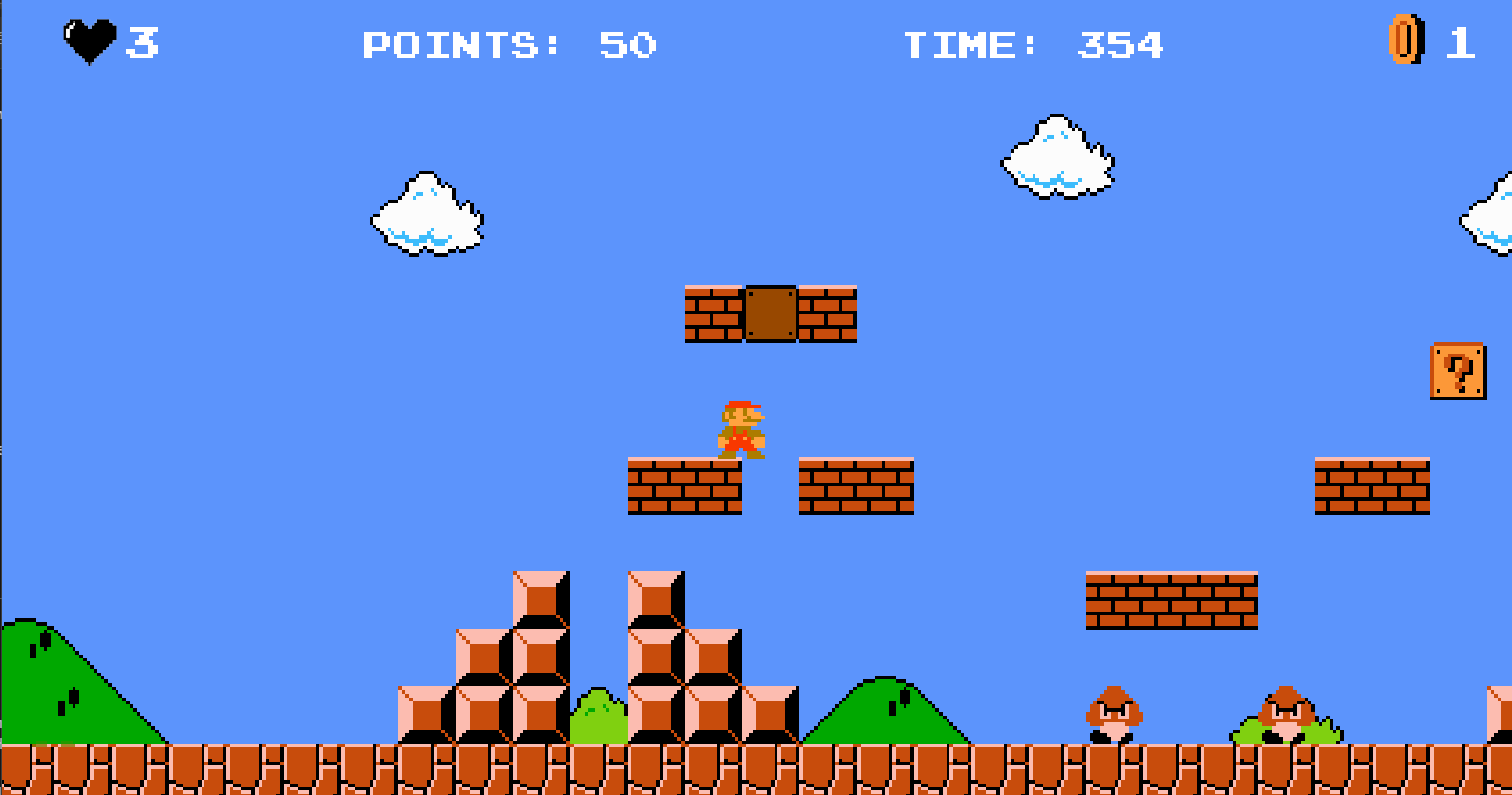
SUPER MARIO



**TABLE OF CONTENT**

[**CHAP 1: INTRODUCTION** 2](#_Toc185244081)

[1.1 Introduction 2](#_Toc185244082)

[1.2 About the game project 2](#_Toc185244083)

[1.3 Developer team 2](#_Toc185244084)

[**CHAP 2: ABOUT SUPER MARIO BROSJECT** 3](#_Toc185244085)

[REQUIREMENTS 3](#_Toc185244086)

[2.1 Easy to Access & Play 3](#_Toc185244087)

[2.2 What we have 3](#_Toc185244088)

[2.3 What we want 3](#_Toc185244089)

[2.4 Working tools, platform 4](#_Toc185244090)

[2.4.1 Programming language: Java with addition libraries 4](#_Toc185244091)

[2.4.2 Git and GitHub: For managing project and storing on repository 4](#_Toc185244092)

[2.4.3 IDE: IntelliJ IDE, Netbeans,VSC 4](#_Toc185244093)

[2.5 Use case scenario 4](#_Toc185244094)

[2.6 Use case diagram 4](#_Toc185244095)

[**CHAPTER 3: UML DESIGN AND SYSTEM MODELS** 5](#_Toc185244096)

[3.1. UML Design 5](#_Toc185244097)

[3.1.1. Manager package 5](#_Toc185244098)

[3.1.2. Model package 6](#_Toc185244099)

[3.1.3. view package 10](#_Toc185244100)

[3.2 System models 11](#_Toc185244101)

[3.2.1. Use Case Model 11](#_Toc185244102)

[3.2.2 Dynamic models 11](#_Toc185244103)

[3.2.3 Game System 14](#_Toc185244104)

[**CHAPTER 4: FINAL APP GAME EXPERIENCE** 30](#_Toc185244105)

[Instruction: 30](#_Toc185244106)

[**REFERENCES** 33](#_Toc185244107)

# **CHAP 1: INTRODUCTION**

## 1.1 Introduction

Super Mario, first released in 1985, is a game developed by Nintendo.

This project will aim to create a game that is very similar to the original Super Mario game. The game can be categorized as a level-based adventure game, with Mario being the main character.

The main objective of the game would be to overcome obstacles and enemies with the help of items and reach the end point.

This game will use java programming language to make.

## 1.2 About the game project

The ultimate goal of the game is to clear as many levels as possible with a limited amount of lives (can obtain through items). a level itself consists of enemies, obstacles and traps, but there will also be bonuses like coins and power-up items to help players. The game counts the number of points for each game and the number of coins players have collected, the number of monsters killed and number of lives remaining.

Player can control Mario by just the keyboard like arrow keys and spacebar

## 1.3 Developer team

|  |  |  |
| --- | --- | --- |
| **Name - GitHub username** | **Student ID** | **Contribute** |
| 1. Phan Vũ Tuấn Khanh -   <https://github.com/PhanKhanh-datasc> | ITDSIU23008 | model package, game object, handler, collision detection |
| 1. Huỳnh Tấn Phước -  <https://github.com/phuoc759860> | ITDSIU23018 | model package (hero, brick, Enemy, prize) |
| 1. Nguyễn Phước Nguyên - <https://github.com/Nguyen-Analyst> | ITDSIU23016 | manager package |
| 1. Phạm Huỳnh Nhật Tân - <https://github.com/Random-come> | ITDSIU23022 | View, package, window, animation, texture |

# **CHAP 2: ABOUT SUPER MARIO BROSJECT**

## **REQUIREMENTS**

## **2.1 Easy to Access & Play**

* Working on Window platform
* Developed by using Java cross-platform programming language
* Using .jar file for easy to run and share

## **2.2 What we have**

* Basic knowledge of Java
* Tools with affordable cost
* Coding & algorithm are analysed clearly

## **2.3 What we want**

* Easy to update
* Teamwork with efficient manner
* Complete the project in the possible fastest time while still ensuring the quality

## **2.4 Working tools, platform**

### 2.4.1 Programming language: Java with addition libraries

* java.awt: Used for handling graphical components such as rendering game objects, colours, and layout positioning.
* java.swing: Manages user interface elements like the game window, buttons, and menus.
* java.io: handle files such as images, resources

### 2.4.2 Git and GitHub: For managing project and storing on repository

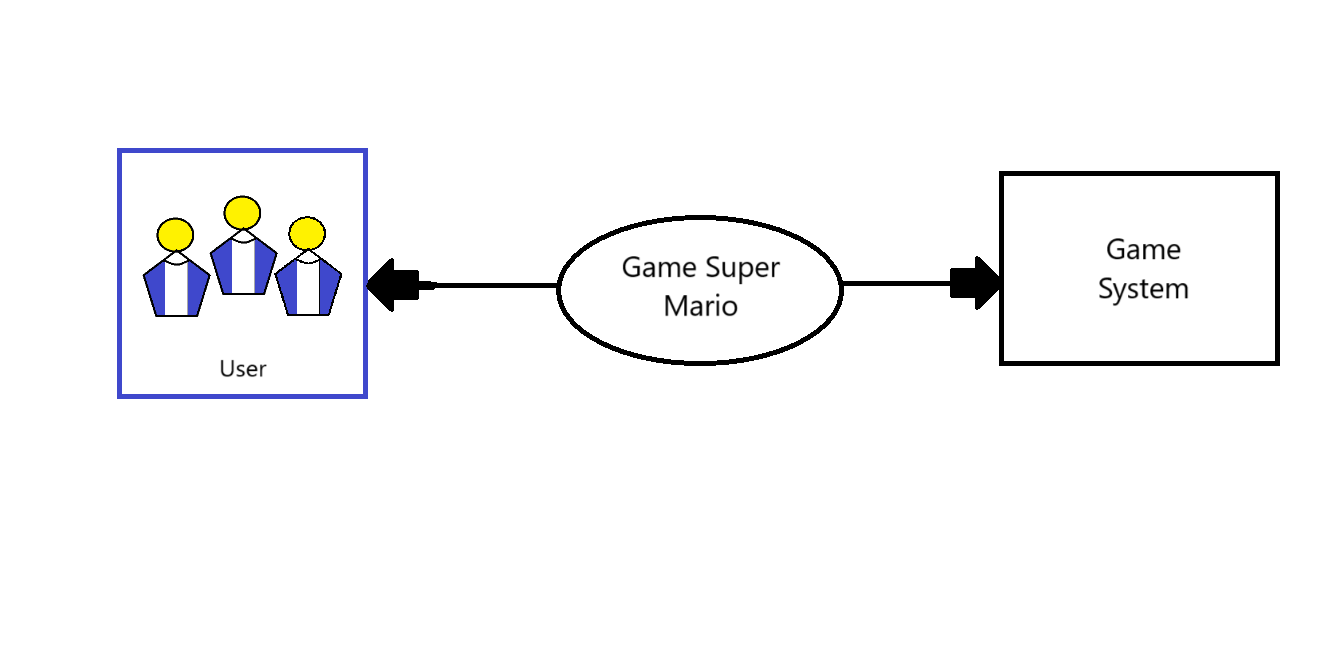
### 2.4.3 IDE: IntelliJ IDE, Netbeans,VSC

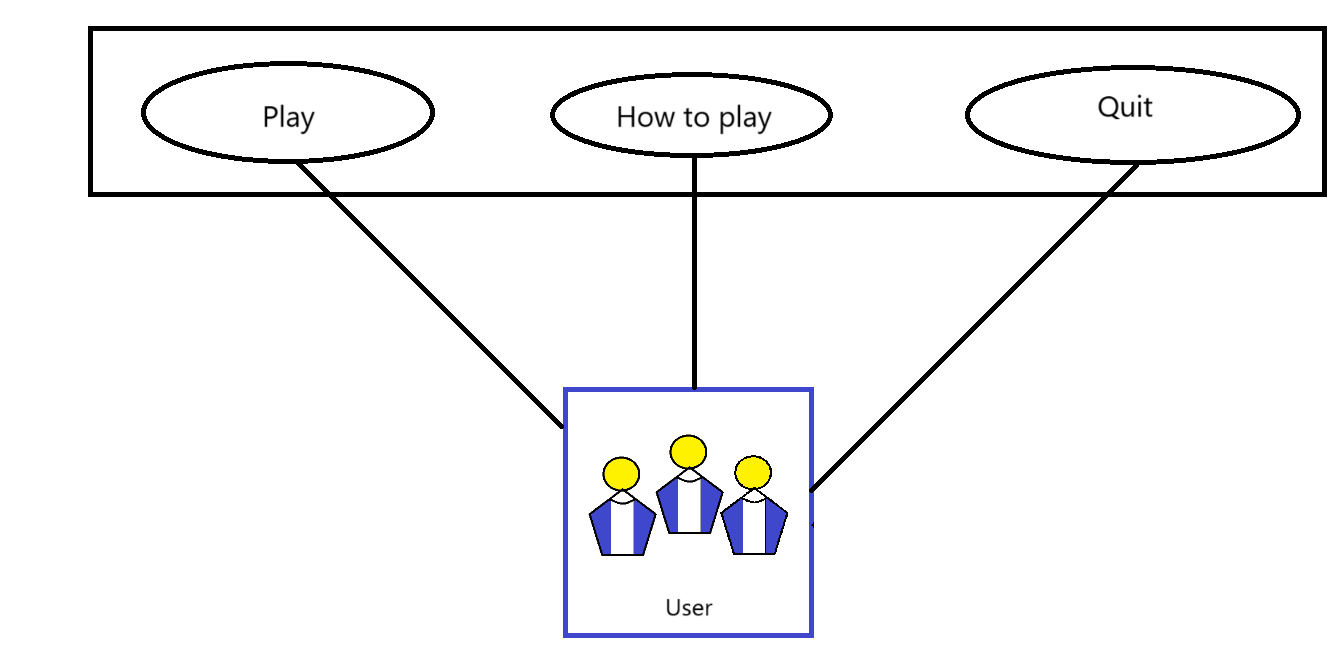
## **2.5 Use case scenario**

Based on the user UX view we created this use case

|  |  |  |
| --- | --- | --- |
| **SUPER MARIO BROS** | PLAY | START GAME |
| RESUME THE GAME |
| EXIT THE GAME |
| HOW TO PLAY | Click “Help” on the start screen |
| QUIT | Exit the game |

## **2.6 Use case diagram**

****

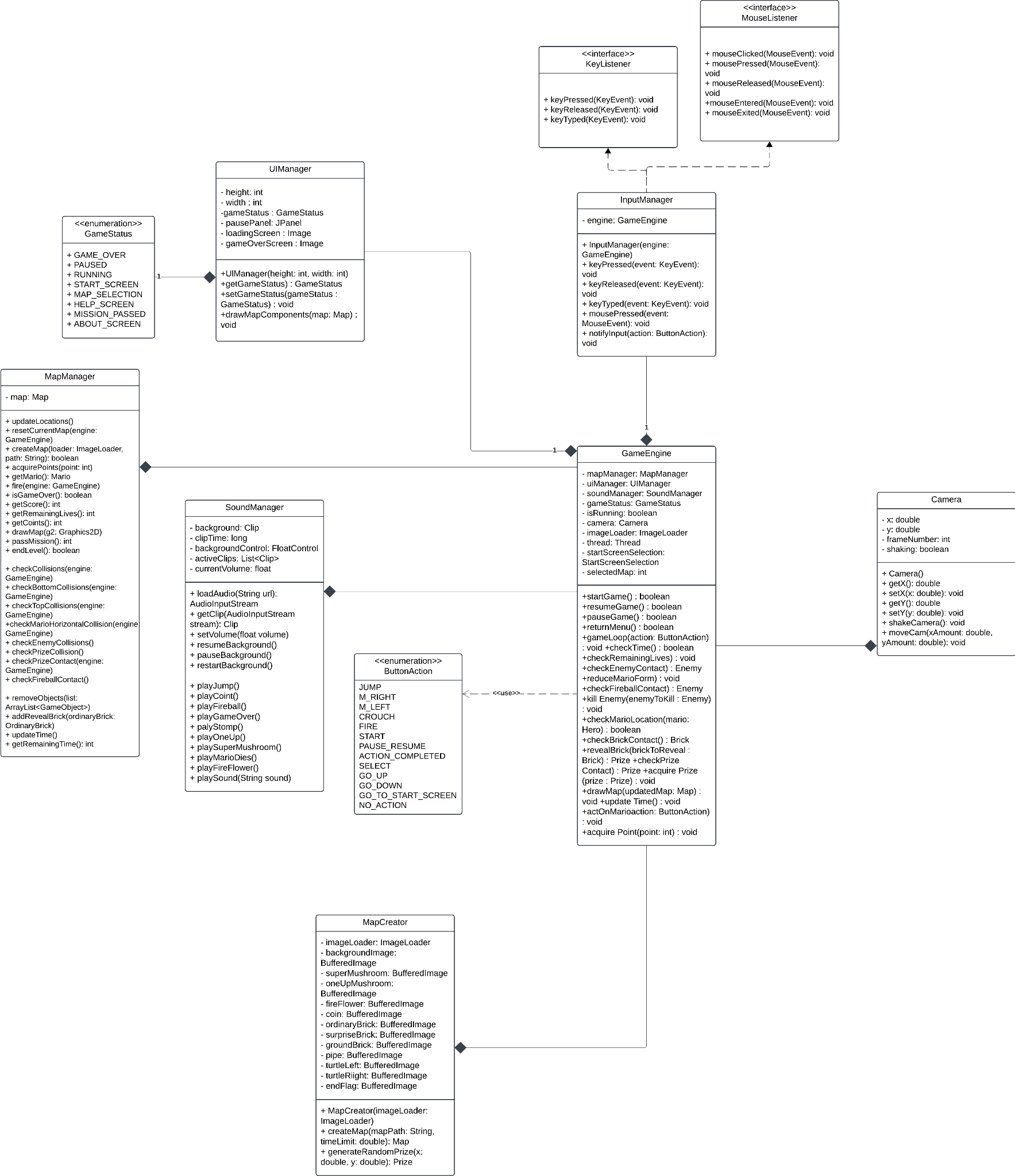


# **CHAPTER 3: UML DESIGN AND SYSTEM MODELS**

## 3.1. UML Design

### 3.1.1. Manager package

The manager package organizes and oversees key game functionalities, including managing camera movements, handling button actions, and coordinating gameplay mechanics. It ensures smooth transitions, player input handling, and dynamic game state adjustments.

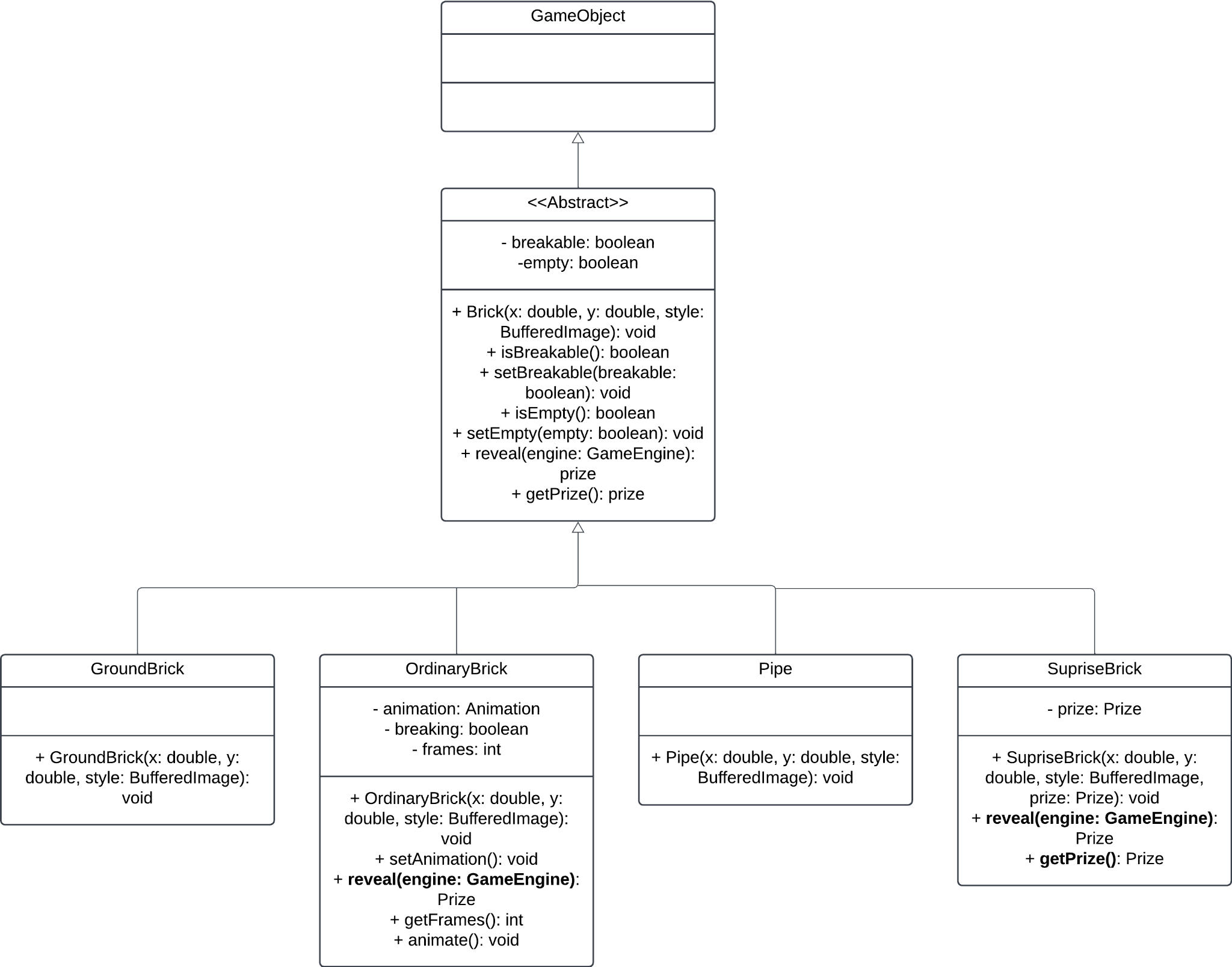


### 3.1.2. Model package

We have 4 main packages (brick, enemy, hero, prize), each of them contain multiple classes for specific object

#### 3.1.2.1 model.brick

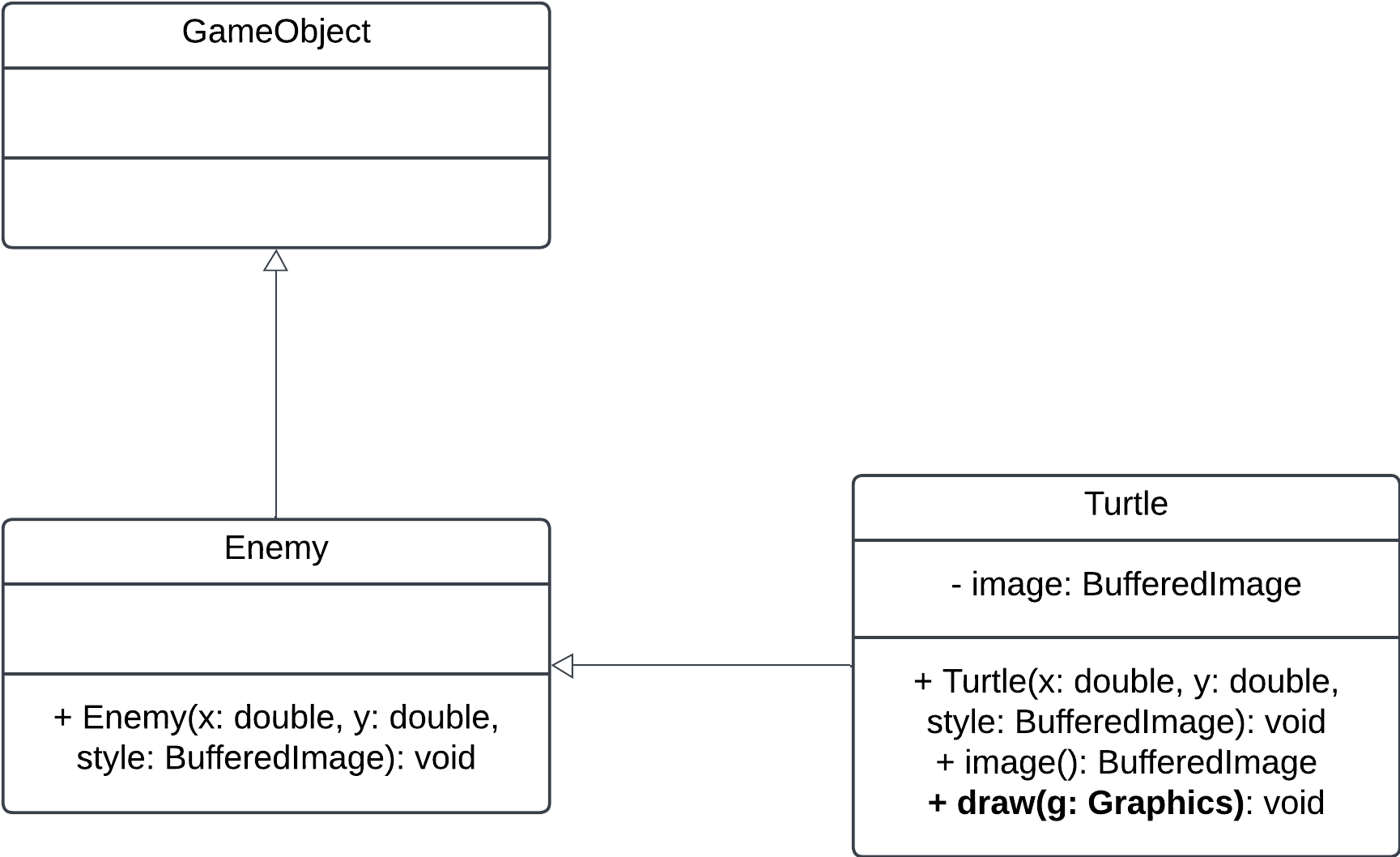
In our Mario game, the map is created from many different types of visible objects including ground brick, ordinary brick pipe and brick containing prizes and brick packages responsible for defining those objects.



#### 3.1.2.2 model.Enemy

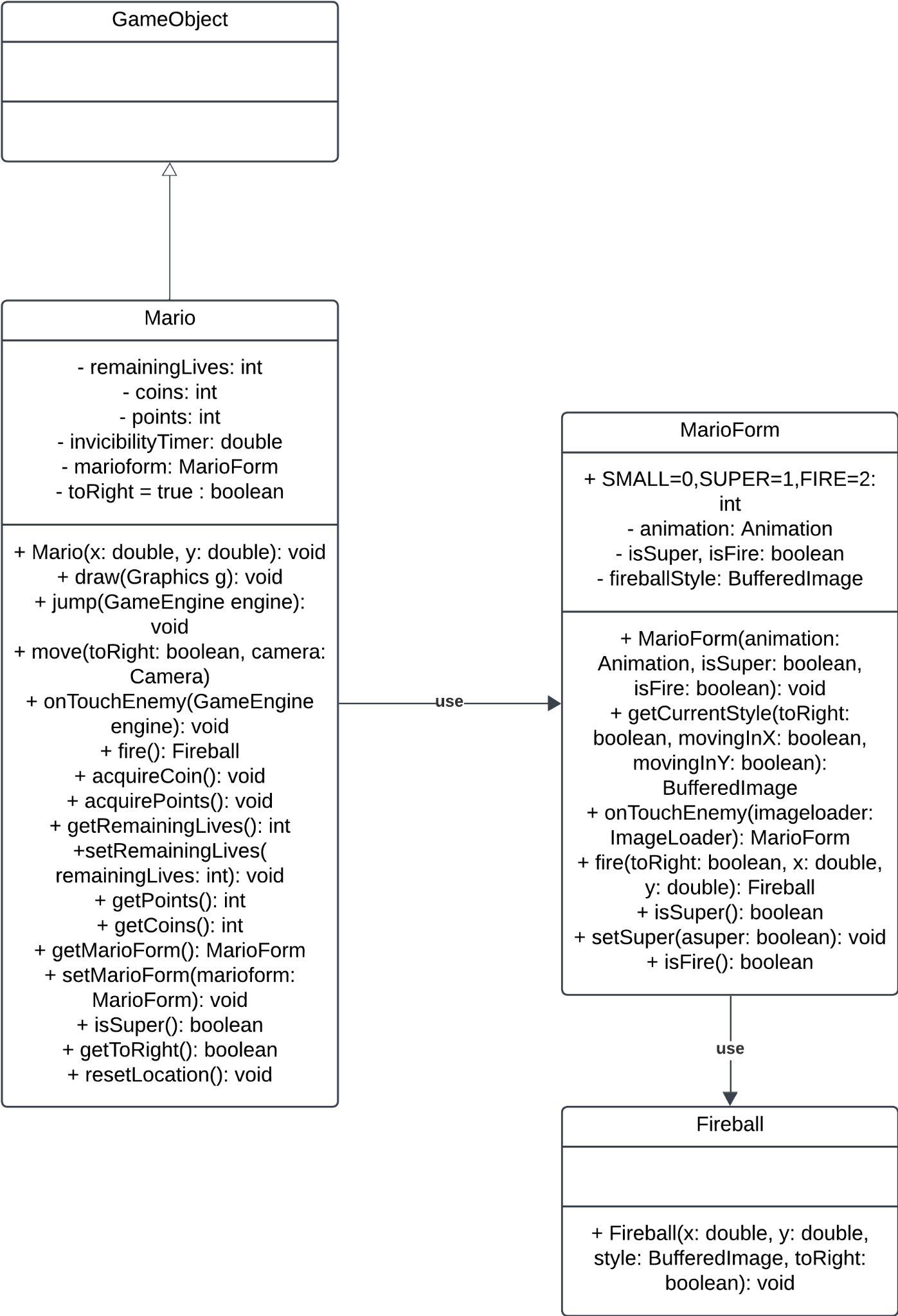
Enemy package, which Responsible to declare the enemy attributes

Enemy class defined:



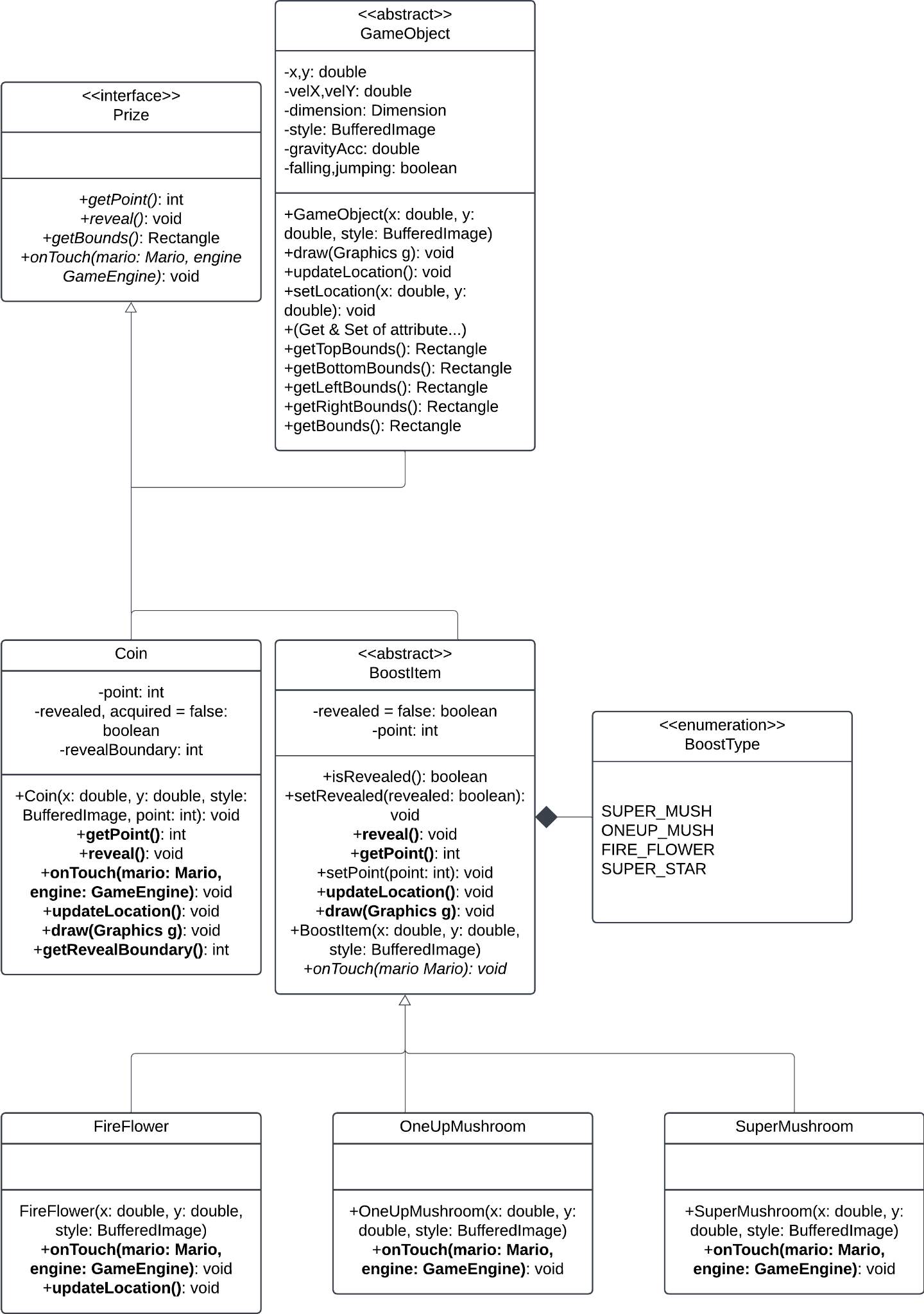
#### 3.1.2.3 model.hero

In the model.hero package, we have Mario, MarioForm, Fireball classes. Fireball classes will have the constructor method for creating fireballs that Fire Mario can shoot. MarioForm has methods for constructing and defining the consequence for touching the enemy in each form.



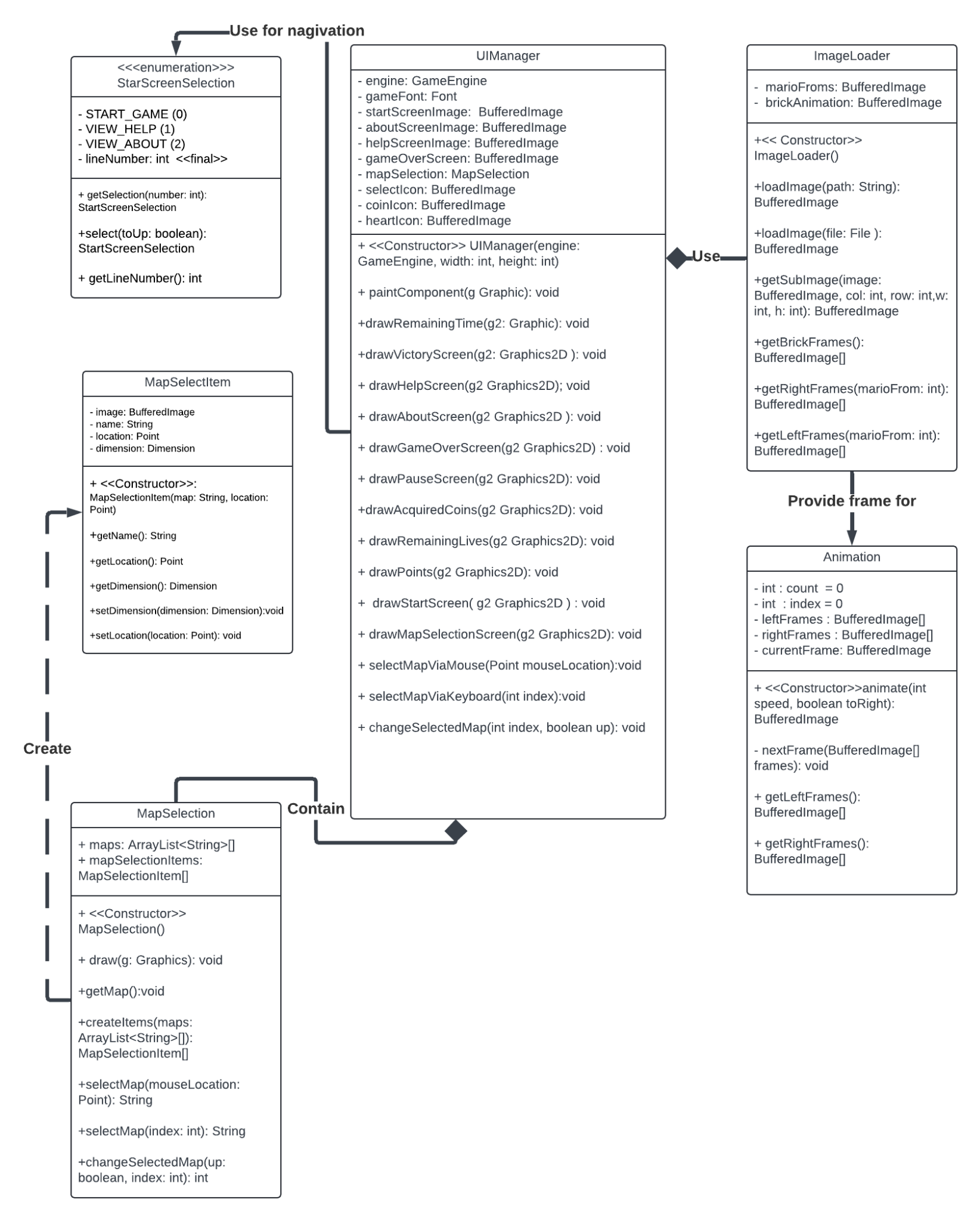
#### 3.1.2.4 model.prize

In the model.prize package we have Coin, SuperMushroom, OneUpMushroom, etc. Each of the class will have attributes and methods that will represent the constructor, states of object, and collision of objects



### 3.1.3. view package

"The view package is responsible for managing the entire user interface and visual components of the Mario game. It includes the following classes"



## 3.2 System models

### 3.2.1. Use Case Model

A **Use Case Model** captures the interactions between the user (player) and the system (Mario game).

* **Player**: Controls Mario using inputs (keyboard or controller).

Use Cases:

1. **Start Game**: Begin a new game or resume a saved game.
2. **Control Mario**: Move, jump, or crouch Mario.
3. **Collect Items**: Collect coins, power-ups, and keys.
4. **Defeat Enemies**: Stomp or use power-ups to defeat enemies.
5. **Reach Goal**: Complete the level by reaching the flagpole or other endpoint.
6. **Pause Game**: Temporarily halt the gameplay.
7. **Game Over**: Triggered when all lives are lost.

Use case Description

### 3.2.2 Dynamic models

**Scenario Name**: Start Game and Load Map

**Scenario**:

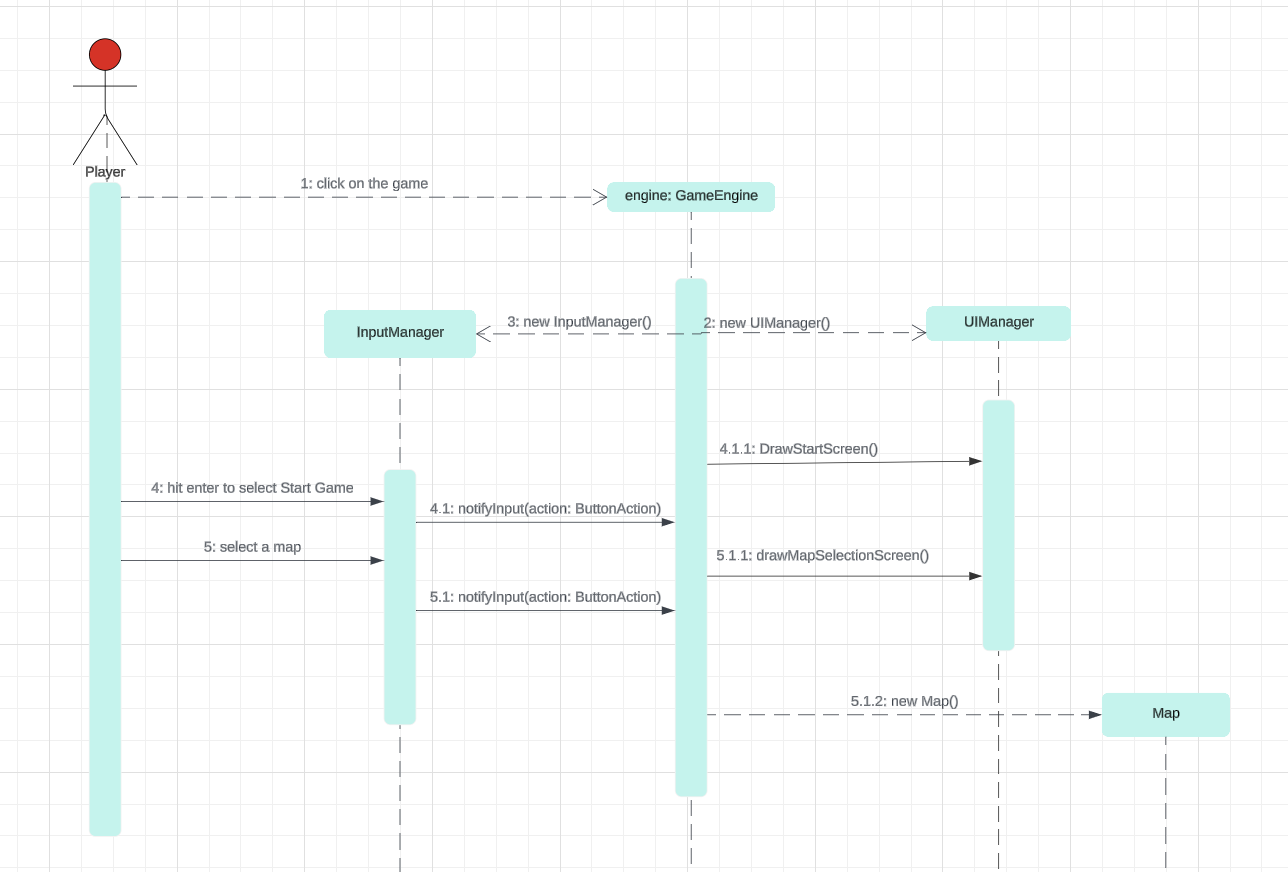
The player clicks one game icon and sees the main menu in the game. He/she chooses *Start Game* from the options by pressing the enter key (the default selection is Start Game). After that he/she chooses a map from possible options (he/she cannot choose a map without passing every map before it). Finally, the map is loaded, and he/she plays the game.

Figure 1. Sequence diagram for initial start of game

**Description:** He/she wants to play the game, so he starts the game. The GameEngine class is the main class of the game, so it is created. GameEngine creates a UIManager which shows an option panel to the player. Player chooses the Start Game option so *GameEngine* calls *drawMapSelectionScreen*() method of *UIManager*. The player selects a possible map and *GameEngine* creates a map accordingly.

**Scenario Name:** Revealing Super Mushroom

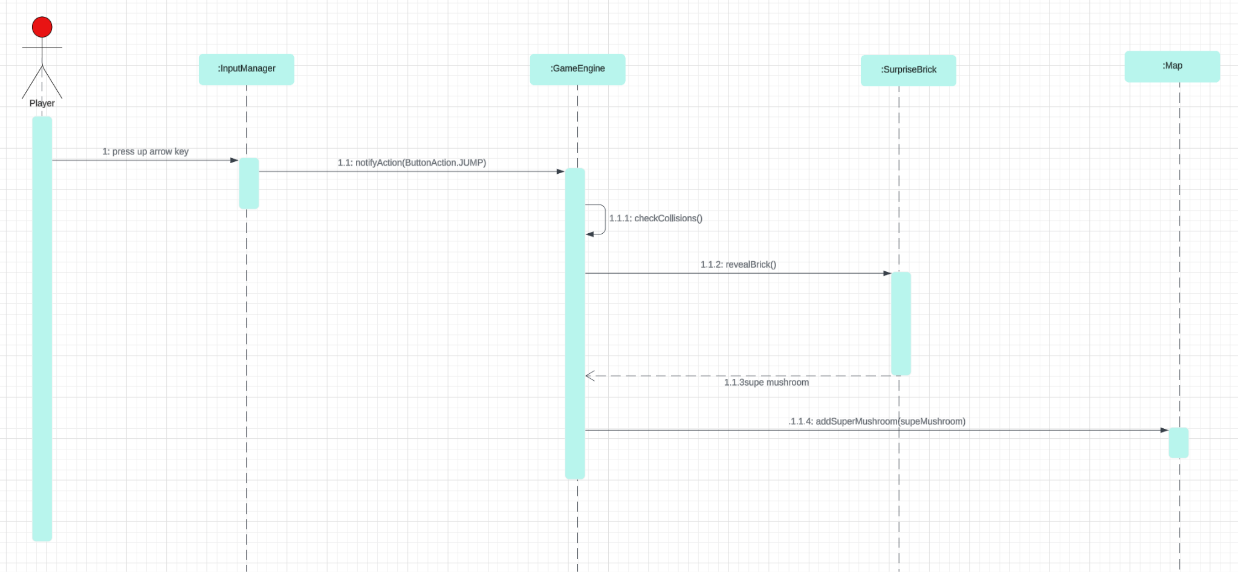
**Scenario:** The player has been playing Super Mario Bros with great joy for a while. He is trying to get coins and mushrooms by hitting *SurpriseBrick*s from the bottom. Turns out that the brick contains a SuperMushroom which is revealed. That SuperMushroom instance is added to Map.

Figure 2. Sequence diagram of revealing a super mushroom

**Description:** The player jumps under a *SurpriseBrick* which contains a *SuperMushroom*. First *GameEngine* is notified of the jump action. The engine updates the locations of all of the game objects and checks for collisions. At this moment *Mario* and the *SurpriseBrick* intersect so the *Brick* will be revealed and the *Prize* it contains will be returned. This *SuperMushroom*instance is put into a Map instance in *GameEngine* and so, it will move and can be eaten.

**Scenario Name:** Eating Super Mushroom

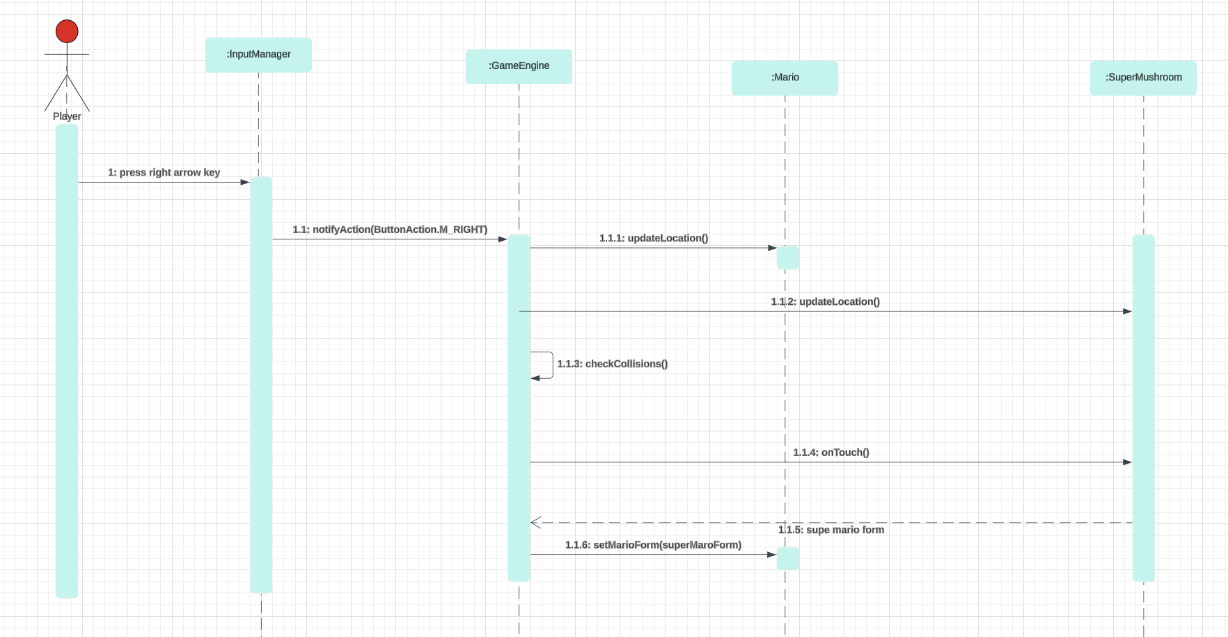
**Scenario:** After the great success of revealing the mystery of *SuperMushroom*, the player seeks more and wants to eat it. So, he moves to the right where the *SuperMushroom* lies and touches it. Then, *Mario* gains great power via this magical mushroom and turns into a super form. 

Figure 3. Sequence diagram of eating a super mushroom

**Description:** The player pressed the right arrow button to move *Mario* to the right. So, *InputManager* calls *notifyInput*() method of *GameEngine*. Then, *GameEngine* updates locations in *Map* class, in this case, *Mario* and *SuperMushroom*. Engine calculates collisions and finds that *Mario* touches SuperMushroom so invokes *onTouch*() method of the *SuperMushroom* which returns super mario form, an instance of the *MarioForm* class. Finally, *GameEngine* calls the setMarioForm() method of *Mario* with super mario form as an argument.

#### Object and Class model

*GameEngine* is the main class which controls the game according to the state of the actors in the map and inputs received from the player. *Map* class contains everything related to map and draws to the screen. It is only a visual container; it does not control anything. Hero class represents the possible characters the player can control during the game. *BoostItem* changes the state of the Hero, it makes the Hero stronger. *Prize*, *Brick*, *Hero* classes are model classes whereas *GameEngine* is controller and *Map* is viewing class.

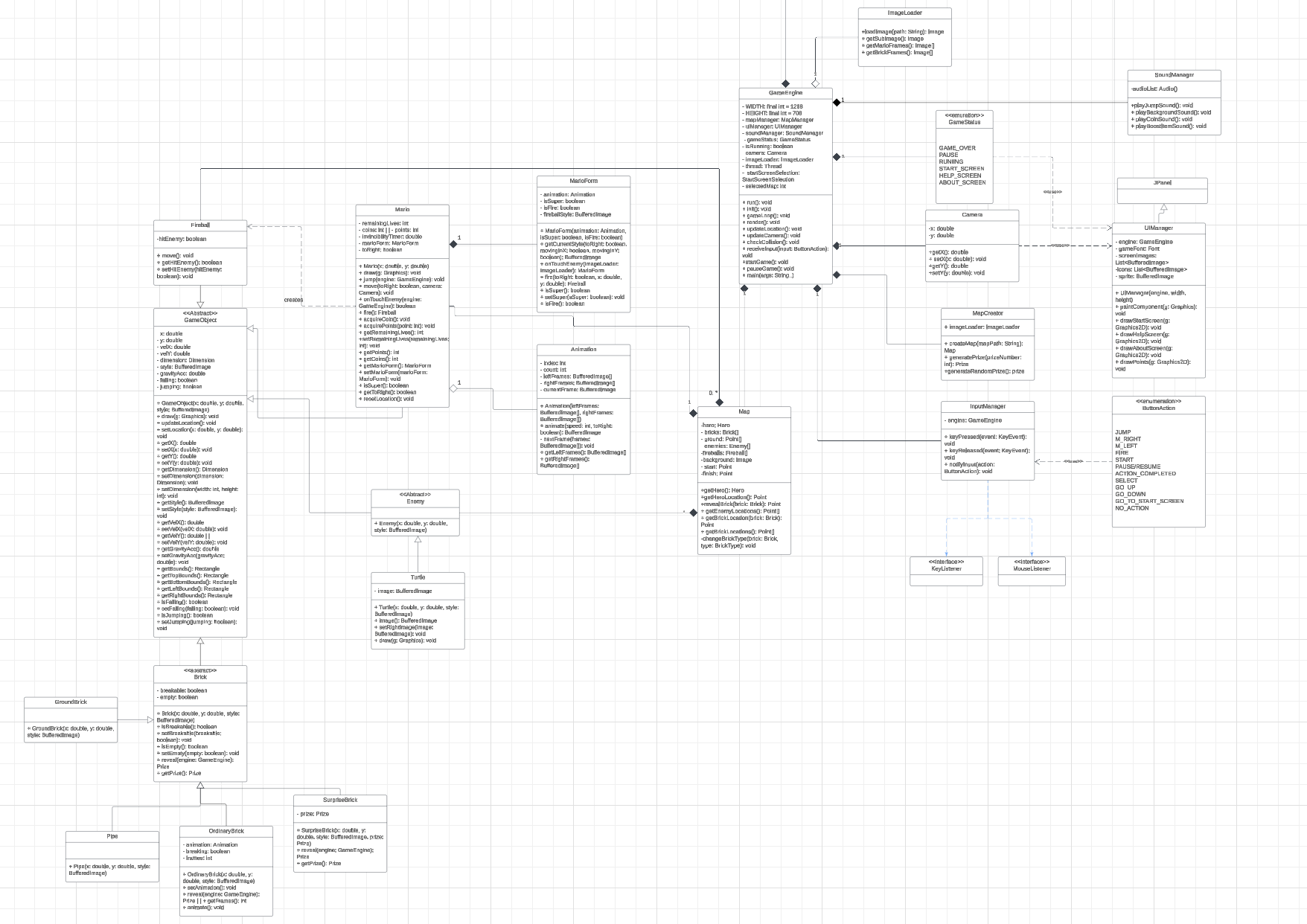


Figure. UML Class diagram of the game

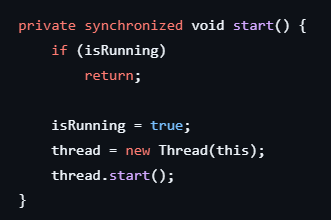
### 3.2.3 Game System

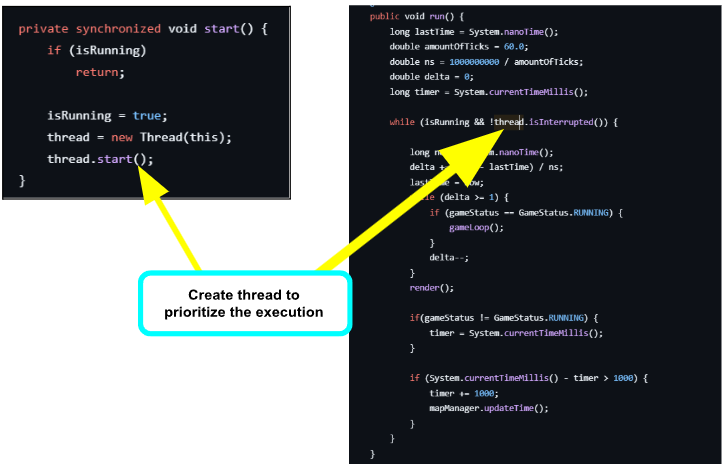
#### i.Game Loop

The gameLoop method in GameEngine is created to update location objects including Mario and enemies with updateLocations() method, continuously checking whether the collision happens between objects or not with checkCollisions(), updating the camera to keep Mario at the centre screen with updateCamera(). The game loop will end if game status consider as GAME\_OVER or MISSION\_PASSED



Game()-begin gameloop and pauseGame()-stop gameloop





* run() method responsible for representing the time has passed since the game start while it is not being interrupted

#### ii.Window:

**Window App management:**

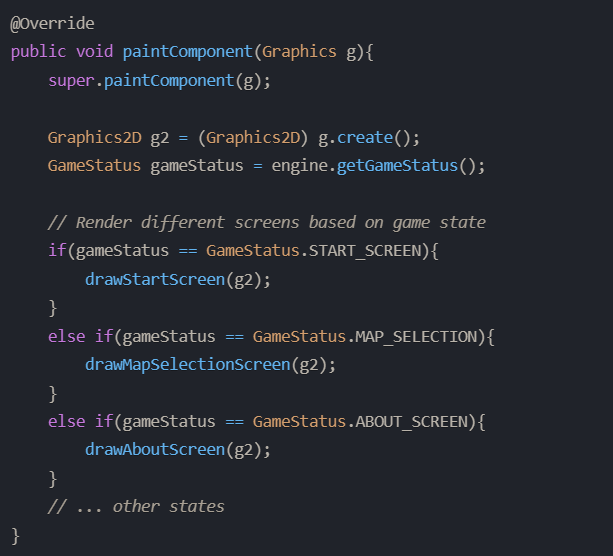
init() method is used for creating the new game’s window containing camera, user interface, map manager and sound manager.



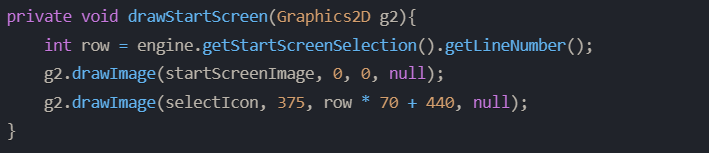
Graphical output is the function of render() method which import from JPanel and java.lang.Component

**Interface management.**

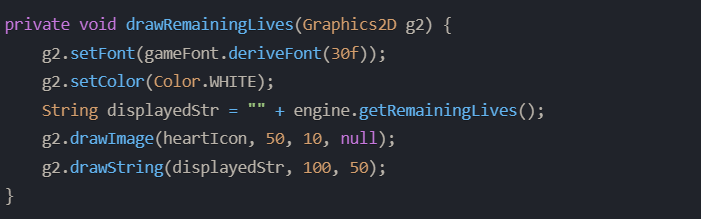
View package has a partial role in interface management. Rendering the Window and managing the frames are assigned to paintComponent() method in UIManager.java of the view package, it uses GameStatuss enumerate class to determine current screen,drawing method of each state is different and also have a way for switching between different game screen and display some information (lives remain of Mario , point and time)



*States in Super Mario Game*

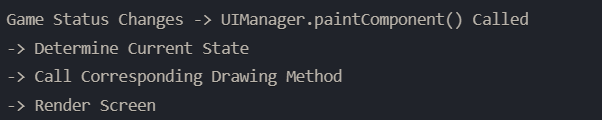


*Draw method that take responsibility on drawing starts selection*



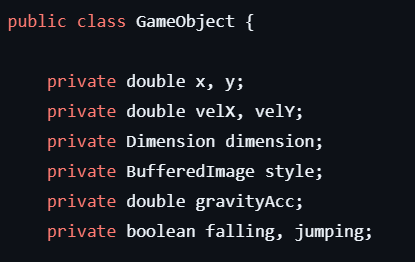
*Method for rendering Mario’s live*

We recognize separate game logic with rendering help us modifying, extending easier and managing game states clearly. Moreover, rendering only the necessary components makes minimal performance overhead more efficient.



*Workflow example for Window*

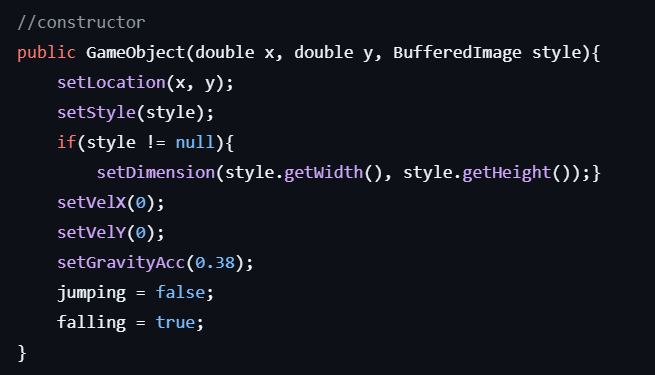
#### iii. Game Objects:

****

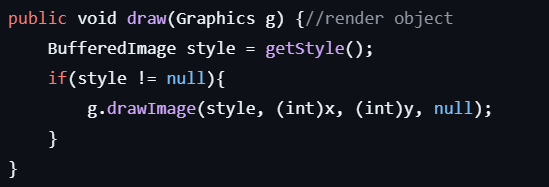
We have some crucial variables for objects:

* x, y: position of object
* velX, velY: velocity of object in x or y direction (how fast)
* dimension: the height and width of objects
* style: This variable holds the image representation of the object
* gravityAcc: accelerate of the gravity

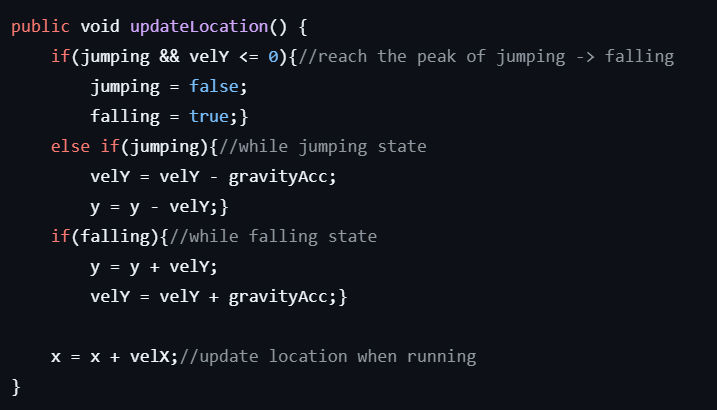
and then, the constructor:



the object when initialized at default as falling and not jumping -> useful for EndFlag object



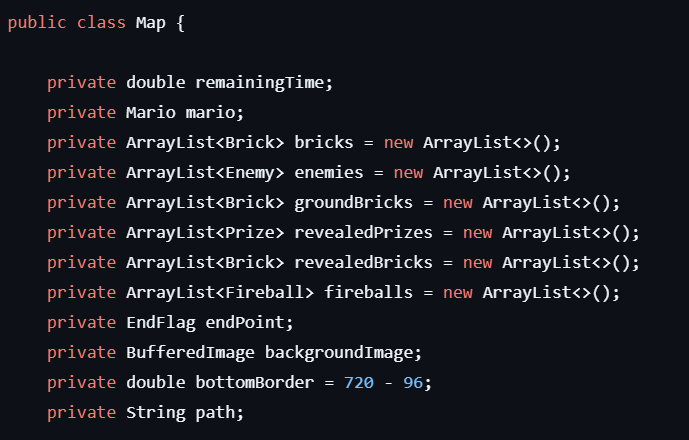
the draw method will hold all the graphic updates(anything displayed on the screen or animation…)



any positional update or state update will be done in updateLocation method

#### iv. Handler

map class where all of our objects will be hold



Variables:

* remainingTime: Represents the remaining time for completing the map

*\*if the player cannot complete level in time -> die*

* mario: Represents the main player character, Mario, in the game
* bricks: Represent a collection of bricks in the game map
* enemies: Represent a collection of enemies present on the map
* groundBricks: List of ground bricks present in the map
* revealedPrizes: A collection of prizes that have been revealed during the game.

*\* These prizes are no longer hidden and are available for interaction*

* revealedBricks: A collection of bricks that have been revealed during the game.

*\* Revealed bricks are those that Mario has interacted with, typically through jumps or collisions, causing them to*

*\* transform or display their hidden contents (such as prizes).*

* fireballs: Represents a collection of active fireballs within the game world.
* endPoint: Represents the ending point of the map in the game.
* backgroundImage: The background image of the map during gameplay.
* bottomBorder: Represents the bottom boundary of the game map.

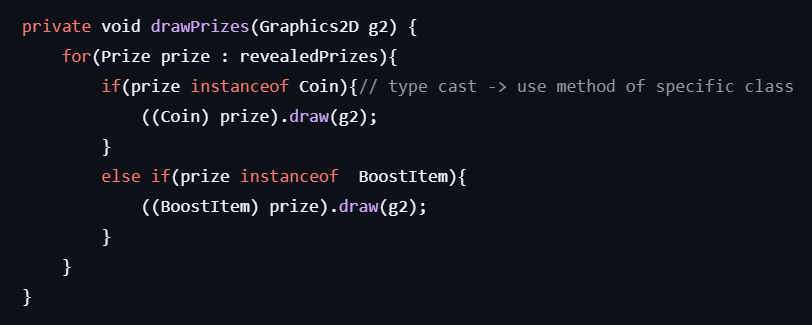
*\* This value is calculated as the difference between the total height of the window (720)*

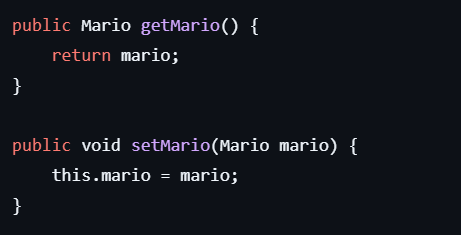
*\* and a predefined vertical offset (96) to account for UI components or other elements.*

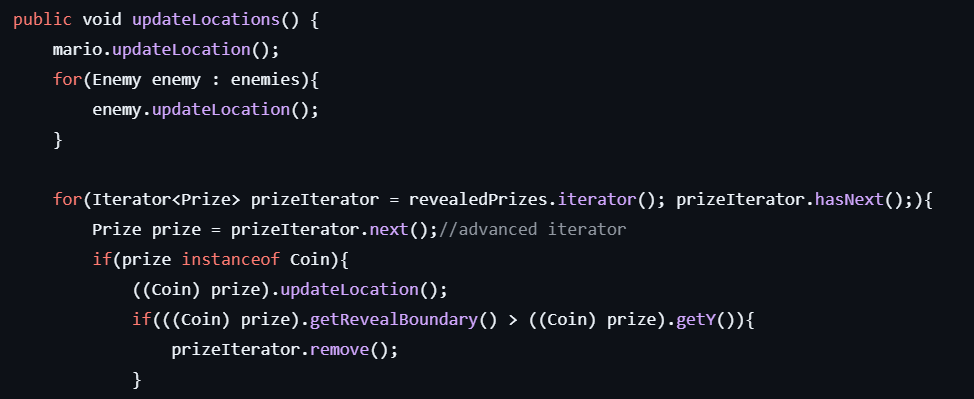
*\* Elements such as Mario, enemies, or other objects should not fall below this boundary.*

* path: Represents the file path to the resources or assets associated with the map.

In the map class, there are multiple methods for rendering specific objects, add objects into an arrayList, getter, setter, and positional control method.







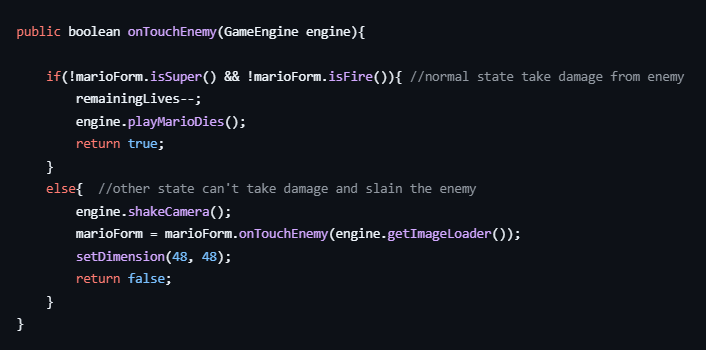
updateLocation() method of this class will control the type of the objects specifically(type casting) -> allows invoking methods specific to that object’s class

#### v. Player

Player: Mario is considered as a game object so Mario class will extend GameObject class (vertical & horizontal velocity-position) with other attributes (remainingLives, coins, points, invincibleTimes, marioForm [small, super, fire], toRight).In addition, the method to get the player-mario bound is necessary to create a rectangle to check if object collide with other objects

* move() method:

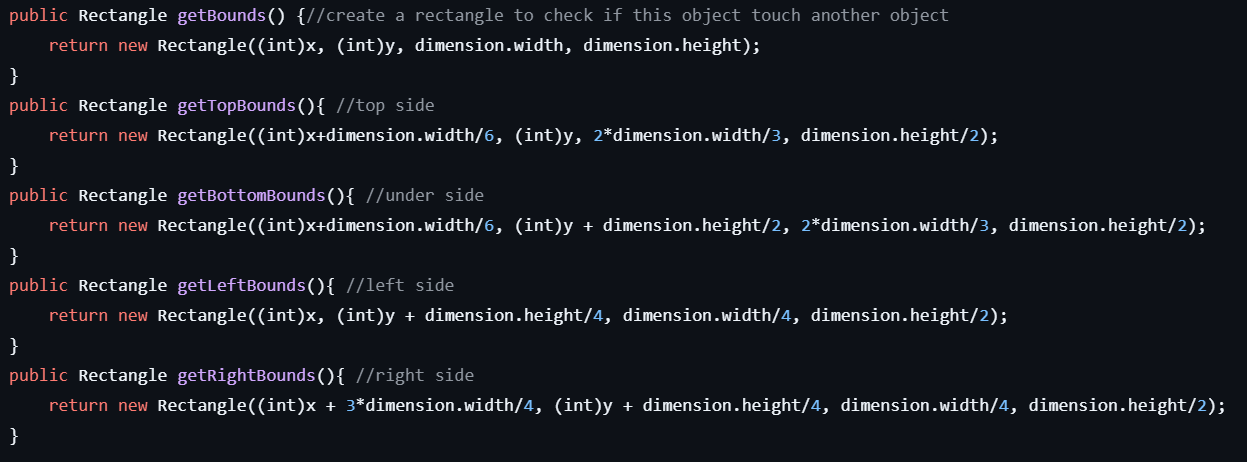


* onTouchEnemy method (result after mario touch enemy):

#### vi. Collision detection

The detection of collisions between different objects, when objects collide each other -> not just go through.

Methods to create bounds for object:



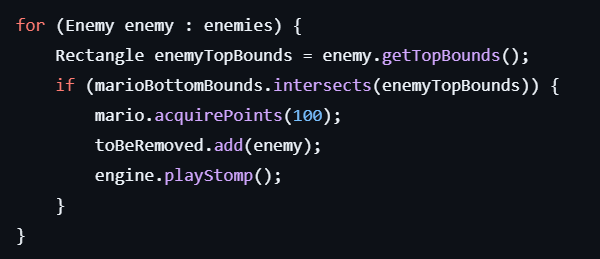
visualized bound:



\* the bounds width and height will be based on the width and height of the objects

\*grey: getBounds() -> surrounding bound to check if this object touch other object

after that, we will have methods for checking collision that have codes such as this:

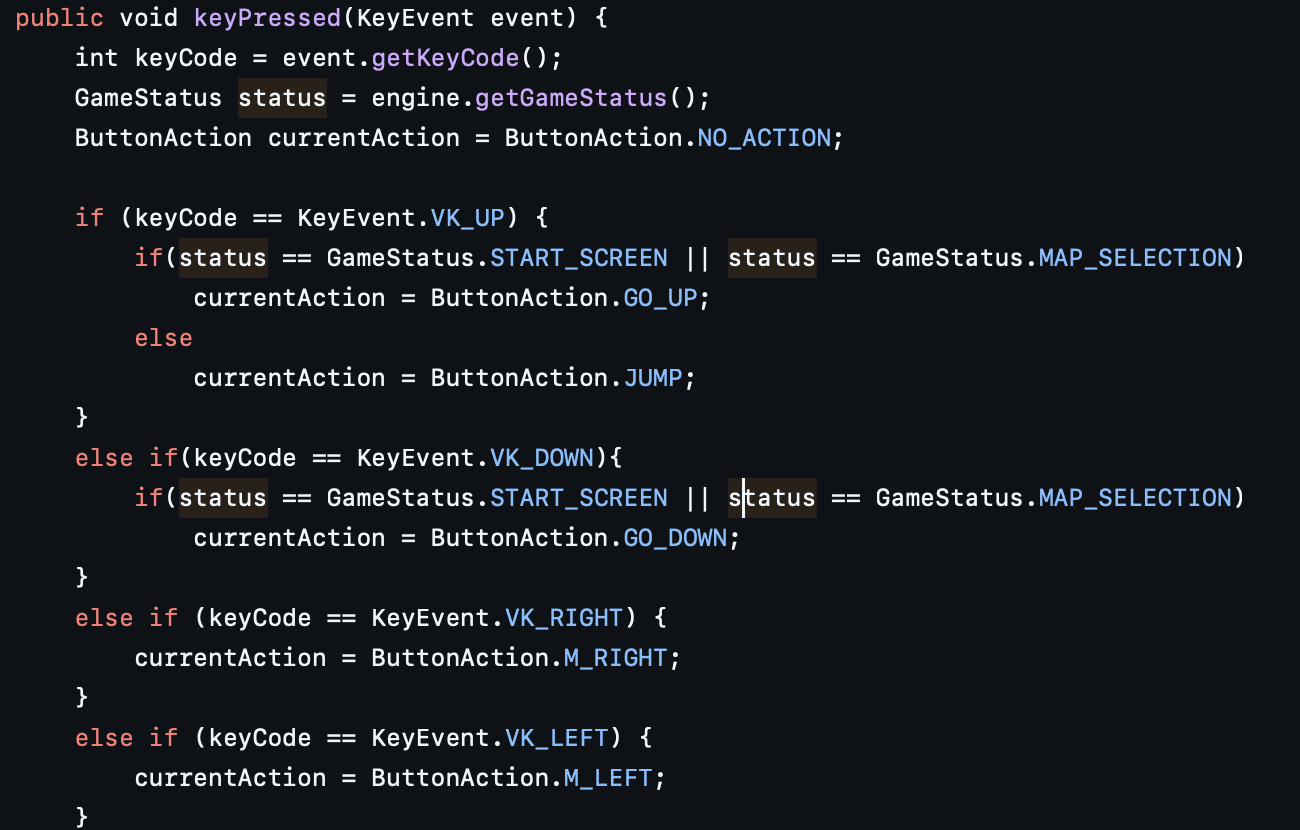


Example of how it works: when Mario bottom bound intersects with enemies’ top bounds, this action will execute consequences like: gain 100 points, remove enemy, play stomp sound.

\*The collision detection method will then be applied in the game loop.

#### vii. Keyboard Input

**Diagonal Movement:**

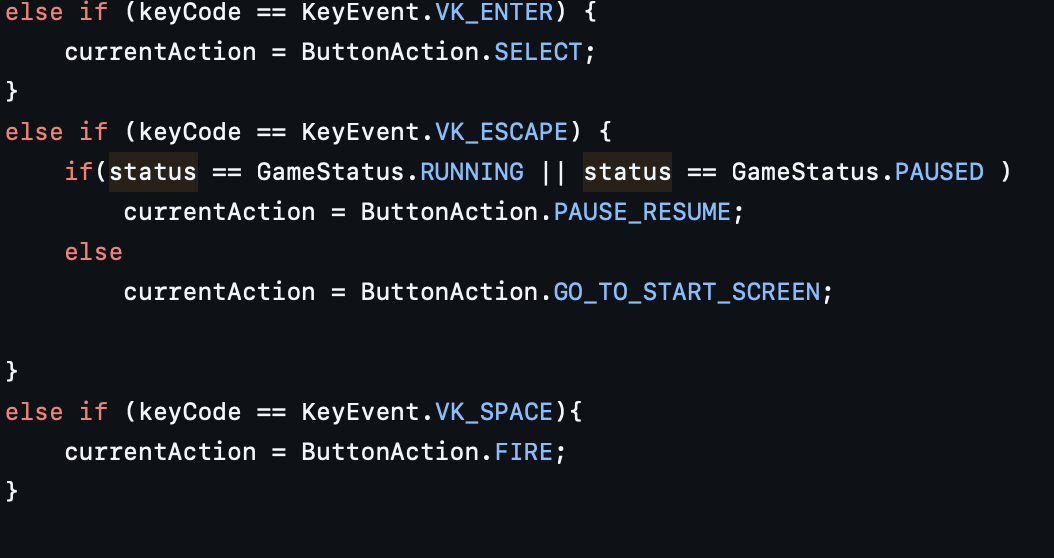
Allow pressing of directions (e.g. UP, RIGHT, DOWN, and LEFT) to enable diagonal movement which could be achieved by combiningkeyPressed states of multiple keys.****

**Action controls Game State**

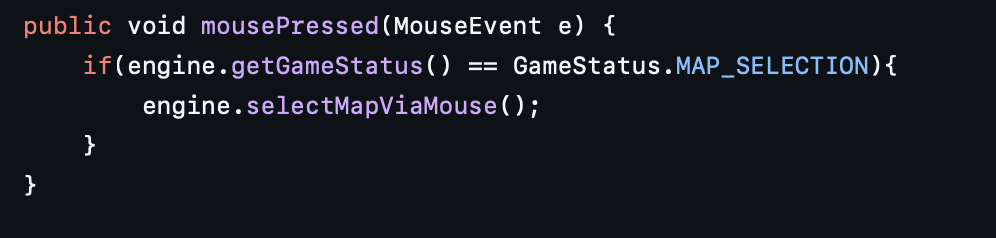
* SPACE key triggers the FIRE action.
* ENTER key is used to SELECT options, such as starting the game or confirming menu choices.

- ESCAPE:

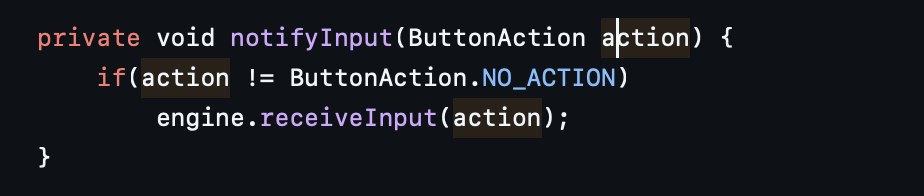
* In RUNNING or PAUSED states, toggles between pause and resume.
* In other states (e.g., START\_SCREEN), returns to the main menu.

-UP and DOWN keys: Used for navigation in the START\_SCREEN and MAP\_SELECTION states.  
****

**7.3. Mouse Input for Map Selection  
-**Mouse interaction allows players to select maps in the MAP\_SELECTION state by clicking.

****

**7.4. Integration with Game Engine**

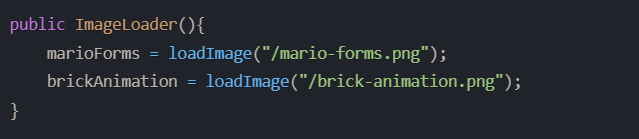
* Key and mouse inputs are tightly integrated with the GameEngine:
  + The notifyInput method ensures that only meaningful actions (ButtonAction) are passed to the game engine for processing.

#### viii. Textures

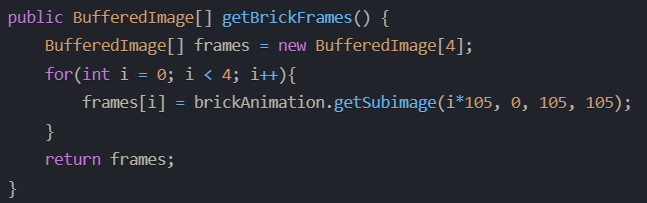
Textures that mainly holds this task is ImageLoader.java which can be considered as “texture manager” of the game, it takes responsibility for managing and providing textures for game objects.

**8.1 Image Loading.**

First responsibility of this class is to load an image from a resource directory which contains different image types (sprite sheets,icon,etc.) and provide error handling for image loading. Second, prepare animation frames for different Mario states based on character forms (basic, super and fire Mario). And the last is to extract specific sub-images into precise pixel-level images from larger sprite sheets , which help complete other missions.



*Loads main sprite sheets and prepares for later uses*



*Extract multiple frames for different game element (e.g. brick breaking animation)*

In addition to similar effects as in the Window section, these activities help the Mario game minimize memory usage and reduce code duplication.

#### ix. Level Design

****

Three maps here are the combination of many different colour bits and each RGB colour define for one object class 

After RGB’s object definition, we use the if-else function to assign the image (sprite’s png file) into the variables and ImageLoader to create the whole map.

**Check every single bit if it matches the RGB panel**

**RGB of**

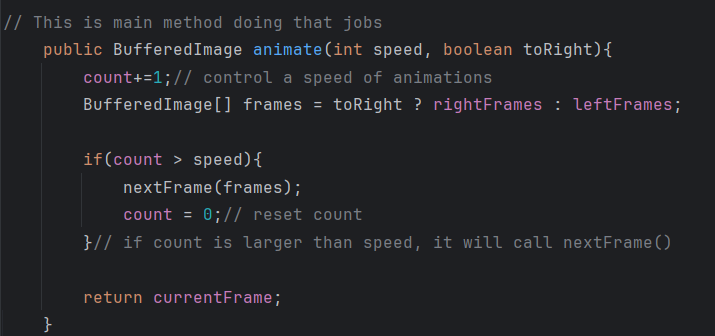
**movable Object**

#### x. Animation

Movementanimation logic of games is mainly handled by Animation class in the view package. The view package manages animation comprehensively, handling everything from frame extraction by ImageLoader to movement control, ensuring a dynamic and responsive character animation system by UIManager.

A screen shot of a computer

Description automatically generated



*Movement control*



count =0

speed = 0

count > speed

count = 0

speed = 0

*Simplified workflow of how animation work*

frame

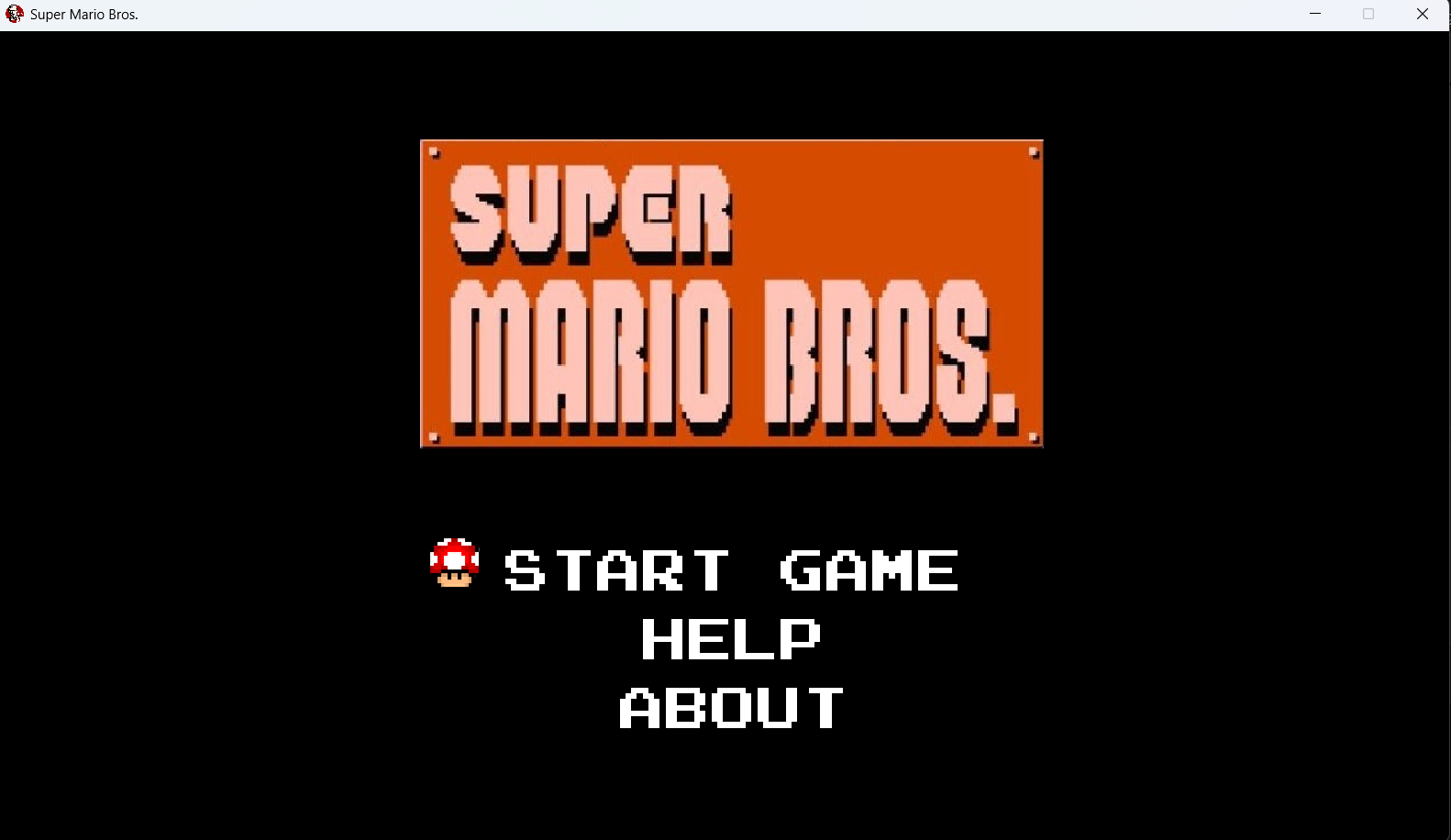
# **CHAPTER 4: FINAL APP GAME EXPERIENCE**

**Demo video:**[**https://youtu.be/phLXYV4G11w**](https://youtu.be/phLXYV4G11w)

## Instruction:

1. *Click on the icon and the start screen (main menu) will appear like the picture* 

*under:*



**Move to Map selection**

**HOW TO PLAY**

1. *The main menu includes 3 main parts: START GAME (choose to move to the next selection map menu), HELP (providing player instruction and how to play the game as well as button), AND ABOUT (the game creator).*

* ***HELP SCREEN:***
* ***ABOUT SCREEN:***



1. *MAP SELECTION SCREEN (Map 1, Map 2, Map 3).*



1. *After map selection, the Mario game will officially start*

*In Game:*



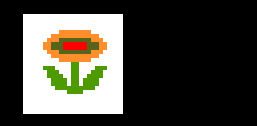
**Lives Left**

**Points acquired**

**Time left**

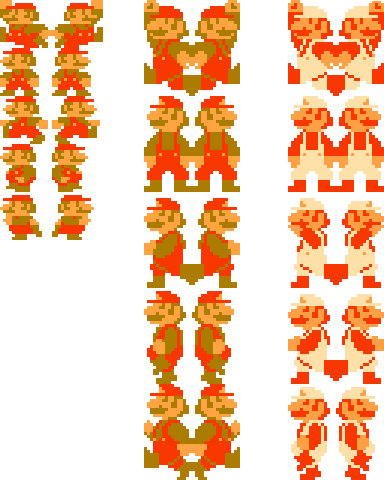


**Surprise Bricks: Let Mario Head touch this brick then the following prizes will appear**

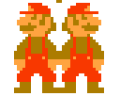
* Coin : collect and acquire 1 coin in your bank account.
* Fire Flower collect and acquire 150 points and turn your Mario character form into fire form (Mario Fire form allows you to shoot fireballs into enemy and delete them by pressing spacebar).
* One up mushroom acquires 200 points and plus 1 for your lives (remaining lives in java).

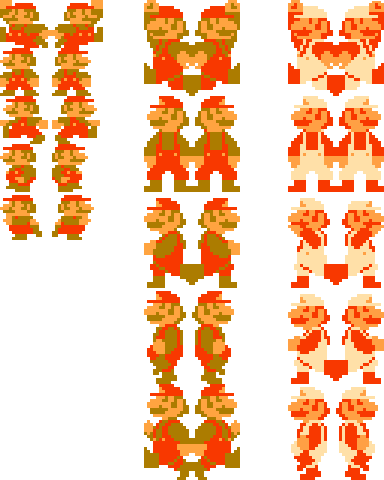


* Super Mushroom acquire 125 points and turn your Mario character form into Super Mario (which Mario can break the brick by touching Mario's head to the bottom of the brick and vulnerable while touching the enemy). Mario will be back to normal after touch 1 enemy.

Mario Form:

* Normal Form:



* Super Form vulnerable to the first touch to enemy then return to normal form
* Fire Form can shoot fireballs from his pants, vulnerable to the first touch to enemy then return to normal form.

# **REFERENCES**

Images:

Wiki, S. M. (2024, December 15). Super Mario Bros. - Super Mario Wiki, the Mario encyclopaedia. Retrieved from: <https://www.mariowiki.com/Super_Mario_Bros>

Sound effects:

Wiki, C.T.S. (n.d.). Super Mario 64.   
Retrieved from: <https://soundeffects.fandom.com/wiki/Super_Mario_64>

YouTube video that we learned:

[(777) Java Game Programming - Super Mario Bros - YouTube](https://www.youtube.com/playlist?list=PLAST8FJClRlmRxbwwJg2AVpG3JSISpnX4)