Bilkent University

Department of Computer Engineering

CS319 – Object Oriented Software Engineering Project

Donkey Kong Game

Design Report

Group 3E

Fuad Ahmadov

Çağatay Küpeli

Sine Mete

Arkın Yılmaz

Instructor: Bora Güngören

Design Report Draft

October 21, 2017

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Object Oriented Software Engineering CS319/3.

Contents

[1 Introduction 1](#_Toc254263637)

[**1.1 Overview 1**](#_Toc254263645)

[**1.2 Purpose of the System 1**](#_Toc254263645)

[**1.3 Design Goals 1**](#_Toc254263645)

[**1.4 Definitions, Acronyms, and Abbreviations 3**](#_Toc254263645)

[2 Software Architecture 3](#_Toc254263638)

[**2.1 Subsystem Decomposition 3**](#_Toc254263645)

[**2.2 Hardware / Software Mapping 6**](#_Toc254263645)

[**2.3 Persistent Data Management 7**](#_Toc254263645)

[**2.4 Access Control and Security 7**](#_Toc254263645)

[**2.5 Boundary Conditions 7**](#_Toc254263645)

[3 Subsystem Services 8](#_Toc254263639)

[**3.1 Design Patterns 8**](#_Toc254263645)

[3.1.1 Facade Design Pattern (Encapsulating Subsystems) 8](#_Toc254263647)

[**3.2 User Interface Subsystem 9**](#_Toc254263645)

[3.2.1 mainFrame Class 10](#_Toc254263647)

[3.2.2 mainMenu Class 10](#_Toc254263647)

[3.2.3 optionMenu Class 11](#_Toc254263647)

[3.2.3 levelSelect Class 12](#_Toc254263647)

[**3.3 Input Management Subsystem 13**](#_Toc254263645)

[**3.4 File Management Subsystem 14**](#_Toc254263645)

[3.4.1 gameData Class 14](#_Toc254263647)

[**3.5 Game Algorithm / Game Visual Subsystem 15**](#_Toc254263645)

[3.5.1 gameEngine Class 16](#_Toc254263647)

[**3.6 Game Entities Subsystem 18**](#_Toc254263645)

[3.6.1 Game Screen Elements Subsystem 18](#_Toc254263647)

[4 References 21](#_Toc254263651)

System Design Report

1. **Introduction**

**1.1 Overview**

In this section, we are tried to imply our main goals and concerns on different topics. It was clear from this section that our main goal is to make user-friendly game, and entertain player. In addition, as it is designated in these sections, we are going to focus on ease of use, efficiency, performance in the most reliable way.

* 1. **Purpose of the System**

Donkey Kong is 2-D arcade game to entertain users by trying to reach finish with avoiding obstacles. Donkey Kong is very challenging game which increases pleasure of the game. On the other hand game is also very user friendly because user do not have to use many controls in other words buttons to control the player. Although, old fashioned Donkey Kong was very difficult, we reduced difficulty for first levels, and game gets more difficult in the upcoming rounds which keeps player focused on the game.

* 1. **Design Goals**

**Usability:**

Difficult controls in software, especially in games is a big burden for a user. Therefore, usability is a big factor in software to keep user interested.

**Ease of Use:**

Our game will be designed for the user to be as user-friendly as possible. We are not going to use complicated controls, or any type of joystick to play our game. Additionally, system will provide user-friendly menu interfaces, by which player will navigate wherever he/she wants easily. We are planning to use arrows and space bar as controls.

**Ease of Learning:**

Although, Donkey Kong is well-known game, player should learn information about controls, power-ups which is vital for having the best experience from the game. If the player wants to learn all information, he/she will be able to learn from the instructions page.

**Portability:**

Portability is an important topic for any software. Therefore, for providing cross-platform portability Java®is the one of the best programming languages. Additionally, using Java®Programming language provides us wide range of users. That’s why, our software will have no problem about portability.

**Reliability:**

Our game will work without crashing and having any bugs. In addition, player will not have any unexpected inputs through the game. Reliability of the system will be tested through the development of the system to not have any bugs and crashes at the end.

**Extendibility:**

For any type of game software, adding new components, features to game as the enough time passes. Therefore, we will add some new features to the game, in other words we are going to update our game with some features. Although, our game is old-fashioned arcade game, we have some ideas to add to the game in the upcoming updates.

**Reusability:**

We are not planning to integrate any of our classes or system with other game or similar system. Therefore, we are not planning to think about reusability in the design process of the game.

**Performance-Memory**

One of our main concerns is to make our game as smooth possible in animations, effects, transitions. In other words, we are going to try keep Fps as high as possible so our player will have the best gaming experience because sudden Fps drops and freezes in games are very user-disturbing facts. Additionally, we will have few collision effects which helps us to keep memory low as possible.

**1.4 Definitions, Acronyms, and Abbreviations**

**Abbreviations**

**FPS:** Frame per second. This abbreviation represents number of graphical frames has been prepared in 1 second.

**Cross-Platforms:** Getting same performance from the different Operating Systems including Windows, Linux, and MAC OS X.

**2 Software Architecture**

**2.1 Subsystem Decomposition**

We chose the three-tier architectural style as a basis for our design because it is the most convenient architectural style to integrate with our system. The three-tier architectural style mainly consist of three layers: an interface layer, an application logic layer, and a storage layer. In the figure shown below, the high-level representation of our system decomposition is found.

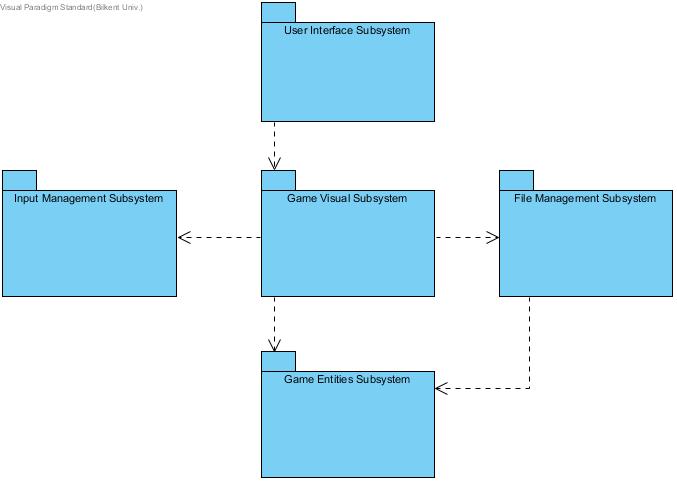
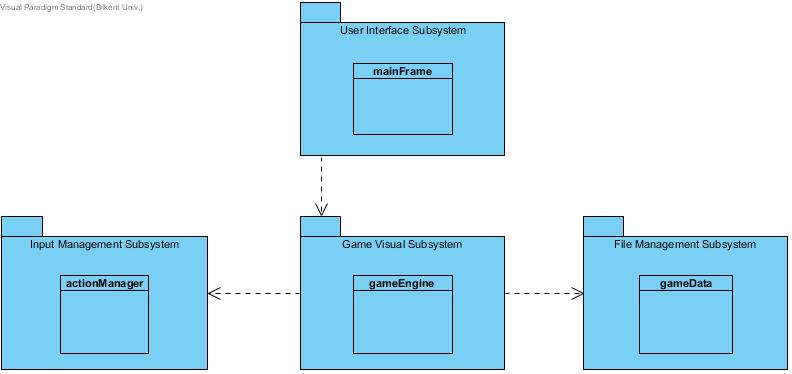


Figure 2.1.1: High-level Representation of Subsystem Decomposition

The interface layer contains all boundary objects which interacts with the user. Therefore, we placed "User Interface Subsystem" package in the interface layer. Inside this package, we put "mainFrame" class, which contains main menu in it. When user wants to start game, first he/she should select "Play Game" option and then select the desired level to start. The chosen level number will be transferred to "Game Visual Subsystem" package.

Figure 2.1.2: Interaction between Interface Layer and Application Logic Layer

The application logic layer contains all control objects. Thus, inside this layer, user's selection will interpreted and the "gameEngine" class will create the game map with cooperating "gameData" class which placed in "File Management Subsystem" package. Also during the game, "gameEngine" class will recieve inputs from "actionManager" class placed in "Input Management Subsystem" package.

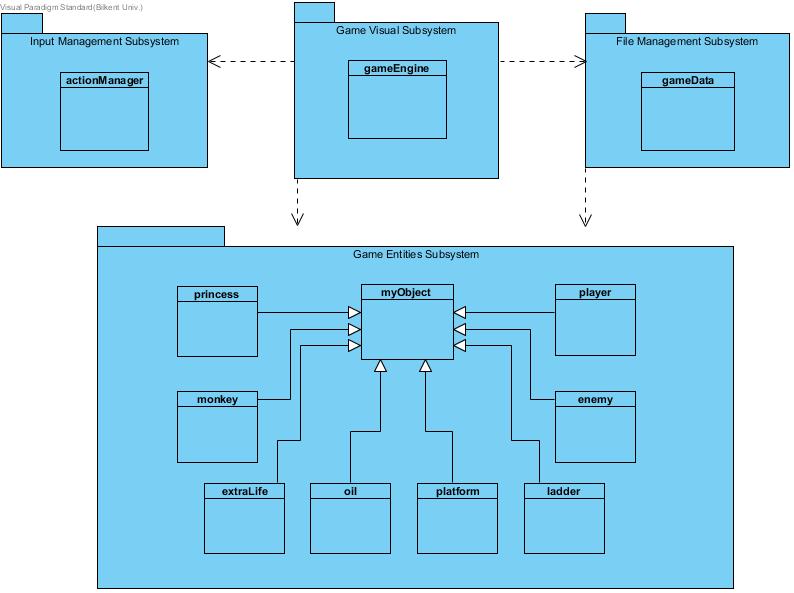


Figure 2.1.3: Interaction between Application Logic Layer and Storage Layer

The storage layer contains realizes the storage and retrieval of persistent objects. So, we placed "myObject" super class and subclasses into "Game Entities Subsystem" package. These objects will be used by both "gameEngine" and "gameData" classes during rendering operations and gameplay.

**2.2 Hardware / Software Mapping**

Our game will be implemented in Java so, in order to run it, the latest version of Java Runtime Environment will be required. In terms of hardware configuration, the game only requires a keyboard to make selections in the menu's and to play game. As system requirements, an average computer with basic softwares will be enough to support our game.

**2.3 Persistent Data Management**

Donkey Kong is a simple retro arcade game. Using database would make the game bulky. Therefore, text files will be used in this project. Game data will be stored in hard disk drive. Moreover, we will load all the necessary files on to the memory and access those files when the gameengine or the rendering system requires. The files are; the background images, images of the game elements will level specialities and high score list as text files in disk. Also, sound effects will be stored.

**2.4 Access Control and Security**

Donkey Kong does not require any internet connection or creation of user profiles. Files must be installed in order to play the game. Therefore , there will not be security issues in Donkey Kong. GameEngine class is the only one can reach the files on system according to user actions. This provides security.

**2.5 Boundary Conditions**

The game is placed on and transferred by an executable .jar file. The game will not be fullscreen because it is rasterized and characteristically small. The first screen is the menu screen and specifies the boundaries to the user. If all the lives of the player are depleted, game will end and return to the main menu. If the user completes the game, game will end and return to the main menu. At the end of the game, high scores will be updated according to the score. Donkey Kong can be terminated by clicking quit game button. If the user opens the program again while it is already running, program terminates. If program does not respond because of any reason, the program will be terminated and the data will be lost.

**3 Subsystem Services**

**3.1. Design Patterns**

**3.1.1 Facade Design Pattern (Encapsulating Subsystems)**

Facade Design Pattern is a structural design pattern that aims to reduce coupling between a set of related classes and the rest of the system. To do that, a single Facade class implements a high level class for a subsystem by invoking the methods of lower level classes. Facade provides a shield for caller so that it does not access the lower level classes directly. (BRUEGGE & DUTOIT, 2010)

In this project, Facade class is gameEngine which can be seen under Game Algorithm / Game Visual Subsystem. gameEngine communicates and associate other lower level classes.

**3.2. User Interface Subsystem**

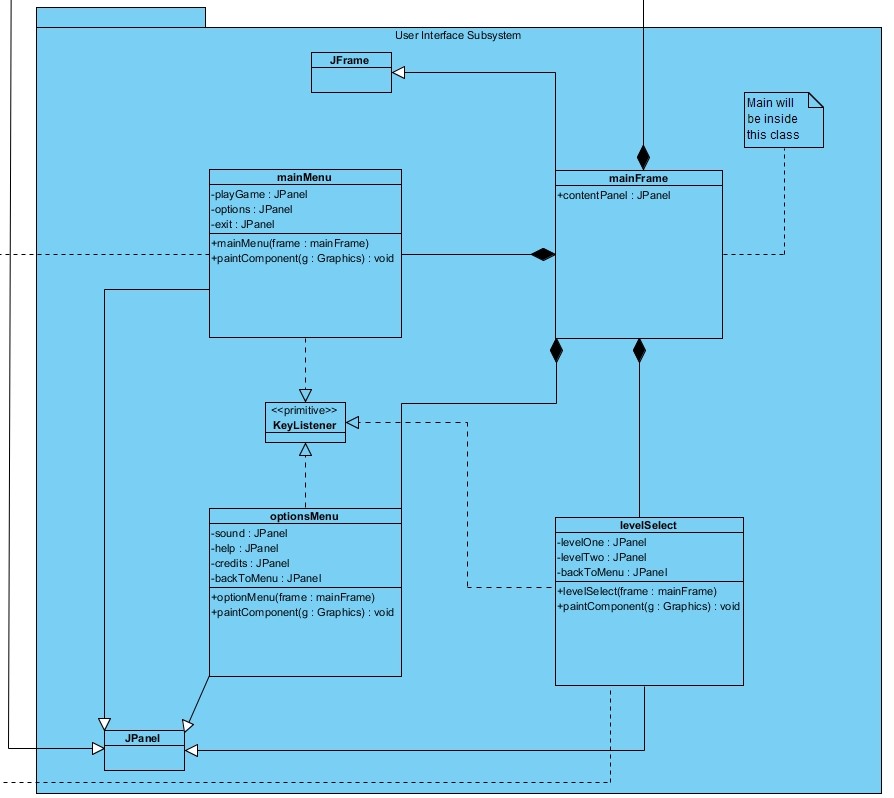


Figure 3.2: User Interface Subsystem

User Interface Subsystem is consisted of 4 different classes which provides graphical components for menu screens. In the current iteration of the design gameEngine handles the game screen. It also implements JPanel; however, it also interact with other classes. Therefore it will be analyzed under Game Algorithm / Game Visual Subsystem.

**3.2.1. mainFrame Class**

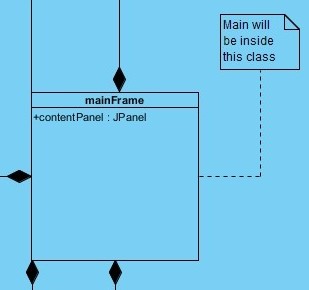


Figure 3.2.1: mainFrame Class

**Attributes**

Public JPanelcontentPanel: This attribute will initilize the border of the content which mainFrame will take from optionMenu, levelSelect, mainMenu and gameEngine.

**Constructors**

Public mainFrame: Initialize the bound, layout and tittle of the window.

**Design Choices**

The main method of the project will be called by mainFrame class.

This class will extends JFrame.

**3.2.2. mainMenu Class**

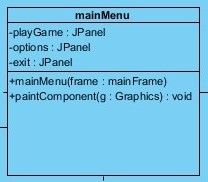


Figure 3.2.2: mainMenu Class

**Attributes**

PrivateJPanelplayGame: This attribute will be work like a button. When the user presses the appropriatekeylistener, it will call the levelSelect class.

PrivateJPanel option: This attribute will be work like a button. When the user presses the appropriatekeylistener, it will call the optionMenu class.

Private JPanel exit: This attribute will be work like a button. When the user presses the appropriatekeylistener, it close the program.

**Constructors**

Public mainMenu: Load the panels and images.

**Methods**

Public void paintComponent(Graphics g): It will draw the screen by using Java Graphics2D library.

**Design Choices**

For Donkey Kong type of arcade games, client usually uses keyboard listeners to move around selections in the menu. Thus, JPanels will be used instead of JButtons. The panel will glow to indicate that user is on that segment of selection.

This class will extends JPanel and implements KeyListener.

3.2.3. optionMenu Class

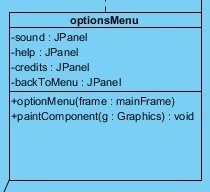


Figure 3.2.3: optionMenu Class

**Attributes**

Private JPanel sound: This attribute will be work like a button. When the user presses the appropriatekeylistener, it will mute the sound effects. Therefore it will glow according to current selection.

Private JPanel help: This attribute will be work like a button. When the user presses the appropriatekeylistener, it will pop up another panel to show how to control the Jumpman.

Private JPanel credits: This attribute will be work like a button. When the user presses the appropriatekeylistener, it will pop up another window to show the credits with close option.

Private JPanelbackToMenu: This attribute will be work like a button. When the user presses the appropriatekeylistener, it take you back to mainMenu class.

**Constructors**

Public optionMenu: Load the panels and images.

**Methods**

Public void paintComponent(Graphics g): It will draw the screen by using Java Graphics2D library

**Design Choices**

This class will extends JPanel and implements KeyListener.

3.2.4. levelSelect Class

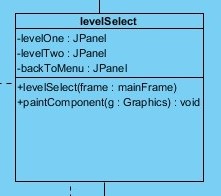


Figure 3.2.4: levelSelect Class

**Attributes**

Private JPanellevelOne: This attribute will be work like a button. When the user presses the appropriatekeylistener, it will call the gameEngine class and load proper level.

Private JPanellevelTwo: This attribute will be work like a button. When the user presses the appropriatekeylistener, it will call the gameEngine class and load proper level.

Private JPanelbackToMenu: This attribute will be work like a button. When the user presses the appropriatekeylistener, it take you back to mainMenu class.

**Constructors**

Public levelSelect: Load the panels and images.

**Methods**

Public void paintComponent(Graphics g): It will draw the screen by using Java Graphics2D library

**Design Choices**

Number of the level panels depends on how many level the design team is going to do.

This class will extends JPanel and implements KeyListener.

**3.3 Input Management Subsystem**

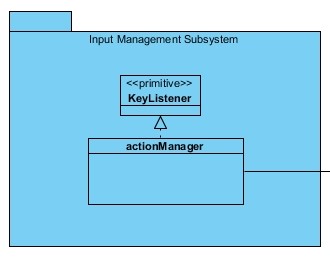


Figure 3.3: Input Management Subsystem

This part is still in progress. Require more research and more thinking. actionManager class aims to make a separate class for keylisteners during gameplay.

**3.4. File Management Subsystem**

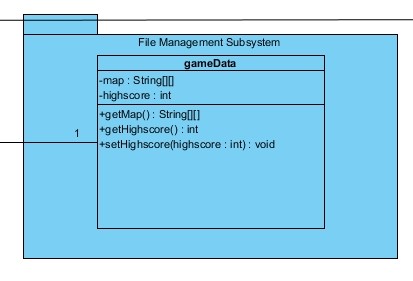


Figure 3.4: File Management Subsystem

3.4.1. gameData Class

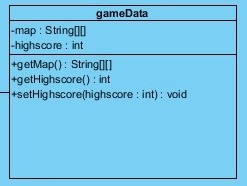


Figure 3.4.1: gameData Class

**Attributes**

Private 2D String Array map: This attribute will contain the map data which gameData will read from txt files.

**Private inthighscore:** This attribute will contain the highest score as integer which also be readed from a txt file.

**Constructors**

**Public gameData:** Take the level as input and load the appropriate level and high score.

**Methods**

**Public 2D String ArraygetMap():** Get map information, this method will be called under gameEngine classvia gameData object.

**Public intgetHighscore():** Get the highest score for that level, this method will be called under gameEngine class via gameData object.

**Public void setHighscore(inthighscore):** Set the highest score via overwriting the txt file, this method will be called under gameEngine class via gameData object.

**Design Choices**

There is no setMap method because of the fact that levels will be design before game execution and game algorithm will be programmed according to that data.

3.5. Game Algorithm / Game Visual Subsystem

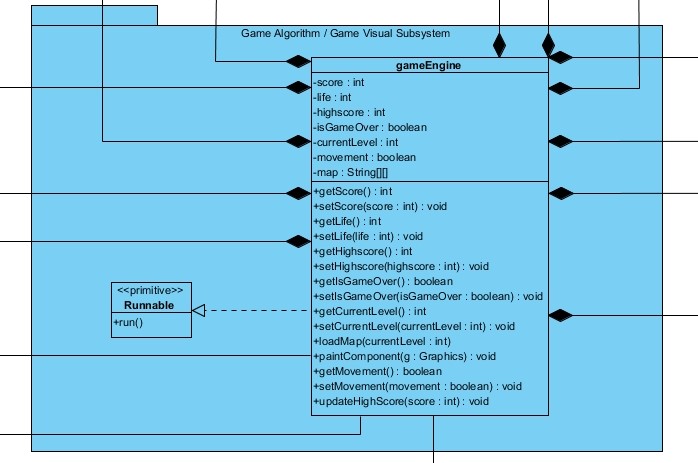


Figure 3.5: File Management Subsystem

3.5.1. gameEngine Class

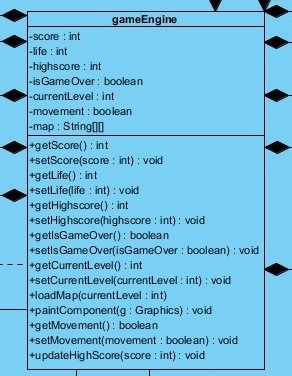


Figure 3.5.1: gameEngine Class

**Attributes**

**Private int score:** This attribute will contain the current score while game is running.

**Private int life:** This will indicate how many life points player has. Every level start with 3 and it will decrease every time player dies. When it reaches 0, game will end.

**Private inthighscore:** This attribute will contain the highest score of the current level. It will be loaded by gameData object which will be created during runtime. When isGameOver true, it will be updated depend on the score.

**Private Boolean isGameOver:** This attribute is created for future implementation purposes. It help us to implement methods and there could be a pause option during gameplay. Thus, it currently has no purpose other than making the implementation easy.

**Private intcurrentLevel:** This attribute will be selected by user under levelSelect class.

Private Boolean movement: This attribute is similar to isGameOver. To make the game different, there will be some power-ups in the game. One of them is going to decrease the speed of objects other than Jumpman so that player will be able to jump and kite easier. Basically it indicates the game speed.

**Private 2D String Array map:** This attribute will contain the map components and depends on the value inside gameEngine will draw the map. It will be loaded by gameData object which will be created during runtime.

**Constructors**

**Public gameEngine:** Take the level as input, draw the appropriate level and render it during runtime.

**Methods**

Public intgetScore: Get the score during run time.

Public void setScore(int score): Update the score during run time.

Public intgetLife(): To check the life during run time to finish the game.

Public void setLife(int life): There is another power-up and it increase your life count by 1.

Public intgetHighScore(): Load the highest score by using gameData object before game start executing.

Public Boolean getIsGameOver(): Determine the game is finished or not.

Public void setIsGameOver(Boolean isGameOver): Every time player dies, game algorithm will check if it is 0 or not then call this function with true.

Public intgetCurrentLevel(): Get the currentLevel to update highscore after game is finished.

Public void setCurrentLevel(intcurrentLevel): Set the currentLevel before execution of the game to reach it later.

Public void loadMap(intcurrentLevel): Will do the interaction with gameData class.

Public void paintComponent(Graphics g): It will draw the screen by using Java Graphics2D library and render it inside run() method.

Public Boolean getMovement(): Check whether or not player reached movement power-up.

Public void setMovement(Boolean movement): If player object and movement power-up collides, slow down the game.

Public void updateHighScore(int score):After game isGameOver is become true, game algorithm will check if the score is higher than highscore or not. Then if score is higher, this method will be called and next time the game executed, the new score will be shown.

Design Choices

This class will extends JPanel to draw the game screen and implements runnable to run the game by using run operation by overriding it.

Gameplay algorithms will be initiate under run() method.

We are planning to separate this subsystem into 2 pieces named as follows: Game Algorithm Subsystem and Game Visual Subsystem; however, we need to think over how to interact them.

* 1. Game Entities Subsystem

## 3.6.1 Game Screen Elements Subsystem

As the name of the subsystem designates this system holds objects of the game. There are several objects for our game including player, enemies, power-up and other objects which all inherits from main object class myObjects.

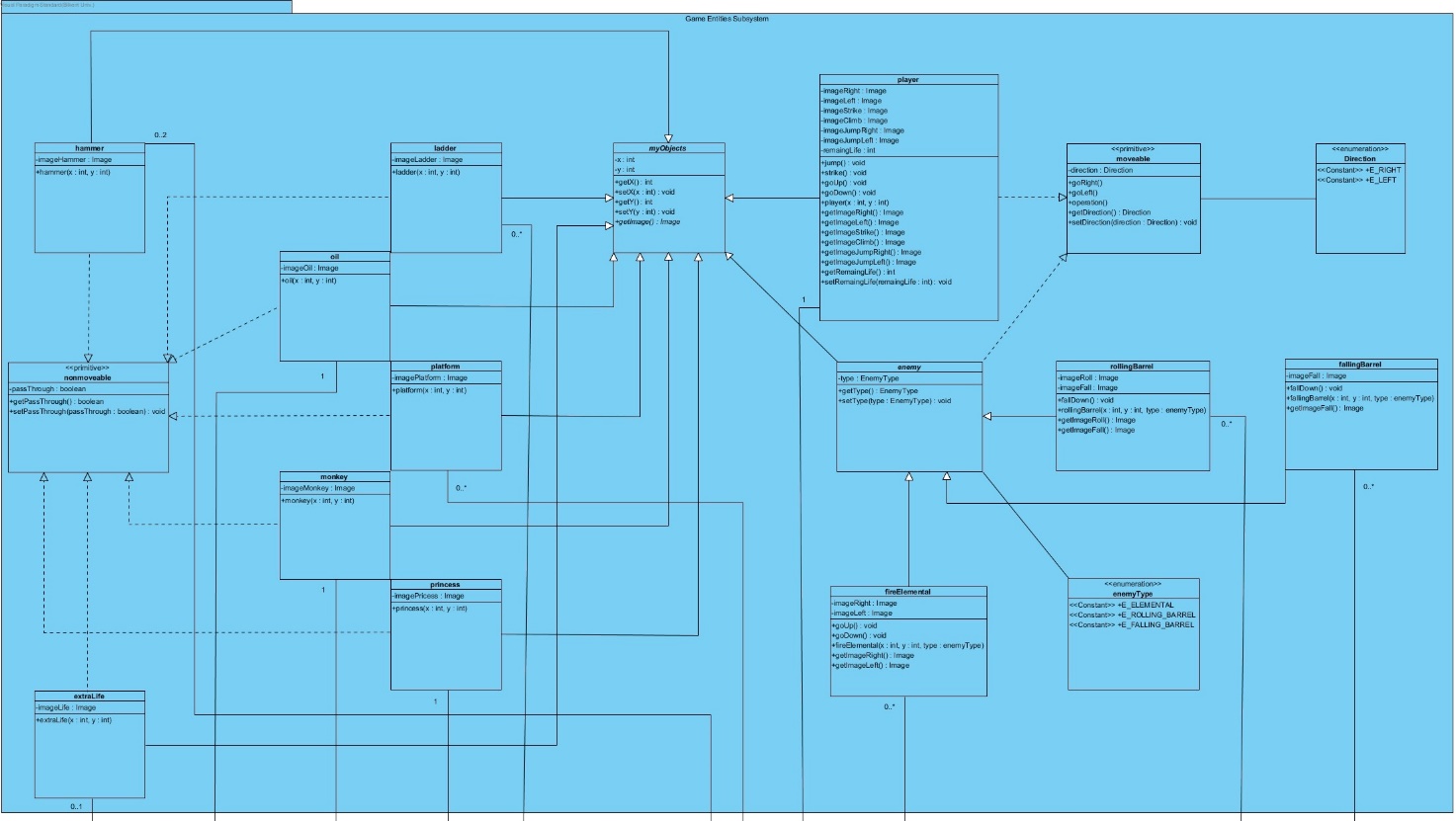
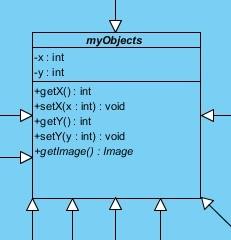


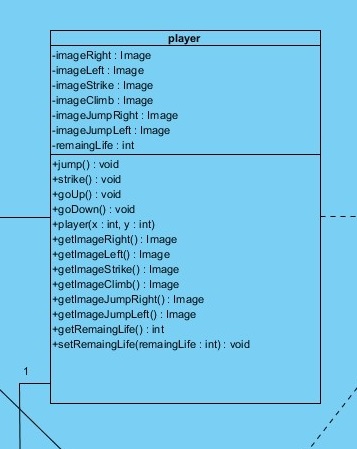
Figure 3.6.1.1 - Diagram of GameScreenElements

**myObject Class**

****

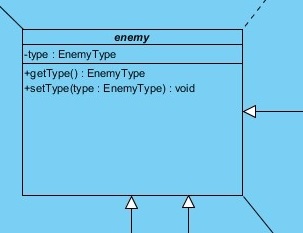
“myObject” class will be parent class of objects in the game, additionally this class will be instantiated when player starts the game. Also, “myObject” class have main operation including getting, setting position of another object and getting image of the needed object. Positions of the characters will be kept in the entities of these class.

**Player Class**

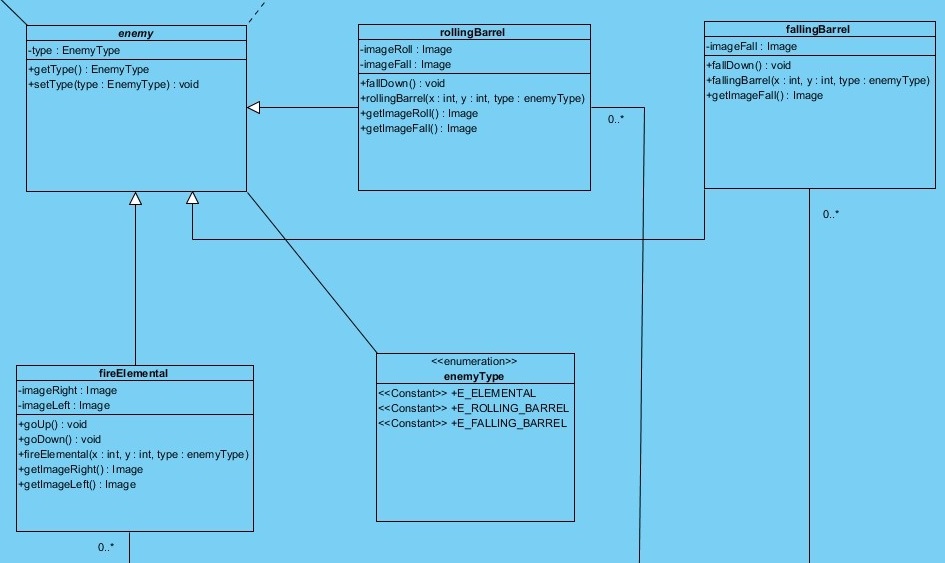
****

The “Player” class have some special operation and attributes including images of different states of the player, and special operations to use this images. Additionally, in this class operations like jump(), strike(), goUp(), goDown() which these operations are very important in player moves. Also, there are entities for keeping number of lives of player which these entites can be set and get by specific operations.

**Enemy Class**

****

Another important class is “enemy” class which all other enemy type classes inherits from this class. This class decides the type of the enemy like rollingBarrel, fallingBarrel, and fireElemental. These diagram demonstrates their relation with enemy class.



Every other class which inherits to the “enemy” class are rollingBarrel, fallingBarrel, and fireElemental, which all have different images for different states of the objects and consturctors that keep their positions and Enumeration type.

On the other hand, there are “nonmovable” objects like ladder, oil, platform, hammer, monkey, princess, extraLife which these objects will only keep images and constructors

1. **References**

[1] C. Horstmann, Big java. [Place of publication not identified]: John Wiley, 2016.

[2] T. Wright, *Fundamental 2D game programming with Java*, 4th ed. Boston, MA: Cengage Learning PTR, 2015.

[3] BRUEGGE, B., & DUTOIT, A. H. (2010). Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition. Prentice-Hall.