tool that streamlines compiling, packaging, and deploying the project. This system ensures consistent builds across various platforms. Game assets, including sprites, sound effects, and level data, are organized within a structured **data folder**, making it easy to manage and update resources without modifying core code. By combining these tools and frameworks, the MegaMan project achieves both technical sophistication and maintainability.

1. **Package and Systems Components**:

This project is structured into several key packages: Control, GameEffect, GameObject, and UserInterface, along with an independent MegaMan class. These components work together to provide a modular and scalable game design.

The **Control** package handles the overall game flow, managing initialization, state transitions, and input handling. Key classes include Button.java, which defines interactive buttons, and RectangleButton.java, which builds on this functionality by adding position and size attributes to buttons. These classes ensure seamless user interaction and flexible interface management.

The **GameEffect** package enhances the visual and auditory experience of the game. It includes classes such as Animation.java for managing sprite animations and CacheDataLoader.java for efficient asset loading and caching. These components ensure that the game runs smoothly with immersive effects.

The **GameObject** package focuses on the interactive entities in the game. It contains classes like Megaman.java, which defines the main character's behavior and actions, FinalBoss.java for the final enemy logic, and BulletManager.java to handle projectile interactions. These classes encapsulate the core mechanics of gameplay, from character movement to enemy AI.

The **UserInterface** package manages the visual and interactive elements displayed to the player. Classes such as GameFrame.java, GamePanel.java, and InputManager.java ensure a cohesive and responsive interface, providing players with essential HUD elements and seamless menu navigation.

Finally, the **MegaMan** class acts as the entry point for the game. Extending the libGDX Game class, it initializes the virtual screen size, defines bit configurations for game objects, and loads all assets required for gameplay. This class is also responsible for setting the initial screen, ensuring a smooth start to the player’s experience.

Together, these packages and classes form the foundation of the MegaMan game, ensuring a structured, maintainable, and engaging gameplay experience.

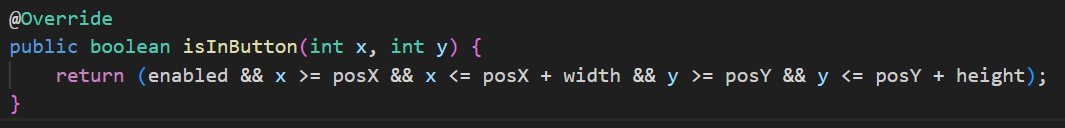
1. Control

The **Control** package is an essential part of the MegaMan game, managing core logic such as initialization, state transitions, and user interactions. It provides the foundational classes for building flexible and interactive game components.

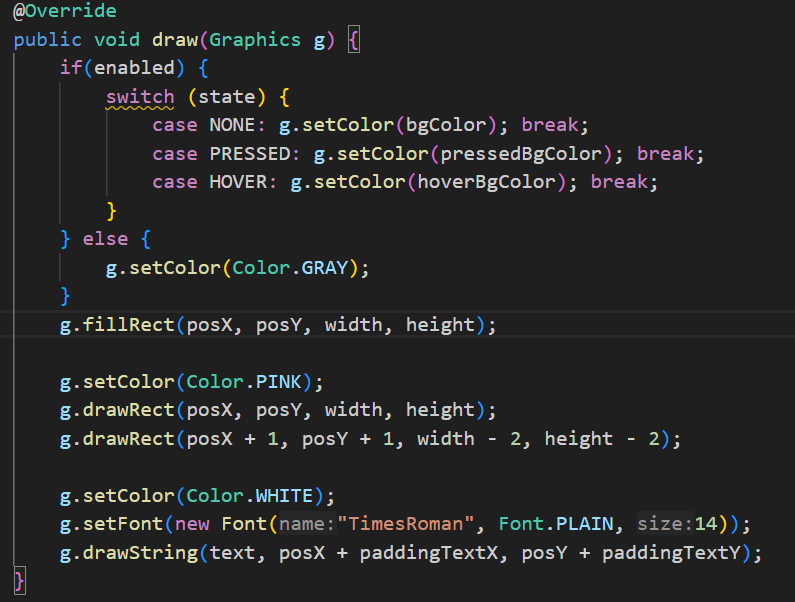
For example, Button.java is an abstract class that establishes the basic structure for interactive buttons. It includes core properties like position, dimensions, and states (e.g., NONE, PRESSED, HOVER) that allow for dynamic interactions. A key method is isInButton, which determines if a mouse click is within the button's bounds:



Building on this, RectangleButton.java extends Button to provide specific logic for rectangular buttons. It implements the isInButton method to detect hover and click events based on the button's position and size:



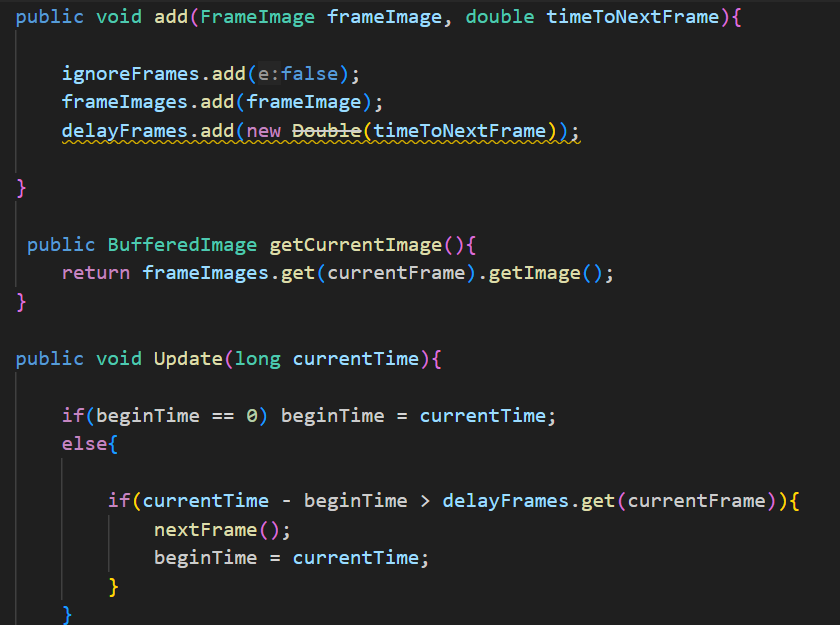
It also renders buttons dynamically with the draw method, changing their appearance based on states like NONE, PRESSED, or HOVER:



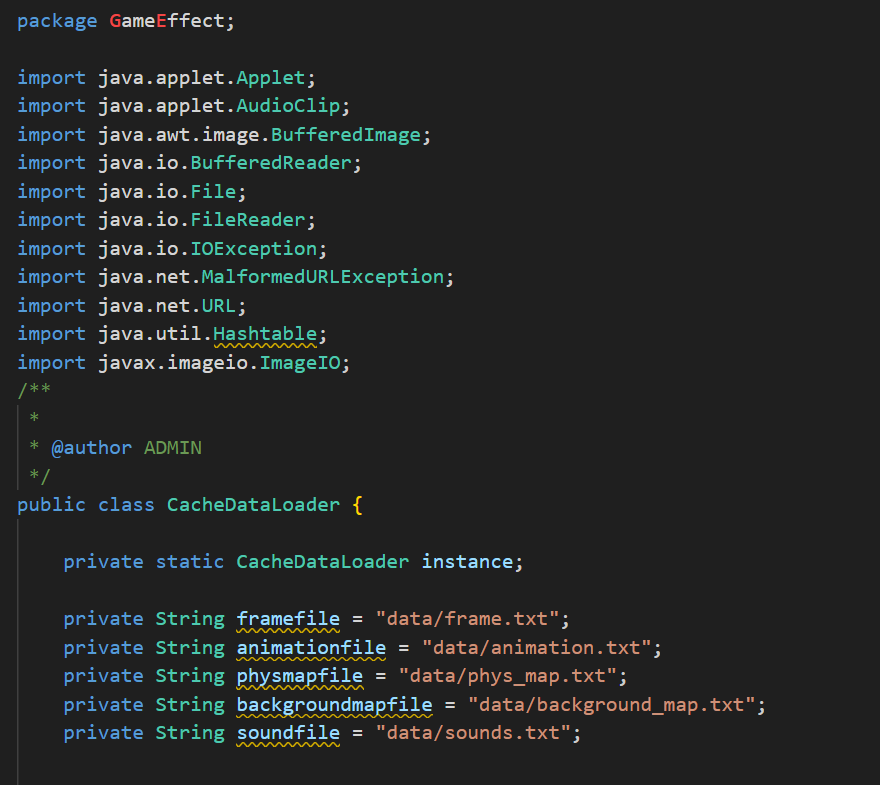
1. GameEffect

The **GameEffect** package is responsible for managing the visual and auditory aspects of the game, enhancing the player experience through animations and sound effects. This package ensures smooth sprite transitions and immersive audio feedback, making the game more dynamic and engaging

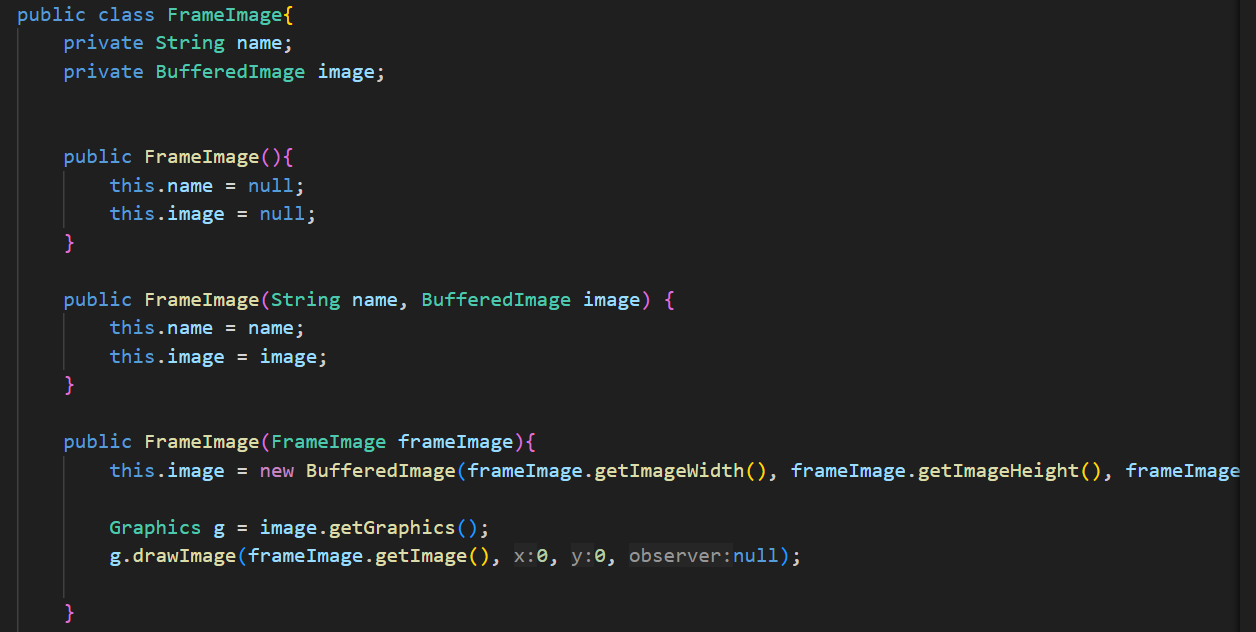
The **Animation** class provides frame-based animations for game objects. It allows objects such as the player and enemies to transition seamlessly between states like idle, running, and attacking, ensuring a polished visual representation during gameplay. Key methods include adding frames to an animation and updating the current frame:



The **CacheDataLoader** class handles the efficient loading and caching of game assets such as images, sprite sheets, and sound effects. This reduces loading times and ensures resources are reused efficiently. For example, the class uses a hashtable to store assets for quick access. It imports necessary libraries for managing files, images, and audio clips, as seen in the snippet below:



The **FrameImage** class supports animations by encapsulating individual frames of a sprite sheet. It provides methods for retrieving and managing frame images, ensuring consistent transitions between frames:

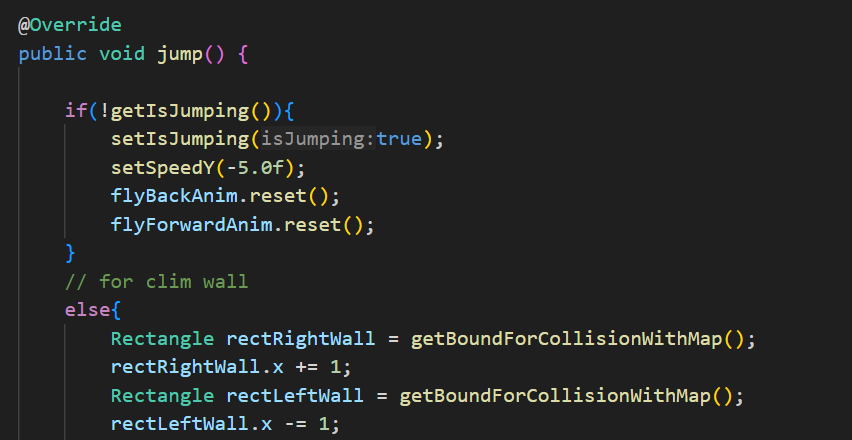


By integrating with other packages such as **GameObject** and **UserInterface**, the GameEffect package provides essential support for animations like running, jumping, and attacking, as well as sound effects triggered during collisions, attacks, and item pickups. This cohesive functionality ensures the game delivers a visually appealing and engaging experience for players.

1. GameObject

The GameObject package is the central backbone of the game, responsible for managing all the entities and their behaviors within the game world. It encapsulates key game mechanics such as movement, interactions, and environmental management. Below is a detailed overview of the primary classes in this package:

**Megaman.java**: This class represents the main protagonist, MegaMan. It is responsible for implementing the character’s core mechanics, such as movement, health management, and attack systems. For instance, the jump() method ensures that MegaMan can perform jumps while keeping his state consistent:



**FinalBoss.java**: The FinalBoss class inherits from the Human class. It represents the ultimate challenge in the game, featuring predefined attack and movement patterns rather than dynamic tracking. The boss interacts with MegaMan based on proximity or scripted behaviors, creating strategic gameplay challenges

**Human.java**: The Human class serves as a base class for human-like characters in the game, including both the player (Megaman) and certain enemies like FinalBoss. It encapsulates shared functionality such as movement, health, and animations

**DarkRaise.java:** The DarkRaise class represents a mid-level enemy with unique attack patterns. It tracks the player and shoots projectiles (defined in DarkRaiseBullet.java) to challenge the player.

**ParticularObject.java**: The ParticularObject class is the base class for all specific game objects, including both characters and interactive elements. It defines shared properties such as position, velocity, and state

**ParticularObjectManager.java**: The ParticularObjectManager class handles collections of ParticularObject instances. It ensures efficient updates, collision checks, and rendering of all game objects.

**FireBall.java**: The FireBall class defines the projectiles used by MegaMan. These projectiles damage enemies and disappear after a set duration. The update() method ensures fireballs are updated and destroyed appropriately

**RedEyeDevil.java**: The RedEyeDevil class represents another enemy type with distinct behaviors. It shoots RedEyeBullet projectiles at the player.

BackgroundMap.java: The BackgroundMap class is responsible for rendering the static environment of the game, ensuring the background remains consistent and visually appealing.

PhysicalMap.java: The PhysicalMap class defines the physical boundaries and collision logic for the game world. Using Box2D, it creates static objects like walls and platforms

GameWorld.java: The GameWorld class serves as the central hub for managing all game objects. It updates and renders entities every frame, ensuring smooth gameplay and interactions.

State.java: The State class manages different game states such as "RUNNING," "PAUSED," and "GAME\_OVER." This allows the game to handle transitions and control flow effectively.

Bullet.java: Represents a generic projectile in the game. It includes movement logic and interactions with game objects.

BulletManager.java: Manages all active bullets in the game, ensuring efficient updates and rendering.

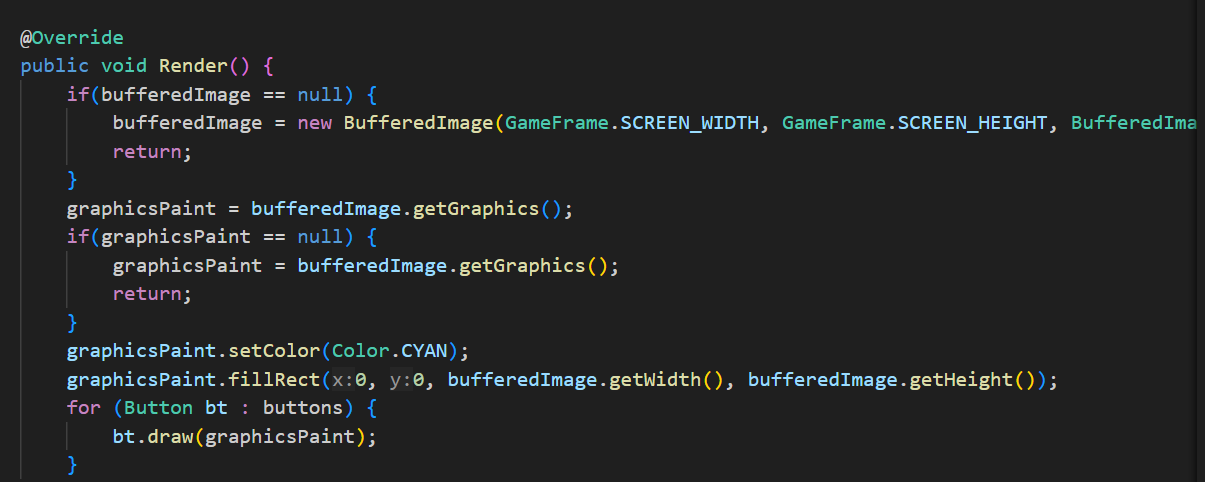
Camera.java: Controls the in-game camera, ensuring it follows the player or focuses on specific objects when necessary.

RobotR.java: Represents a robotic enemy with specific attack patterns and behaviors.

SmallRedGun.java: A small enemy type that fires projectiles at the player, providing a basic challenge.

YellowFlowerBullet.java: Defines specialized projectiles fired by certain enemies.

**MenuState.java**: The MenuState class manages the game’s menu functionality, including navigation and user interaction. It allows players to start the game, view instructions, or exit. For example, its render() method handles menu rendering



1. UserInterface

The UserInterface package is responsible for managing the game’s graphical interface and input handling. Below are the key components:

**GameFrame.java**: This class serves as the central hub for initializing and managing the game window. It provides methods to configure the frame’s properties and integrates the GamePanel.

**InputManager.java**: The InputManager class handles all user input from the keyboard and maps these inputs to game actions. It ensures the game responds dynamically to player commands.

1. **UML**

UML Analysis: The UML diagram provides a comprehensive view of the overall structure and relationships between classes in the MegaMan game project. It highlights the implementation of core Object-Oriented Programming (OOP) principles, such as inheritance, encapsulation, and polymorphism. The following are detailed observations:

Class Hierarchy: At the core of the class structure are foundational classes like GameObject, ParticularObject, and Human, which serve as base classes for more specialized components. For instance, Megaman inherits from Human, while FinalBoss and Enemy extend ParticularObject. This hierarchy ensures shared properties and methods, such as movement and health, are reused and extended, adhering to the principle of inheritance.

Relationships:

Inheritance: Derived classes such as Megaman and FinalBoss leverage the functionality of their parent classes, allowing for code reuse and specialization. For example, FinalBoss overrides attack behaviors defined in Human to create unique combat mechanics.

Composition: Classes like GamePanel and GameWorld integrate multiple objects, such as InputManager and Animation, to perform specific tasks. This modular design enables flexibility and scalability in managing game states and rendering.

Subsystems: The system is divided into distinct subsystems to streamline functionality and maintain separation of concerns:

User Interface (UI): This subsystem includes classes like GameFrame, GamePanel, and InputManager. Together, they manage user inputs and graphical rendering, ensuring seamless interaction between the player and the game.

Game Entities: Classes such as Megaman, Enemy, and Bullet represent in-game objects. These classes encapsulate properties and behaviors specific to their roles, such as movement, attacking, and interaction with other objects.

Core Systems: Central components like GameWorld, PhysicalMap, and CacheDataLoader handle the game's state, physics, and resource management. For instance, GameWorld updates the state of all entities and ensures smooth gameplay.

Key Features:

Encapsulation: Shared behaviors and properties, such as position and state, are encapsulated in base classes like ParticularObject. This ensures consistency and reduces redundancy across the codebase.

Polymorphism: Derived classes override methods from their parent classes to implement specific behaviors. For example, FinalBoss redefines attack patterns, making it distinct from other Enemy objects while maintaining compatibility with the ParticularObject interface.

Overall, the UML diagram underscores a well-structured and modular design, enabling easy maintenance and scalability of the MegaMan game project.