
Bilkent University

Department of Computer Engineering

# Object Oriented Software Engineering Project

*CS319 Project Group 1G: Super Mario Bros.*

## Design Report

Ahmet Çandıroğlu, Unas Sikandar Butt

Course Instructor: Uğur Doğrusöz
Course Assistant: Hasan Balcı

Progress/ Iteration 1

Oct 21 2017

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfilment of the requirements of the Object Oriented Programming course CS319/1

# 1 Introduction

## 1.1 Purpose of the System

Super Mario is a classic game known to almost every person from the ‘90s. The main purpose of the system is to provide a similar game, with minor changes. The game can be classified as an adventure and level based game. Comparing to current standards of gaming, our system may be lacking in the graphics and complexity departments, nonetheless the game will provide a challenging gameplay which is addictive and really enjoyable. The difficulty level of the game is intentionally going to be kept hard so that the game can become a platform for gamers to compete on.

## 1.2 Design Goals

Reliability

The game should be reliable in the sense that it should be able to handle any problems. For example if the CPU is too crowded the game should report a warning message or if the keyboard or mouse is not connected there should be an error message.

Extendibility

The best of systems are the ones which are extendable. The game will be implemented in such a way that will permit future additions and development. This way the game can be improved and new features can be added to the current system.

Portability

We want the game to be available to as many users as possible. So portability is a must for the system. The game will be developed in the java environment which is supported by most modern machines today. This will enable us to target greater clients.

Rapid development

This project has a time constraint of one semester. Taking that into account rapid development has to be carried so that the project can be finished in time.

Minimum number of errors

Ideally the game has to be completed without a single error. However we know that we may not be able to accomplish that, so one of the design goals is to have a minimum number of errors so that the system can be as close to perfection as possible.

Ease of learning

The user of any game has to first understand how to play the game. Most modern games provide a tutorial at the start of the game so that the player can get accustomed to the game. Our game however, if compared to most modern games, is very simple. So a simple controls menu should be enough for the player to understand how the game is to be played.

Modifiability

As our system will be implemented using the Object Oriented Programming principles, the system will be kept modifiable. During the design stage we will try to minimize coupling and maximize coherence. This in turn will make it easier to modify any given features of the game.

# 2 Software Architecture

Before designing the system the most suitable architectural style has to be chosen. For our project the MVC architecture is the most suitable. The MVC stands for Model View Controller. This style of architecture also comes with its own advantages:

1. Faster development

In this architecture the system undergoes system decomposition, this allows for a faster and parallel development process.

2. Modification

Using MVC enables modifications to subsystem parts without affecting the other parts. This makes future updates to the system easier.

## 2.1 Subsystem Decomposition

We decided to use MVC design pattern so we divided our project into three main parts: Model, view and controller. Model classes represents objects in the game. These classes only state locations, types, abilities etc. View classes renders what should be rendered according to Controller's inputs to it. Controller classes gathers input and makes calculations according to its methods.

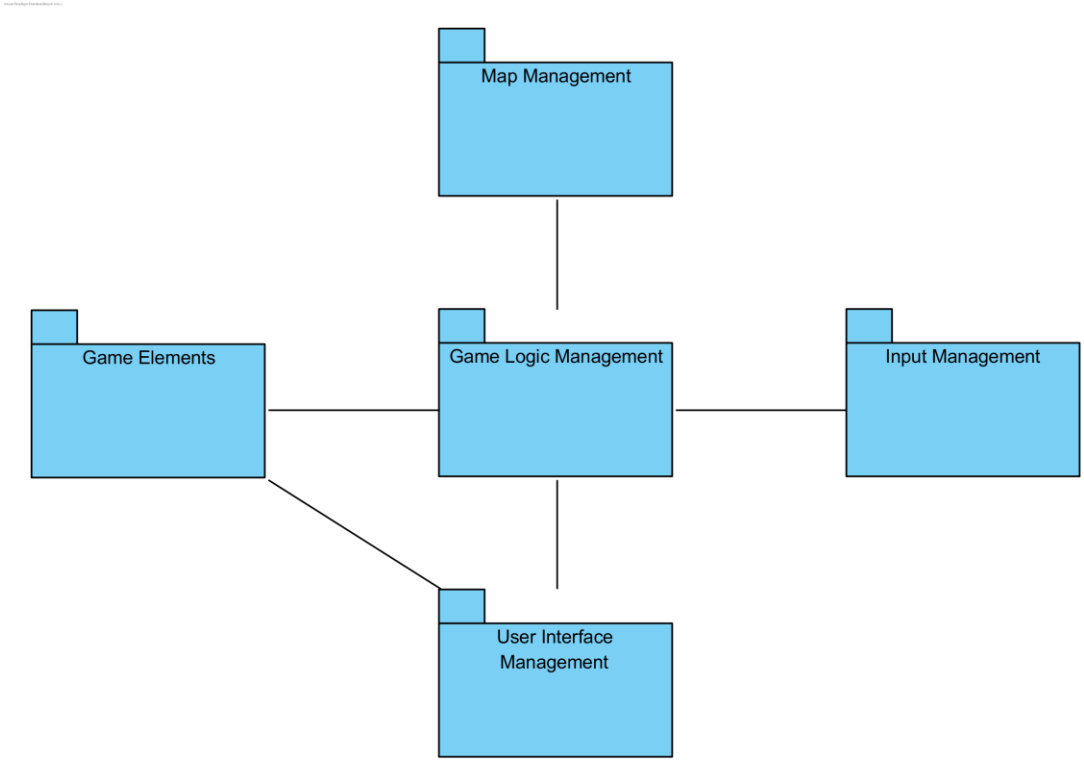


Figure 0 : High-level Composition Diagram

Game logic management represents controller side of the design which controls everything and informs other management systems. Input management connects user and the game. User Interface Management system is the view side

of the design and only renders the game onto screen. And finally Game Elements are model side of the MVC design pattern.

## 2.2 Hardware/Software Mapping

As the game will be developed in a java environment first and foremost a java compiler will be required which can compile and run the .java file. For hardware specification, a keyboard is required and a mouse (optional). No internet connection is required to play the game. As for the computer any modern computer with average specifications should be able to run the game. A graphics processing unit (GPU) can be optional just to further enhance the performance of the game. In the OS criteria any OS which supports java environment will be compatible. Also for the data storage the computer hard disk will be used to store and load the save game data.

## 2.3 Persistent Data Management

Most of our in game data consists of the map, Mario, monsters and bricks. To increase performance these objects will not be drawn instead they will be kept on the hard disk as image files and fetched directly. We hope this will make the game much smoother. Other than game image files we will also have some sound files for the in game music and sound effects. These files will have the .wav file format. Apart from these as mentioned earlier the project will also have some saved game files. These files will be kept in the .txt file format. To aid the simplicity of the project no complex database system will be used.

## 2.4 Access Control and Security

Playing this game the user will have no issues with the security of the system. The access control will be with all the users on the machine, be it administrator or otherwise. Also there should be no chance of game files going corrupt as the user or any third party software should not be able to modify them. Also as the game needs no internet connection there is no chance of any viruses or malicious files being downloaded.

## 2.5 Boundary Conditions

Initialization

The game will be executed using a .jar file (java archive). The .jar file will contain all the class files and resources (texts, images). The .jar file is a suitable distribution method because it contains all the data needed for the game in one file. Also there is lesser chance of an individual file getting corrupted. At start up the program will access the user saved games files and user configuration files and load them. As the user starts the program he/she will be shown the main menu page, where the user can load a previously load game, start a new game, view controller options or change game settings.

Termination

The user can terminate the program by using the exit button on the main menu. Alternatively, if the user wishes to exit while playing the game they can use the pause button wish displays the pause menu. From the pause menu the user will have to first go to the main menu and then exit from there. Another method of termination can be directly from the system task manager.

Error

Our program being very simple should have minimum possible errors. Since we are not dealing with a system that does parallel processing or has multiple users working at the same time or something like client/server architecture or service-oriented architecture, our system will not have many complicated errors. The online errors we may face will be run time execution failing or corrupted game files. If the system running the game has a low processing power or low random access memory the game may also face some errors or may get stuck. However, as modern systems have average processors ranging from 2.0-3.0 GHz these problems may never occur.

# 3 Subsystem Services

## 3.1 Detailed Object Design

Class diagram for whole system is shown in next page. It is divided into subsystems which makes it easy for people to develop.

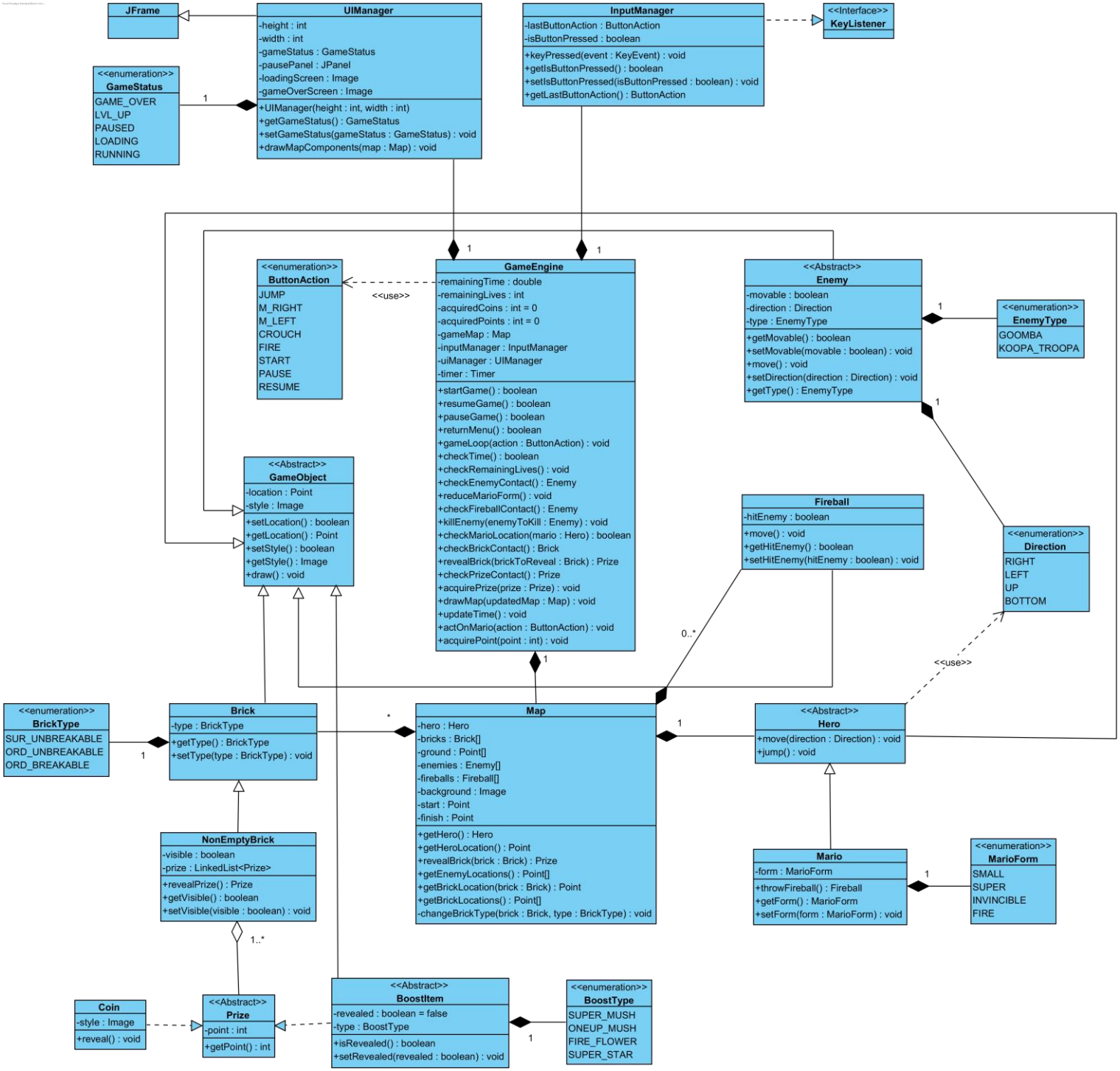


Figure 1 :Object Diagram

## 3.2 User Interface Management Subsystem

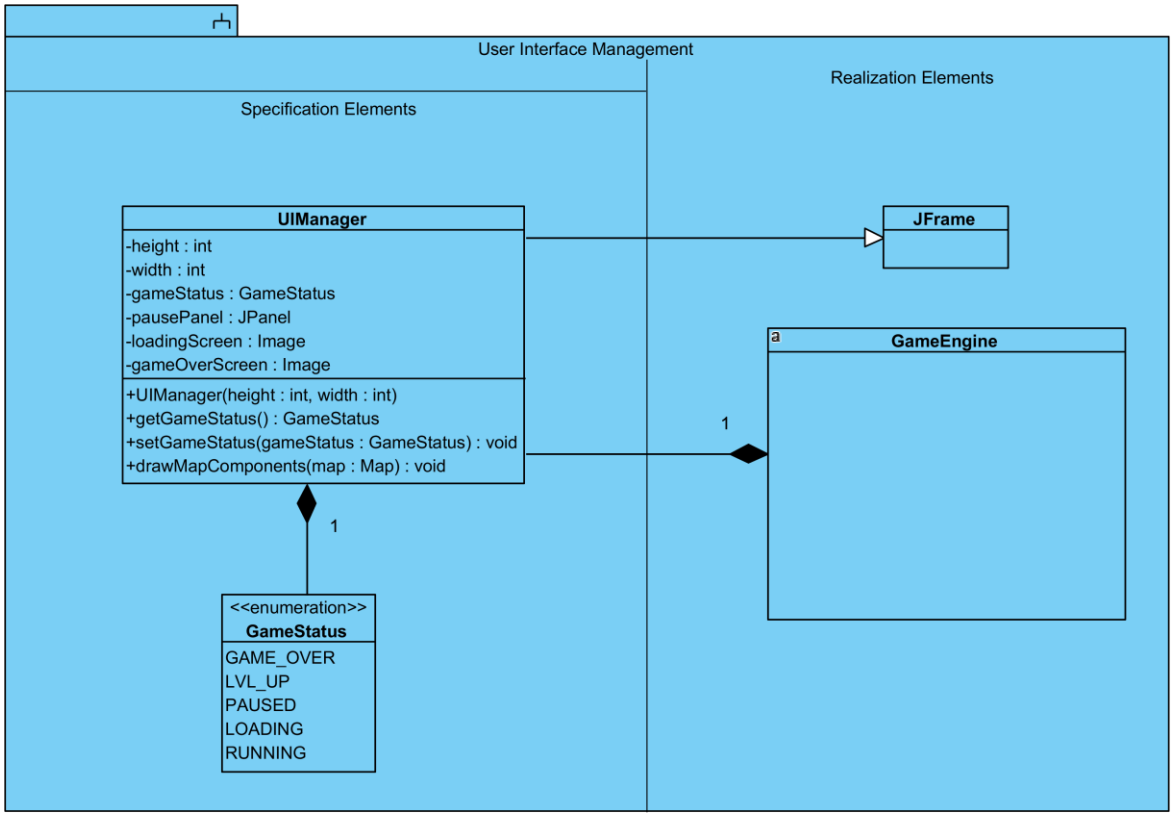


Figure 2 :UI Management Subsystem

User Interface Management Subsystem consists of one main class which is UIManager, an enumeration GameStatus and a realized class, GameEngine. UIManager class extends javax.swing.JFrame. It has height and width properties which defines frame size. The game has many statues so a GameStatus enumeration is defined. According to the GameStatus frame will be rendered differently. If game is running then normal game map will be rendered. Else if game is on pause, pause menu panel will be rendered and so. Game status of UIManager can be changed via its set method. This can be done by GameEngine which is the controller of the game.

## 3.3 Game Logic Subsystem

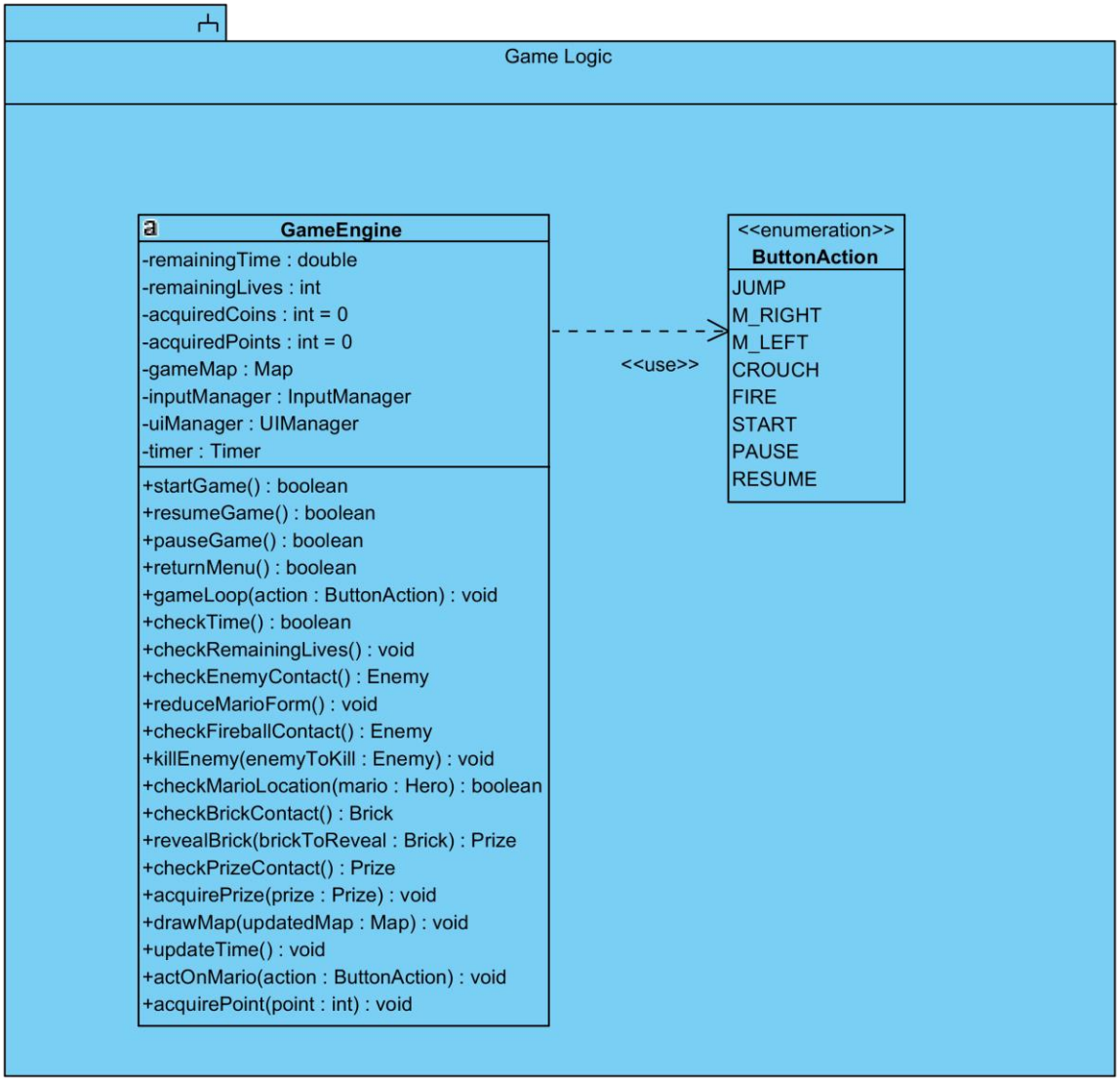


Figure 3 :Game Logic Subsystem

Game Logic subsystem consists of only one class which is GameEngine, main class of the game. GameEngine is the controller class of our game which is MVC modelled. So, this class gathers information from outside, makes calculations and let's view class to know everything user should know. To be specific, GameEngine has an instance of InputManager class which gets the input from user via keys (mouse inputs are not included to our game maybe in next iteration). There are several action can be created by user and that is why

ButtonAction enumeration is needed. There are actions like JUMP, M_RIGHT, M_LEFT, FIRE and CROUCH which are hero controlling actions. Others like START, PAUSE, and RESUME are general actions about game.

GameEngine gets last input (if there is any) in each game loop and changes position of hero or game status accordingly. GameEngine calls gameLoop() method in each loop which calls several methods to make sure either game is going or not. First remaining time is checked via checkTime() method. If there is time left then remaining lives are checked. If there is any life remaining then given action can be processed. If given action is hero related action it is passed on actOnMario() method and Mario's position will be changed. After that function, several checks must be done.

- Enemy contact, checkEnemyContact() - If the hero touches the enemy one of its lives burns. If the hero smashes the enemy then enemy will be killed by calling killEnemy() method and that enemy will be given as an argument.
- Brick contact, checkBrickContact() – If the hero touches any brick from the bottom, touched brick will be returned. After then revealBrick() method will be called to reveal the brick.
- Prize contact, checkPrizeContact() - If the hero touches any prize then the prize will be return and acquired by the hero.

After all these checks remaining time will be updated by updateTime() method. Now it is time to update view and drawMap() method will be called and updated map (gameMap property) will be given as an argument to this method.

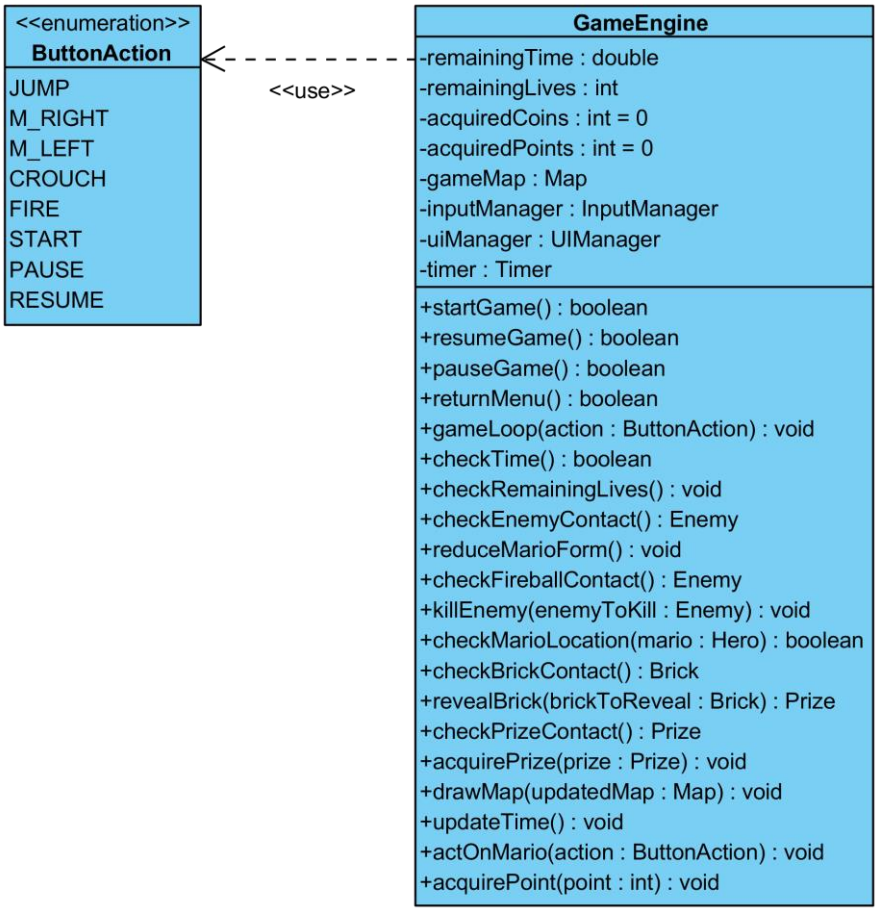


Figure 4 :Game Engine Class

## 3.4 Game Screen Elements Subsystem

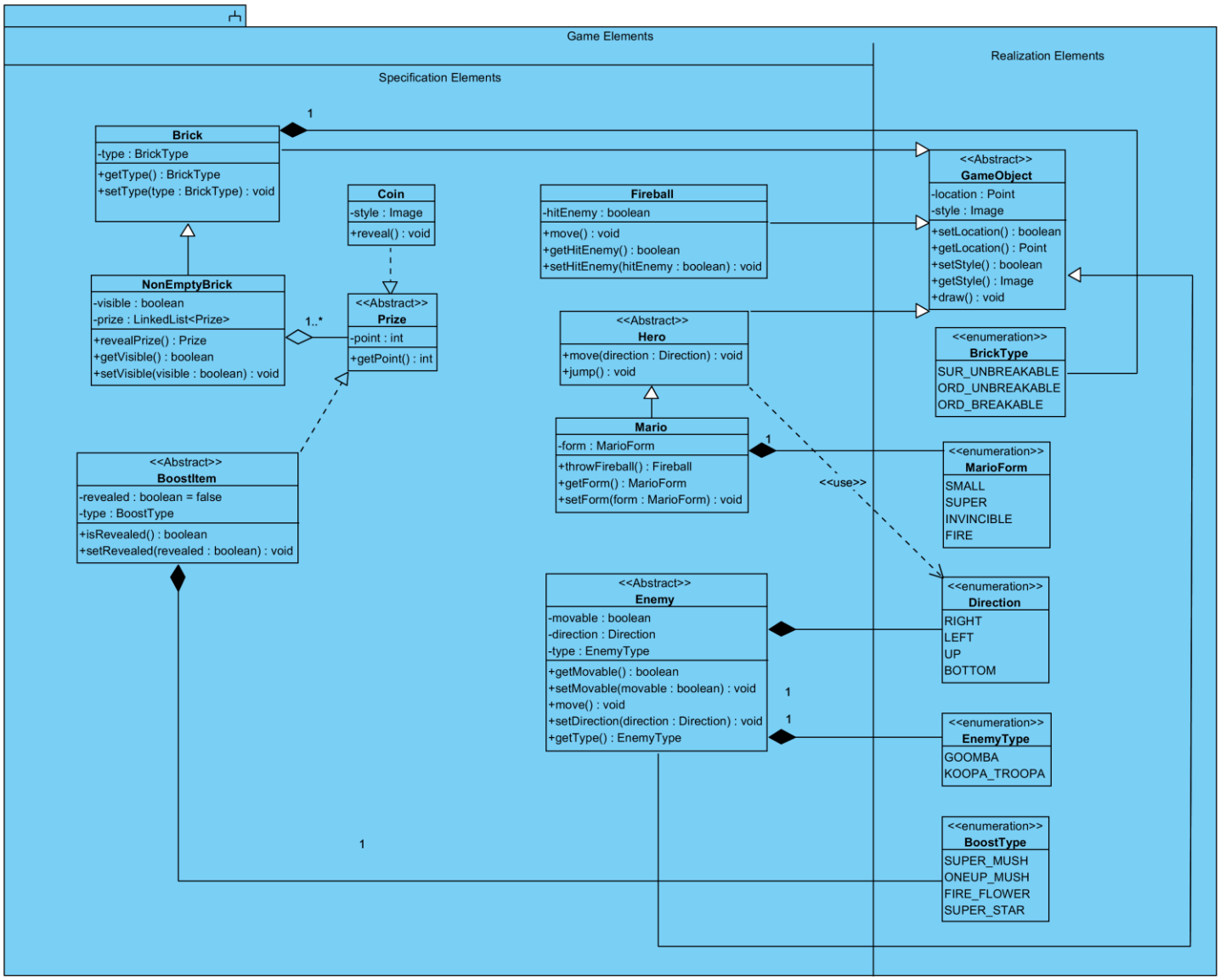


Figure 5 :Game Elements Subsystem

Game elements subsystem consists of model classes which represents all the objects in the game. This subsystem can be divided into parts which makes it easy to examine whole subsystem.

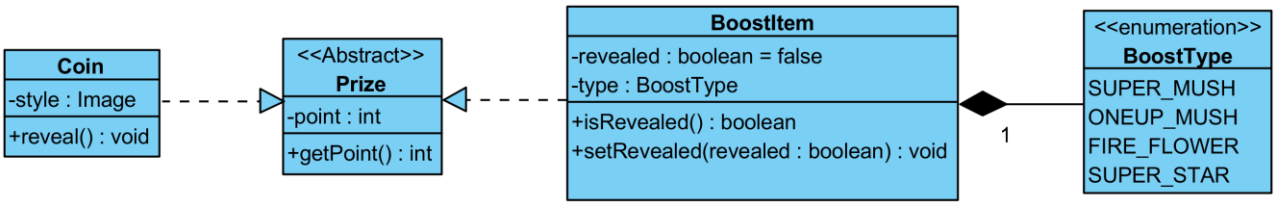


Figure 6 :BoostItem Classes

BoostItem represents mushrooms and other boost items which change form of Mario. There are many type of BoostItem which is represented by BoostType. Types define what form will Mario have when he acquire the BoostItem. Prize is an abstract class and is parent of Coin and BoostItem. Coin can be acquired by the hero which gives points.

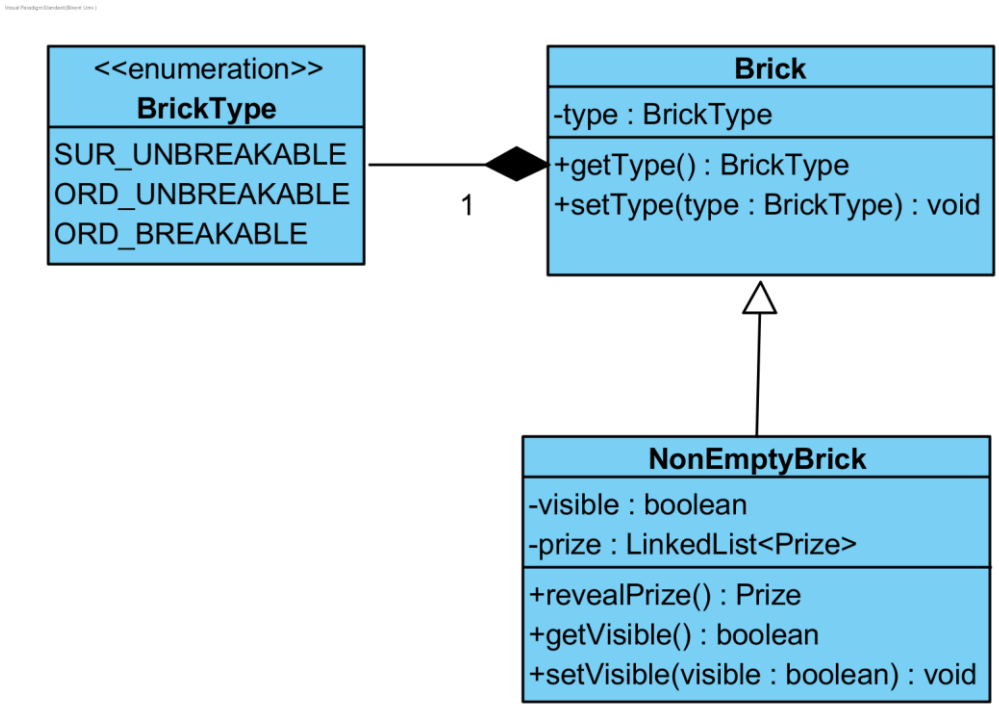


Figure 7 :Brick Classes

Brick class represents bricks on the map which has lots of types and that's why BrickType enum is used. Brick may contain Prize or not and it can be breakable or unbreakable by Mario when he is in super form depending on its type. Brick type and style can be changed by hitting from bottom.

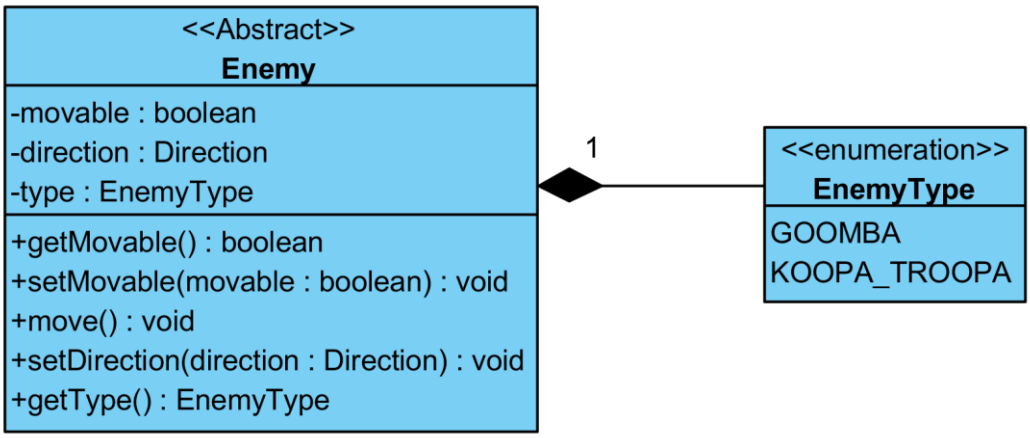


Figure 8 :Enemy Class

Enemy is as the name implies enemies of the hero which can be killed by smashing. They can kill the hero in case the hero touches them. There are many EnemyTypes and these types define enemies' movability, direction of move etc.

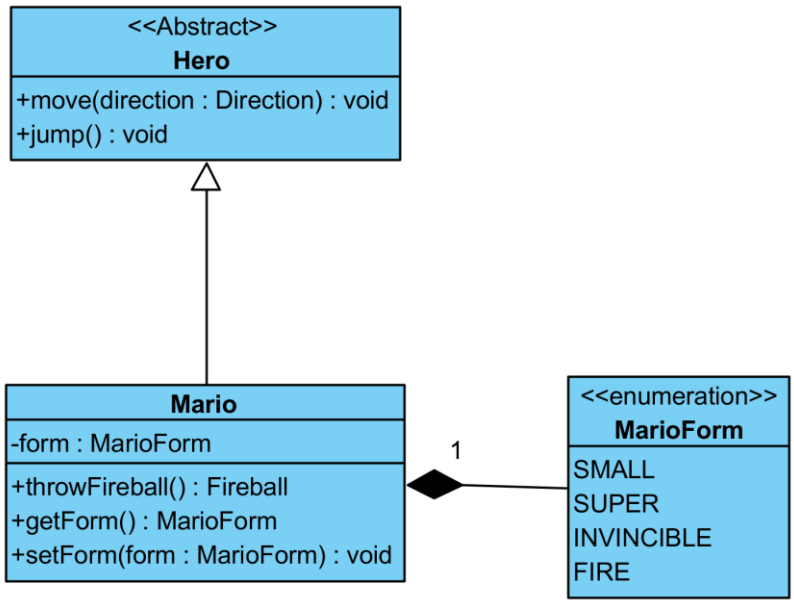


Figure 9 :Hero Classes

Hero is an abstract class which is parent of heroes like Mario which will be controlled by user. Hero can move in one direction in one time and can jump. Mario has different types of forms which defines his strength. Mario can break breakable bricks in SUPER form, can throw fire in FIRE form and is invincible in INVINCIBLE form in a brief time period. His form will change according to enemy contact (if he touches enemy he returns to small form, if he is in small form he loses one life) and obtained BoostItems.

## 3.5 Input Management Subsystem

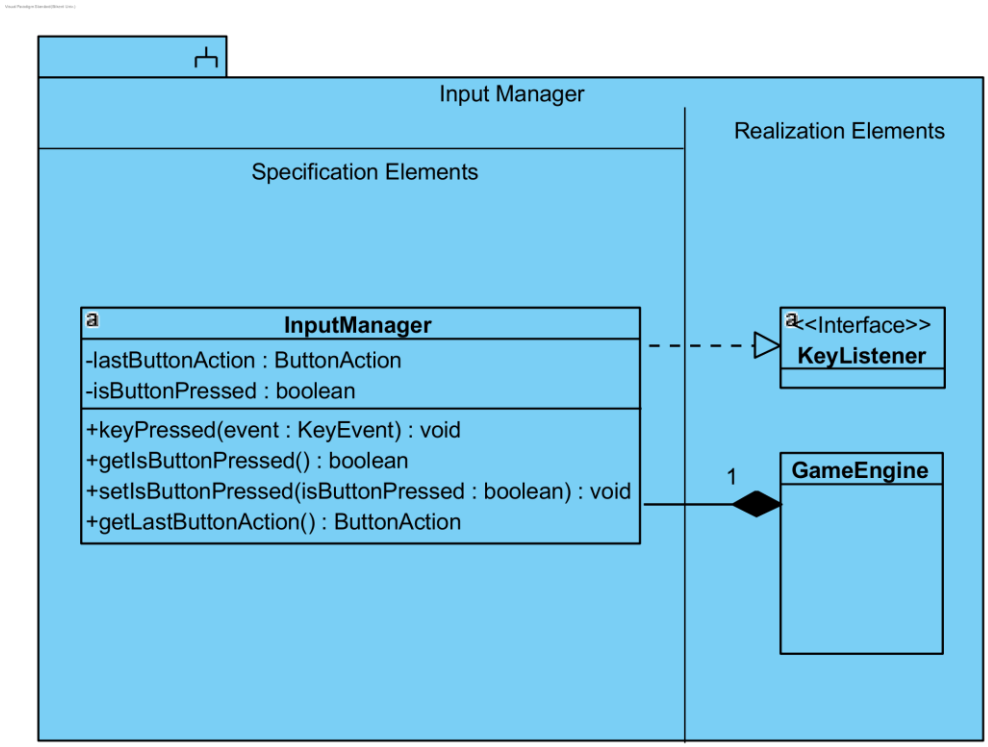


Figure 10 :Input Management Subsystem

Input management subsystem consists of only InputManager class. InputManager gets input from the user and records the last button action in its property. GameEngine gets given input from InputManager and updates game accordingly. InputManager extends KeyListener of Java and overrides keyPressed() method.

## 3.6 Game Map Management Subsystem

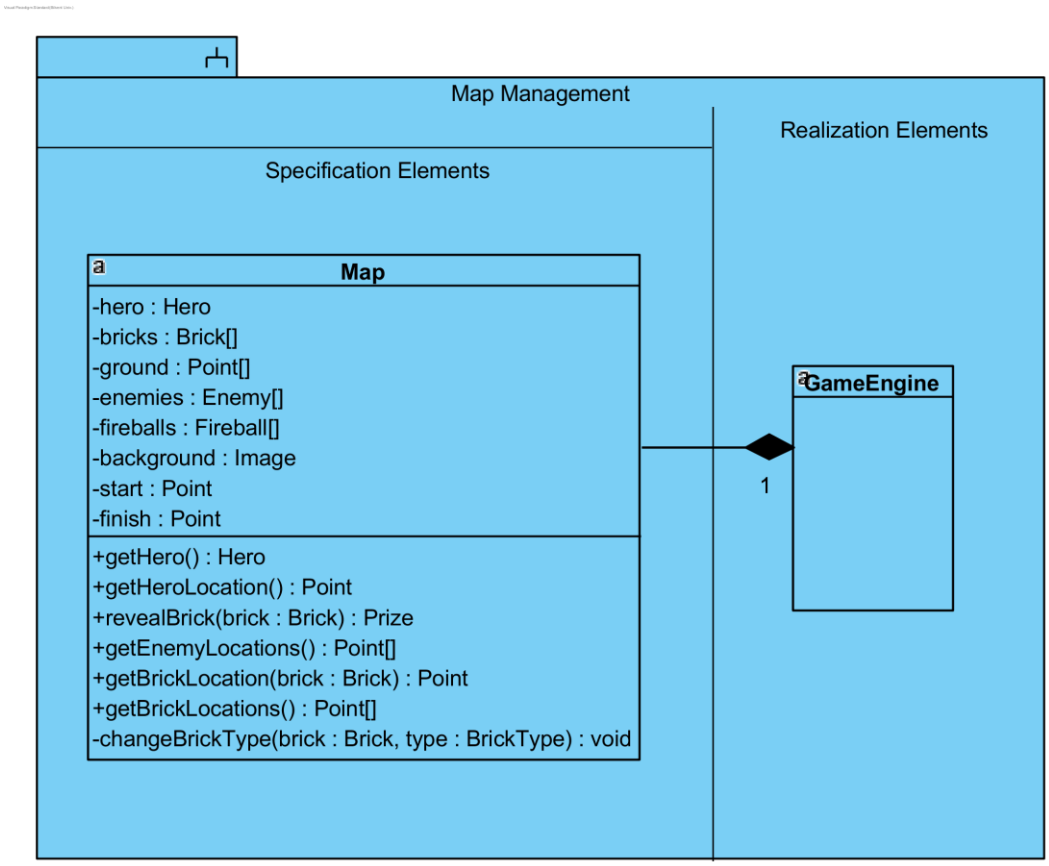


Figure 11 :Map Management Subsystem

Map class contains everything that can be seen in a map. There is only one Hero which user can control. Bricks are defined according to type of the Map however in this iteration we have only one type of map. Ground property is an array of Points which defines the ground of the map where can the hero and other GameObjects can go. Brick types can be changed by changeBrickType() method.

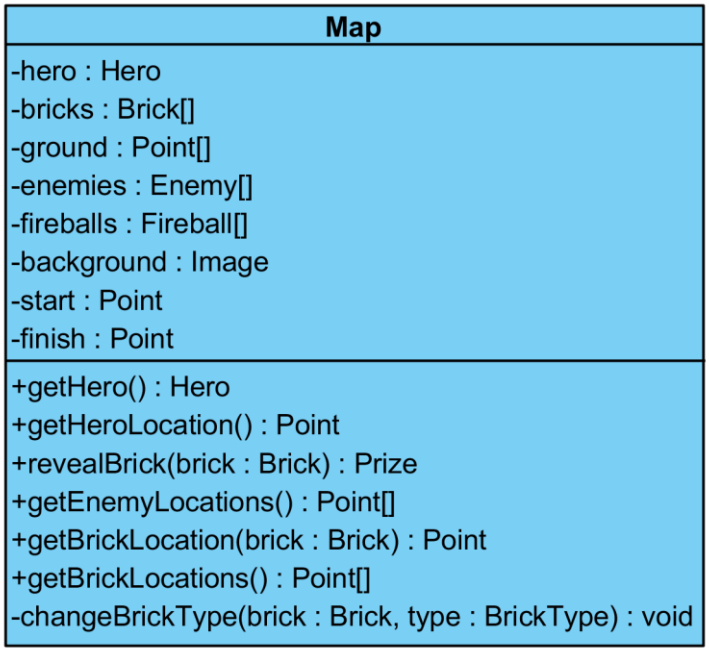


Figure 12 :Map Class

## 3.7 Description of the Interactions between Classes According to the Use Cases

**PlayGame:** Ahmet wants to play this marvelous game and presses shortcut of this game. GameEngine which is the main method of the game is initialized and initializes other classes. After UIManager is initialized Ahmet can finally presses Start game button and then InputManager calls startGame() method of GameEngine. GameStatus of the UIManager changes into RUNNING and gameLoop() method is called for each loop in the game until game is over. Ahmet jumps and moves Mario until he get SUPER type BoostItem. GameEngine checks many things such as enemy contact and brick contact. GameEngine finds a contact between brick and Mario and then it calls revealBrick() method with that specific brick as an argument. Then that brick will reveal the Prize which is inside it with reveal() method of the prize (SUPER BoostItem). When the prize is reveal Ahmet moves the Mario and he touches the prize and acquires the price with acquirePrize() method. After that Mario will turn into SUPER form via setForm(MarioForms.SUPER) method call.