Bilkent University

Department of Computer Engineering

CS319 – Object Oriented Software Engineering Project

Project short-name: Donkey Kong Game

Design Report (Final Draft)

Group 3E
Fuad Ahmadov
Çağatay Küpeli
Sine Mete
Arkın Yılmaz

Supervisor: Bora Güngören

Analysis Report (Final Draft)
Dec 1, 2017

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS319/3.

Contents

Design Report (Final Draft)

Project short-name: Donkey Kong Game

1. Introduction

1.1 Purpose of the System

Donkey Kong Game is a re-mastered version of popular arcade game, Donkey Kong 1981. In this game, user try to reach finishing point without hitting any hostile obstacles. It is one of the well-designed game in its time, so it is user-friendly and easy to understand. Even though Donkey Kong 1981 was a successful game, it was lacking some features. In order to add them, the design and gameplay of the game is altered.

In addition, the original levels in Donkey Kong 1981 will not be remade for this system, instead levels will be redesigned. This allow us to show the new features of the game and make it much easier. This design choice allow us to make levels that shows what are the features added into the game.

1.2 Design Goals

Usability: The design of Donkey Kong 1981 inspired us in terms of usability requirement. A person who never played Donkey Kong 1981, can play this game without any hesitation because every arcade game use the similar or same patterns for user inputs. We created a list of characteristics the user controls share in software.

- Control inputs should make sense. For example you cannot use UP button one keyboard to go right or left.
- Control inputs should be close to each other in order to create better gameplay experience.
- Control inputs should be easy to understand.
- There should be always a help option to explain core concepts of the game such as control inputs.

Performance: Performance is important design goal for games. You cannot expect people to enjoy the game if it has some optimization problems such as sudden FPS drops and freezes. In other words, the game should run at high FPS and should preserve it.

Portability: Portability is crucial for any software. In order to make Donkey Kong Game portable, Donkey Kong Game will be developed with Java because Java is one of the few programming languages which allows cross-platform portability. This attribute of Java allows Donkey Kong Game to work any environment which installed JRE; therefore user will not worry about operating system requirements.

Reliability: Reliability is another important design goal for games. The game system should be bug-free and it should not crash due to unexpected reasons. Reliability of system

will be tested through the development of the system in order to not have any bugs and crashes at execution time.

Extendibility: Extendibility is a must for our system because as it was explained 1.1 our purpose for this system is to improve Donkey Kong 1981. Thus, the system should be coded in a way that it should be simple to add new components and features to the system.

- New levels should can be added just by creating a txt file with appropriate name.
- New power-ups and enchantments should can be added without changing old part of the code.

Trade-Offs

Reusability vs. Performance: Reusability is important for designs that might see some future usage in other projects. However we have no plan of making another arcade game. Therefore we are not planning to make our code usable unless we might decide to use the functionality inside another class.

Memory vs. Speed: Even though memory usage is important in order to make the game fast as possible, we are not concern about memory space.

Every objects which can be seen on the game panel will be created separately in order to make collision detection faster. This design choice will cost us so much memory; however, it will help us to detect collisions faster.

1.3 Definition, Acronyms & Abbreviations

Frame Per Second (FPS): A measurement for how many unique consecutive images occur each second.

Arcade Game: A type of game genre that is a fast-paced action game, requiring hand-eye coordination skill to play.

Cross-Platform: Software that can run on multiple types of Operating Systems.

Java Runtime Environment (JRE): A set of software tools for development of Java applications.

2. Software Architecture

2.1.1 Subsystem Decomposition

We chose the MVC architectural style for our gameplay (in game) design because it is convenient way to integrate our gameplay design. The MVC style consists of three layers as it is designated in its name, Model, View, and Controller.

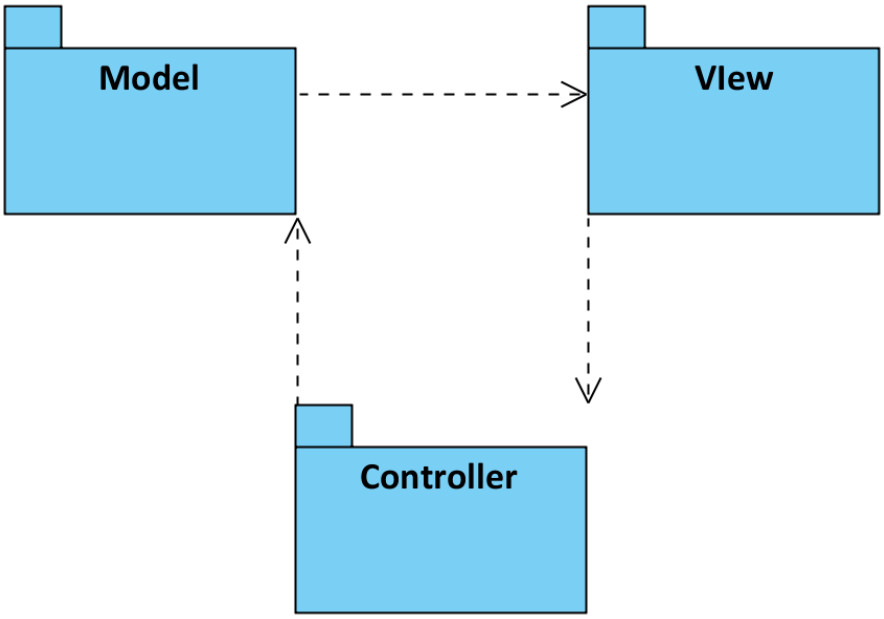

Figure 2.1.1: High-level implementation of Subsystem Decomposition

Firstly, controller layer accepts inputs from the user and commands model (sometimes view) to execute according to inputs. Mainly, controller sends updates to model, which by this way model updates model's state. Secondly, model which is also central and most important component of the layer. Additionally, model gets data from controller and sends to view which layer view displays. Additionally, View-Controller is optional for different systems.

In Section 3.1, class diagram is placed which it is separated to 3 parts according to MVC architectural style. As it is seen from 3 layers, every layer has main classes for each layer. Main class of the model part is "GameEngine.java" class which object classes are connected to "GameEngine.java". Besides, for controller layer "Controller.java" class is main class for controller layer. "Controller.java", "MapData.java", and "ScoreData.java" classes have relationship with "GameEngine.java" which demonstrates relationship between model layer ("GameEngine.java") and Controller layer ("Controller.java", "MapData.java", "ScoreData.java"). Finally, after these relationship between View layer displays the program (game).

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. System Model

3.1 Design Patterns

Design Pattern	Instances		
Façade	Class	Located in	Provides access to
	Controller	Game Management	GameEngine & GamePanel

3.2 Detailed Class Diagram

The full class diagram is attached to the next page in order to provide a better look on interactions between classes and subsystems.

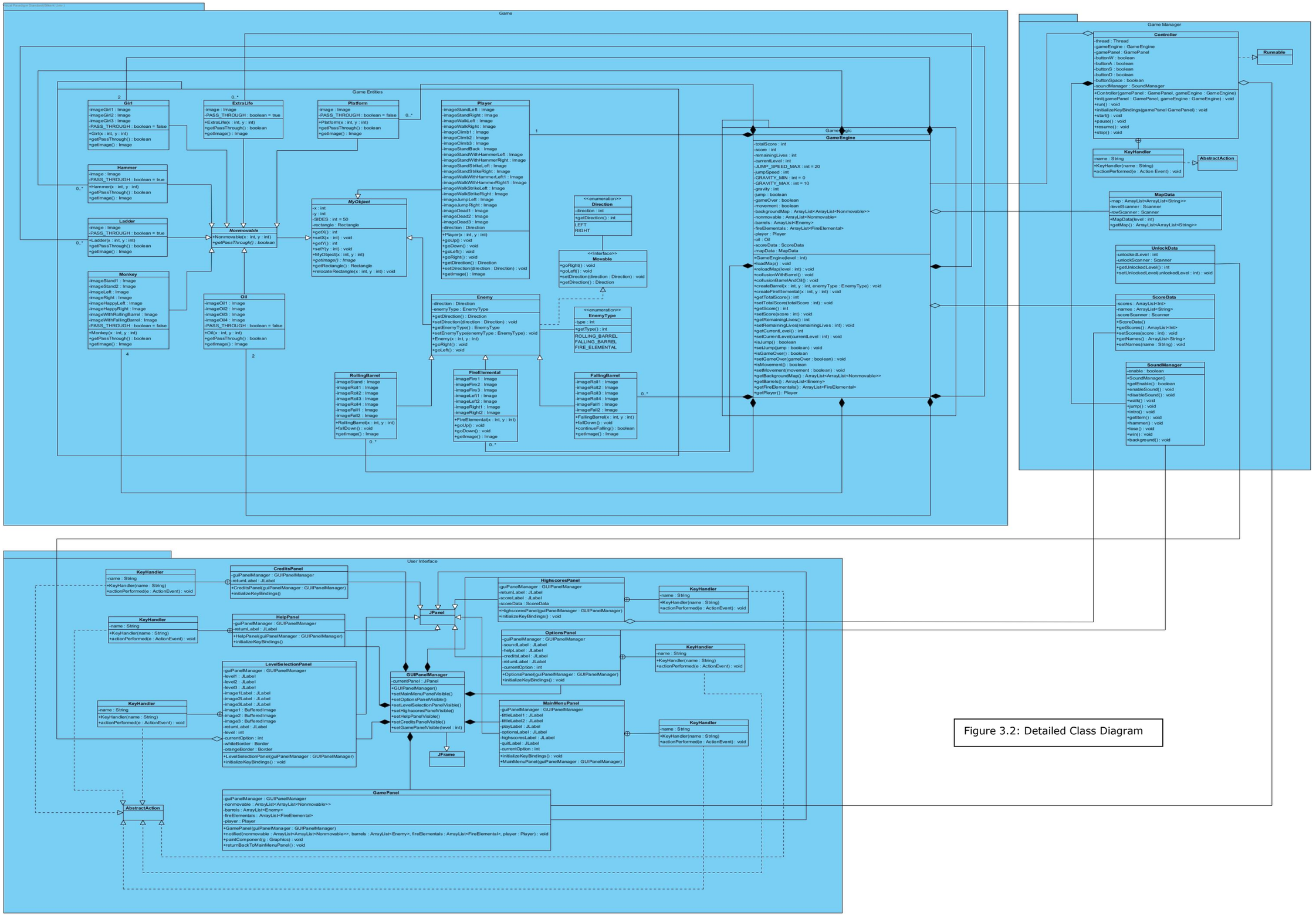

Figure 3.2: Detailed Class Diagram

3.3 Subsystems

3.3.1 User Interface Subsystem

The User Interface Subsystem class diagram is attached to the next page in order to provide a better look on interactions between classes.

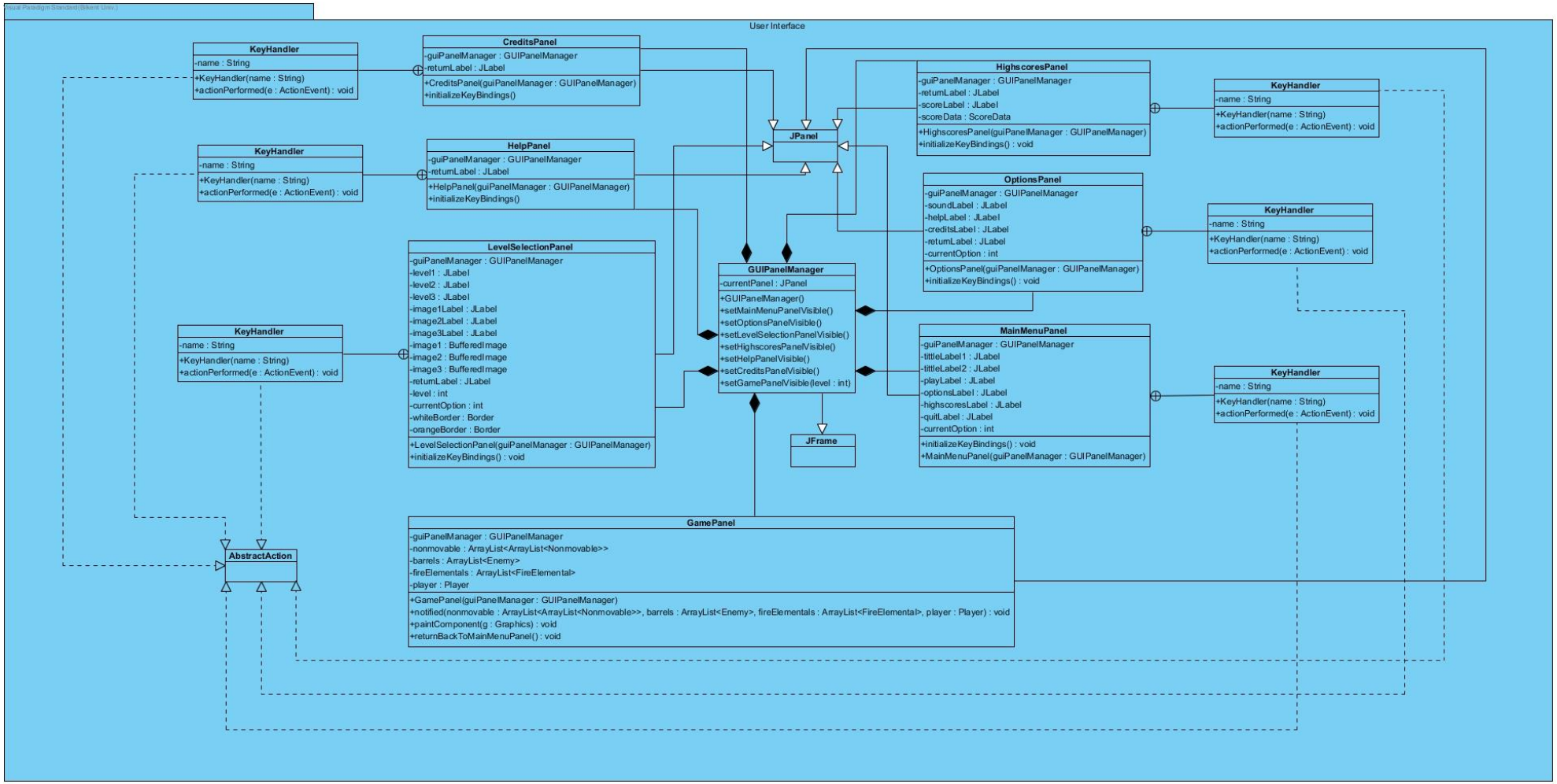

Figure 3.3.1: User Interface Subsystem

3.3.1.1 GUIPanelManager Class

GUIPanelManager class creates the main frame of the game and it is also handles which panel will be shown to user. Other classes in User Interface Subsystem cannot interact directly, instead they call GUIPanelManager class to make other class visible to user.

- GUIPanelManager extends JFrame

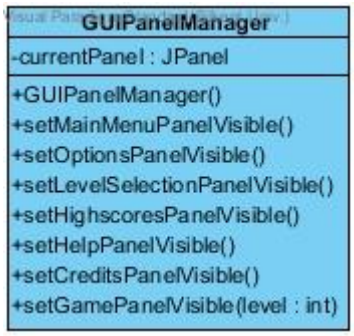

Figure 3.3.1.1: GUIPanelManager Class

Attributes

private	currentPanel	JPanel	Keeps the current panel that is shown to the user

Constructors

public	GUIPanelManager()	Creates GUIPanelManager object

Methods

public	setMainMenuPanelVisible()	void	Changes the current panel to MainMenuPanel
public	setOptionsPanelVisible()	void	Changes the current panel to OptionsPanel
public	setLevelSelectionPanelVisible()	void	Changes the current panel to LevelSelectionPanel
public	setHighscoresPanelVisible()	void	Changes the current panel to HighscoresPanel
public	setHelpPanelVisible()	void	Changes the current panel to HelpPanel
public	setCreditsPanelVisible()	void	Changes the current panel to CreditsPanel
public	setGamePanelVisible(level : int)	void	Changes the current panel to GamePanel with respected level name

3.3.1.2 KeyHandler Class

KeyHandler class is a nested class that create an interaction between keyboard inputs from user and classes. Except for KeyHandler nested class inside Controller class which interact with GamePanel class and GameEngine class, every other class create its nested class with the same name. The rasoning behind this was it was easy to manage during implementation and easy to modify.

- KeyHandler implements AbstractAction

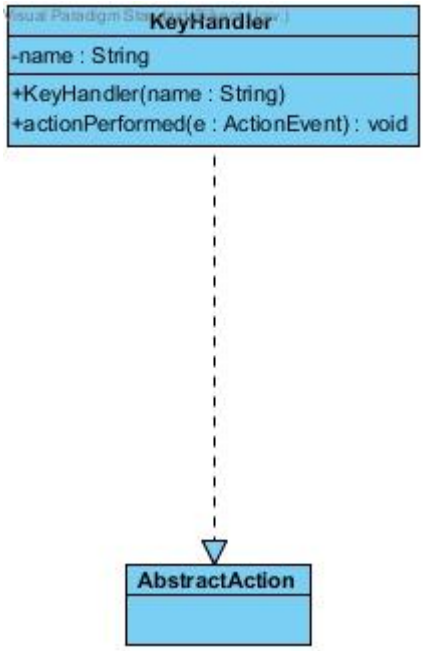

Figure 3.3.1.2: KeyHandler Class

Attributes

private	name	String	Keeps the name of the input from keyboard

Constructors

public	KeyHandler(name : String)	Creates a KeyHandler object that convert user inputs from user into a meaning

Methods

public	actionPermormed(e : ActionEvent)	void	Perform appropriate action according to user input name

3.3.1.3 MainMenuPanel Class

MainMenuPanel class creates a main menu panel and its components in order to display in GUIPanelManager class. All of the attributes in this class is for user interface design.

- MainMenuPanel class extends JPanel

- It has a nested class called KeyHandler class which allow MainMenuPanel class to take inputs from user. Reason for this design choice was KeyHandler class was easy to use and many of the classes take same keyboard inputs for different reasons.

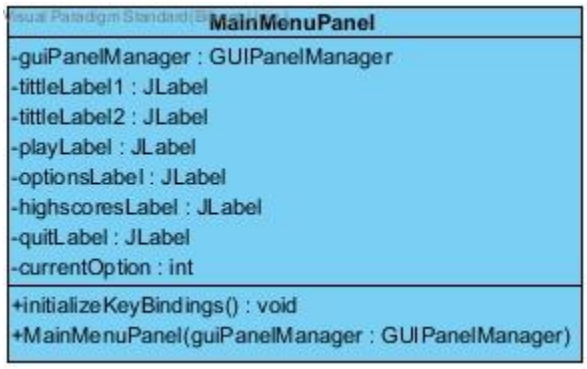

Figure 3.3.1.3: MainMenuPanel Class

- During implementation, we decided to use JLabel's for buttons due to the fact that we are using KeyBindings as a keyboard listener.

Attributes

private	guiPanelManager	GUIPanelManager	MainMenuPanel calls setVisible() methods in GUIPanelManager via this object
private	tittleLabel1	JLabel	1st part of the tittle, it says "Donkey"
private	tittleLabel2	JLabel	2nd part of the tittle, it says "Kong"
private	playLabel	JLabel	Label for play button
private	optionsLabel	JLabel	Label for options button
private	highscoresLabel	JLabel	Label for highscores button
private	quitLabel	JLabel	Label for quit button
private	currentOption	int	Work as an indicator for button labels

Constructors

public	MainMenuPanel(guiPanelManager : GUIPanelManager)	Creates a MainMenuPanel object that organize how main menu should look

Methods

public	initializeKeyBindings()	void	Create the interaction between JPanel and Keyboard inputs and focuses the current panel

3.3.1.4 HighscoresPanel Class

HighscoresPanel class creates a highscore panel and its components in order to display in GUIPanelManager. All of the attributes in this class is for user interface design.

- HighscoresPanel extends JPanel

- It has a nested class called KeyHandler which allow HighScoresPanel class to take inputs from user.

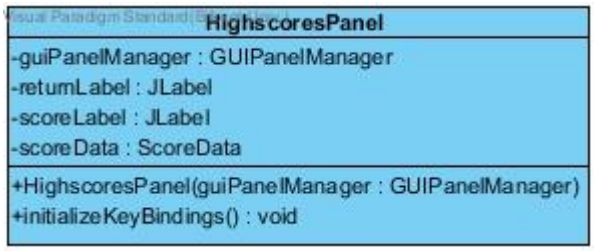

Figure 3.3.1.4: HighscoresPanel Class

- During implementation, we decided to use JLabel's for buttons due to the fact that we are using KeyBindings as a keyboard listener.

Attributes

private	guiPanelManager	GUIPanelManager	HighscoresPanel calls setVisible() methods in GUIPanelManager via this object
private	returnLabel	JLabel	Label for return button
private	scoreLabel	JLabel	Scores taken from ScoreData object will be displayed inside this label
private	scoreData	ScoreData	Provide a highscore table for HighscoresPanel class

Constructors

public	HighscoresPanel(guiPanelManager : GUIPanelManager)	Creates a HighscoresPanel object that organize how highscores should look

Methods

public	initializeKeyBindings()	void	Create the interaction between JPanel and Keyboard inputs and focuses the current panel

3.3.1.5 LevelSelectionPanel Class

LevelSelectionPanel class creates a level selection panel and its components in order to display in GUIPanelManager. It also interact with UnlockData class in GameManager to show which levels are open to user. All of the attributes in this class is for user interface design.

- LevelSelectionPanel extends JPanel
- It has a nested class called KeyHandler which allow LevelSelectionPanel class to take inputs from user.

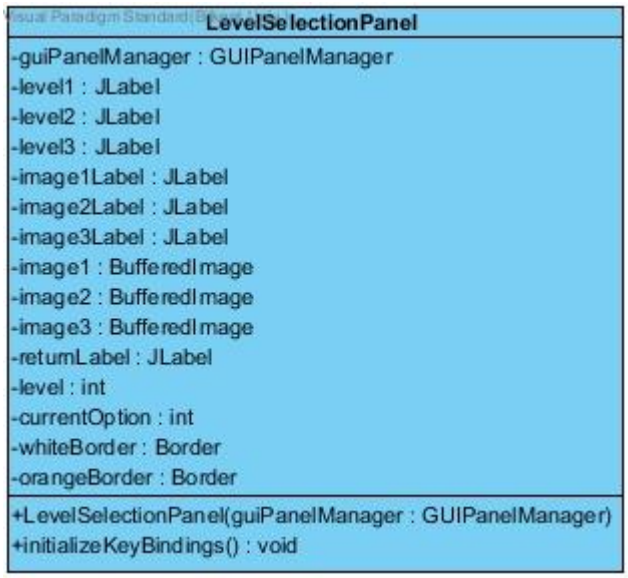

Figure 3.3.1.5: LevelSelectionPanel Class

- During implementation, we decided to use JLabel's for buttons due to the fact that we are using KeyBindings as a keyboard listener.

Attributes

private	guiPanelManager	GUIPanelManager	LevelSelectionPanel calls setVisible() methods in GUIPanelManager via this object
private	level1	JLabel	Label for level 1 button
private	level2	JLabel	Label for level 2 button
private	level3	JLabel	Label for level 3 button
private	imageLevel1	JLabel	Level 1 image will be displayed inside this label
private	imageLevel2	JLabel	Level 1 image will be displayed inside this label
private	imageLevel3	JLabel	Level 1 image will be displayed inside this label
private	image1	BufferedImage	Keeps the image of level1 in order to display inside imageLevel1
private	image2	BufferedImage	Keeps the image of level2 in order to display inside imageLevel2
private	image3	BufferedImage	Keeps the image of level3 in order to display inside imageLevel3
private	returnLabel	JLabel	Label for return button
private	level	int	Keeps the selected level number in order to call setGamePanelVisible(level : int) with it when user presses appropriated keyboard input which

			initialize inside initializeKeyBindings() method
private	currentOption	int	Work as an indicator for button labels
private	whiteBorder	Border	Visualized version of currentOption attribute. If an image has a white border, this means it is not selected.
private	orangeBorder	Border	Visualized version of currentOption attribute. If an image has an orange border, this means it is selected currently.

Constructors

public	LevelSelectionPanel(guiPanelManager : GUIPanelManager)	Creates a LevelSelectionPanel object that organize how level selection should look

Methods

public	initializeKeyBindings()	void	Create the interaction between JPanel and Keyboard inputs and focuses the current panel

3.3.1.6 OptionsPanel Class

OptionsPanel class creates an option panel and its components in order to display in GUIPanelManager. All of the attributes in this class is for user interface design.

- OptionsPanel extends JPanel
- It has a nested class called KeyHandler which allow OptionsPanel class to take inputs from user.

Figure 3.3.1.6: OptionsPanel Class

- During implementation, we decided to use JLabel's for buttons due to the fact that we are using KeyBindings as a keyboard listener.

Attributes

private	guiPanelManager	GUIPanelManager	OptionsPanel calls setVisible() methods in GUIPanelManager via this object

private	soundLabel	JLabel	Label for turn on/off sound button
private	helpLabel	JLabel	Label for help button
private	creditsLabel	JLabel	Label for credits button
private	returnLabel	JLabel	Level for return button
private	currentOption	int	Work as an indicator for button labels

Constructors

public	OptionsPanel(guiPanelManager : GUIPanelManager)	Creates a OptionsPanel object that organize how options panel should look

Methods

public	initializeKeyBindings()	void	Create the interaction between JPanel and Keyboard inputs and focuses the current panel

3.3.1.7 HelpPanel Class

HelpPanel class creates an option panel and its components in order to display in GUIPanelManager. All of the attributes in this class is for user interface design.

- HelpPanel extends JPanel
- It has a nested class called KeyHandler which allow HelpPanel class to take inputs from user.

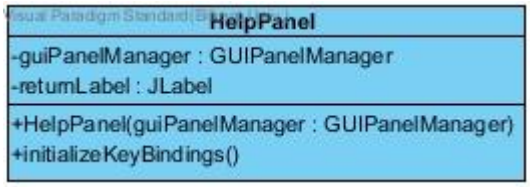

Figure 3.3.1.7: HelpPanel Class

- During implementation, we decided to use JLabel's for buttons due to the fact that we are using KeyBindings as a keyboard listener.

Attributes

private	guiPanelManager	GUIPanelManager	HelpPanel calls setVisible() methods in GUIPanelManager via this object
private	returnLabel	JLabel	Level for return button

Constructors

public	OptionsPanel(guiPanelManager : GUIPanelManager)	Creates a OptionsPanel object that organize how options panel should look

Methods

public	initializeKeyBindings()	void	Create the interaction between JPanel and Keyboard inputs and focuses the current panel

3.3.1.8 CreditsClass Class

CreditsPanel class creates an option panel and its components in order to display in GUIPanelManager. All of the attributes in this class is for user interface design.

- CreditsPanel extends JPanel
- It has a nested class called KeyHandler which allow CreditsPanel class to take inputs from user.

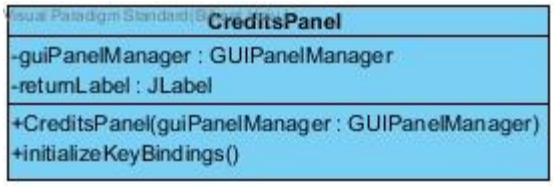

Figure 3.3.1.8: CreditsPanel Class

- During implementation, we decided to use JLabel's for buttons due to the fact that we are using KeyBindings as a keyboard listener.

Attributes

private	guiPanelManager	GUIPanelManager	CreditsPanel calls setVisible() methods in GUIPanelManager via this object
private	returnLabel	JLabel	Level for return button

Constructors

public	CreditsPanel (guiPanelManager : GUIPanelManager)	Creates a CreditsPanel object that organize how options panel should look

Methods

public	initializeKeyBindings()	void	Create the interaction between JPanel and Keyboard inputs and focuses the current panel

3.3.1.9 GamePanel Class

GamePanel class display the game state with respect to data sent by Controller class inside GameManager Subsystem. This class does not interact with user unlike other classes inside User Interface Subsystem because GamePanel has no knowledge about game state. User inputs will

interpret inside GameEngine class and GamePanel class takes the state of the game via Controller class to display it on the screen.

- CreditsPanel extends JPanel

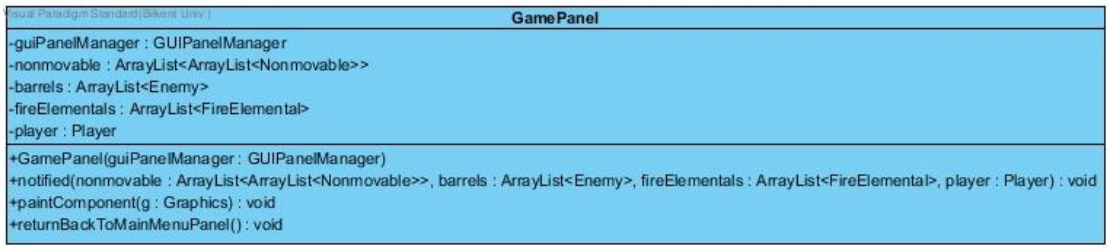

Figure 3.3.1.9: GamePanel Class

Attributes

private	guiPanelManager	GUIPanelManager	GamePanel calls setVisible() methods in GUIPanelManager via this object
private	nonmovable	ArrayList<ArrayList<Nonmovable>>	Keeps Nonmovable object such as Girl, Monkey and Ladder, etc. in order to use in paintComponent(g: Graphics) method.
private	barrels	ArrayList<Enemy>	Keeps Barrel objects in order to use in paintComponent(g: Graphics) method.
private	fireElementals	ArrayList<FireElemental>	Keeps FireElemental objects in order to use in paintComponent(g: Graphics) method.
private	player	Player	Keeps Player objects in order to use in paintComponent(g: Graphics) method.

- Nonmovable, barrels, fireElementals and player has a instance inside this class in order to reach them paintComponent(g : Graphics) method easily. Otherwise there is no reason to keep them as objects.

Constructors

public	GamePanel (guiPanelManager : GUIPanelManager)	Creates a GamePanel object that renders game state on screen

Methods

public	initializeKeyBindings()	void	Create the interaction between JPanel and Keyboard inputs and focuses the current panel
public	notified(nonmovable : ArrayList<ArrayList<Nonmovable>>, barrels : ArrayList<Enemy>, fireElementals : ArrayList<FireElementals>, player : Player)	void	Controller class uses this method in order to notify GamePanel. This method set old attributes with new ones and calls repaint() method to re-render the screen
public	returnBackToMainMenuPanel()	void	Create the interaction between JPanel and Keyboard inputs and focuses the current panel
public	paintComponent(g : Graphics)*	void	Organizes map objects according to their X and Y coordinates and display them.

- (*) paintComponent(g : Graphics) is not called directly. GamePanel uses repaint() method to call paintComponent(g: Graphics) method to render the screen.

3.3.2 Game Manager Subsystem

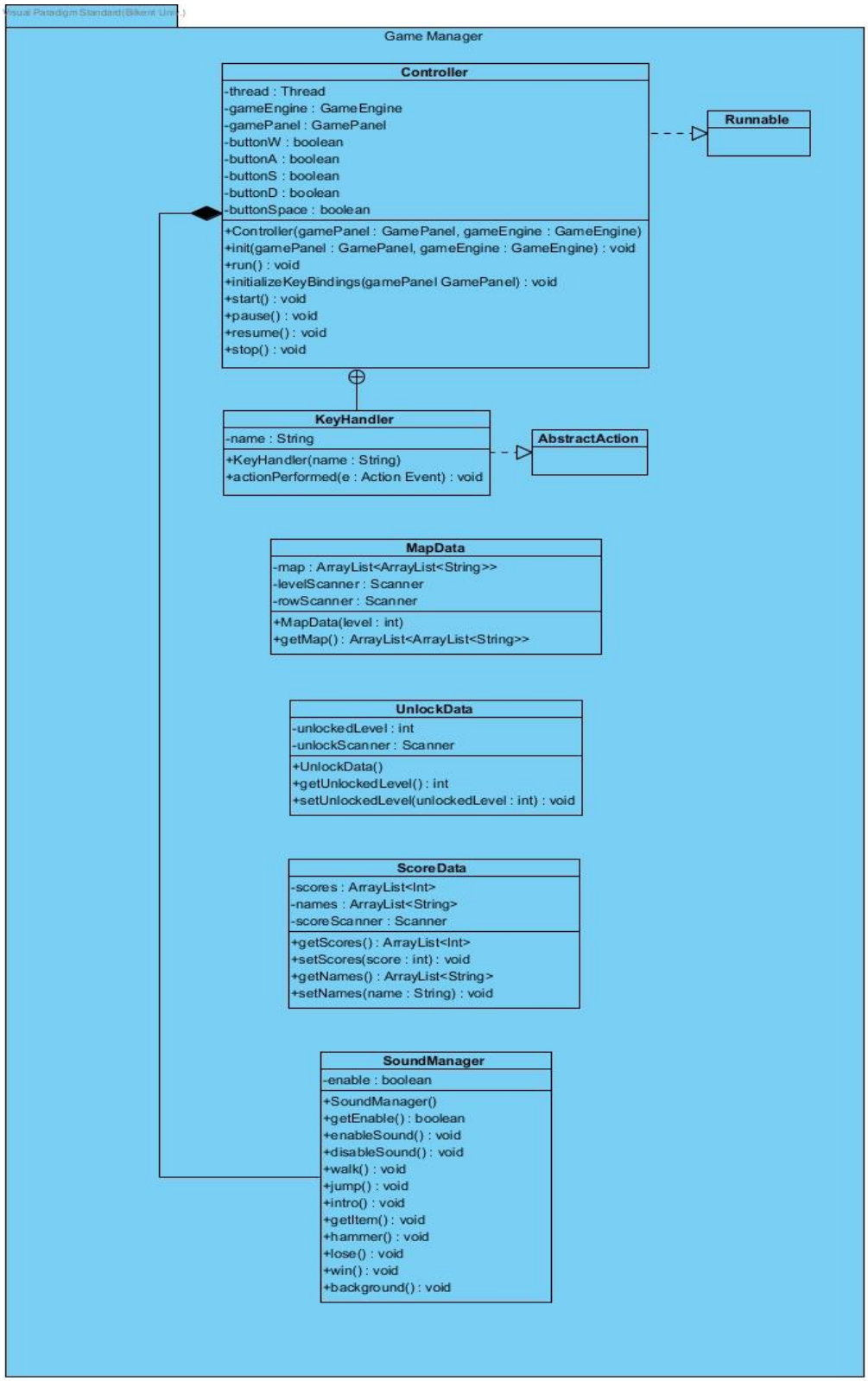

Figure 3.3.2: Game Manager Subsystem

3.3.2.1 SoundManager Class

SoundManager class make sounds via methods such as walk() or jump(). Controller class commonly uses SoundManager to play sounds during game play. There is no logic inside this class, it play the sound according to which method is called. In other words, Controller class manages which sound will be played via SoundManager's methods. Also OptionPanel uses the same SoundManager object to enable or disable the sound.

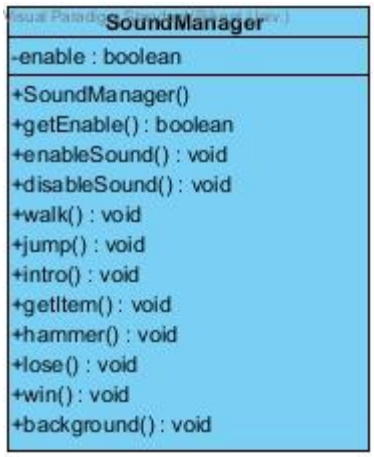

Figure 3.3.2.1: SoundManager Class

Attributes

private	enable	boolean	Determines whether SoundManager make sound. If enable is true, SoundManager can work normally. However if enable is false, even though you call a method, SoundManager will not make sounds.

Constructors

public	SoundManager()	Creates a SoundManager object that make sounds according to which of its method is called

Methods

public	getEnable()	boolean	Returns the value of enable.
public	enableSound()	void	Basically a setter for enable, OptionPanel uses this method to turn on the sound.
public	disableSound()	void	Basically a setter for enable, OptionPanel uses this method to turn off the sound.
public	walk()	void	Make walking sound for Controller class to use it during game

public	jump()	void	Make jumping sound for Controller class to use it during game
public	intro()	void	Make intro sound for Controller class to use it during game
public	getItem()	void	Make getting item sound such as getting a coin or extra life for Controller class to use it during game
public	hammer()	void	Make hammer strike sound for Controller class to use it during game
public	lose()	void	Make lose sound for Controller class to use it during game
public	win()	void	Make win sound for Controller class to use it during game
public	background()	void	Make background sound for Controller class to use it during game

3.3.2.2 ScoreData Class

ScoreData class stores the highscore datas inside a text file in order to preserve it for following execution of the game. The highscore datas should be stored for next execution of the game. Otherwise every execution, highscores will be lost. Therefore ScoreData class keeps highscores inside a text file.

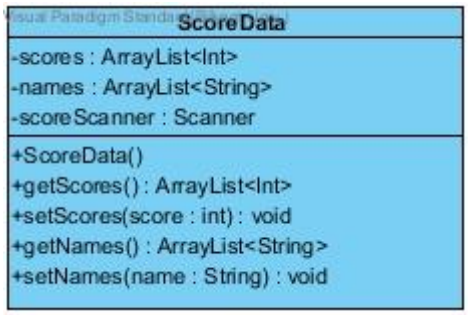

Figure 3.3.2.2: ScoreData Class

Attributes

private	scores	ArrayList<int>	An ArrayList to store scores of players with respect to their index. HighScorePanel class uses this ArrayList to reach highscores.
private	names	ArrayList<String>	An ArrayList to store names of players with respect to their index. HighScorePanel class uses this ArrayList to reach the names of players.

private	scoreScanner	Scanner	Scanner for score text file

Constructors

public	ScoreData()	Creates a ScoreData object that stores the highscores.

Methods

public	getScores()	ArrayList<int>	Returns the value of scores for other classes
public	setScores(score: int)	void	When a highscore detected, setScores(score: int) is used to store the new highscore.
public	getNames()	ArrayList<String>	Returns the value of names for other classes
public	setNames(name : String)	void	When a highscore detected, setNames(score: int) is used to store the name of the player.

3.3.2.3 UnlockData Class

UnlockData class stores the unlocked levels inside a text file in order to preserve it for following execution of the game. Purpose of this class is same as ScoreData, which is preserve the progress of user. Every execution of the mage, user should not be force to play all the levels to unlock them.

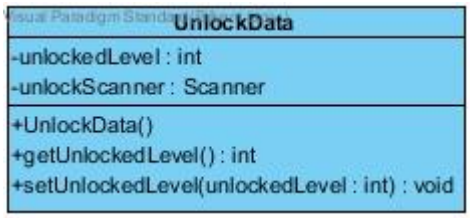

Figure 3.3.2.3: UnlockData Class

Attributes

private	unlockedLevel	int	An int to store unlocked level. LevelSelectionPanel class uses this int to display user which levels he/she can play.
private	unlockScanner	Scanner	Scanner for unlockLevel text file

Constructors

public	UnlockData()	Creates a UnlockData object that stores the index of last level unlocked

Methods

public	getUnlockedLevel()	int	Returns the the index of last level unlocked
public	setUnlockedLevel(unlockedLevel : int)	void	When user successfully finish a level, this method is called to save his/her progress.

3.3.2.4 MapData Class

MapData class provides game maps for GameEngine class. Similarly levels also stored inside text files. Unlike ScoreData or MapData, no one can change the starting state of the map.

Figure 3.3.2.4: MapData Class

Attributes

private	map	ArrayList<ArrayList<String>>	An 2D ArrayList to store starting state of the game for GameEngine class.
private	levelScanner	Scanner	Scanner for level text file
private	rowScanner	Scanner	Scanner for level text file

- The reason we are using 2 Scanner is that levels are stored as matrixes inside text file. One is move around rows and other one is cols.

Constructors

public	MapData(level : int)	Creates a MapData objects according to the level input. For example MapData(1) provides level 1 map and MapData(2) provides level 2 map

Methods

public	getMap()	ArrayList<ArrayList<String>>	Returns the 2D ArrayList which stores starting state of the game. GameEngine class calls this method.

3.3.2.5 Controller Class

Controller class creates the connection between GameEngine class and GamePanel class.

- Controller implements Runnable
- Controller class uses a thread to check whether or not user enters a keyboard input.
- Controller class decides which sound will be played according to inputs from GameEngine class.

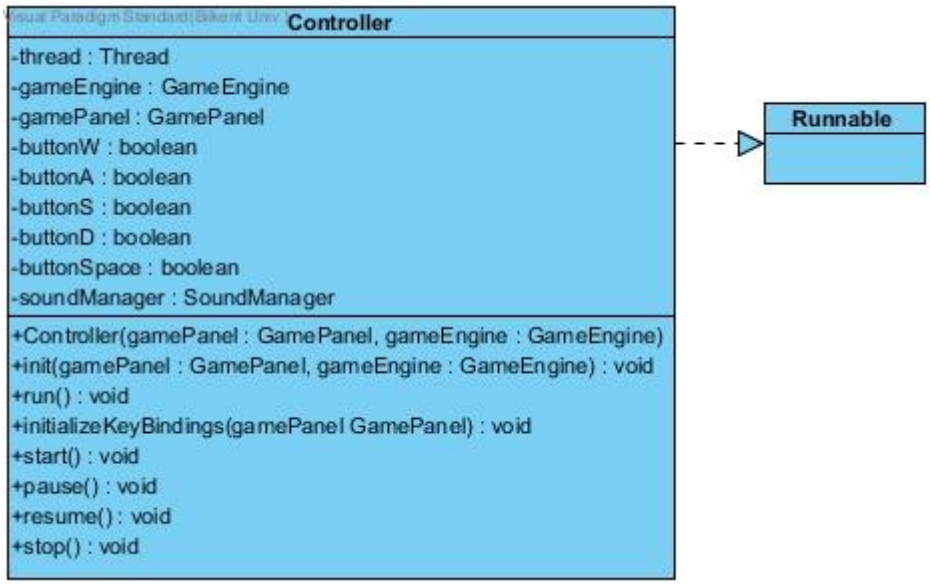

Figure 3.3.2.5: Controller Class

Attributes

private	thread	Thread	A thread to run the game.
private	gameEngine	GameEngine	GameEngine object which provide game states and game algorithms
private	gamePanel	GamePanel	GamePanel object which render the game state taken from GameEngine object. To render the screen Controller calls notify method inside GamePanel
private	buttonW	boolean	Indicate if W button pressed.
private	buttonA	boolean	Indicate if A button pressed.
private	buttonS	boolean	Indicate if S button pressed.
private	buttonD	boolean	Indicate if D button pressed.
private	buttonSpace	boolean	Indicate if SPACE button pressed.
private	soundManager	SoundManager	Controller makes sound by using this SoundManager object.

- We have 5 different Booleans for KeyBindings. The reasoning behind this is that to detect 2 KeyBinding at the same time. If we uses actionPermormed(e : ActionEvent) to call events in GameEngine, it only calls the 1st button pressed.

Constructors

public	Controller(gamePanel : GamePanel, gameEngine : GameEngine)	Creates a Controller establish the connection between GamePanel and GameEngine.

Methods

public	init()	void	Initialization method for Controller
public	run()	void	Override method from Runnable. When we start the thread, it automatically calls run() method.
public	initializeKeyBindings(gamePanel : GamePanel)	void	Create the interaction between GamePanel and Keyboard inputs and focuses GamePanel.
public	start()	void	Initialize a new thread and starts it
public	pause()	void	Pause the game by calling setMovement(False)
public	resume()	void	Resume the game by calling setMovement(True)
public	stop()	void	Stops the thread

- run() is a special method because it has a while loop inside that loops until game is over. It constantly ask for a new game state form GameEngine and notify GamePanel to display that game state.

3.3.3 Game Subsystem

The Game Subsystem class diagram is attached to the next page in order to provide a better look on interactions between classes.

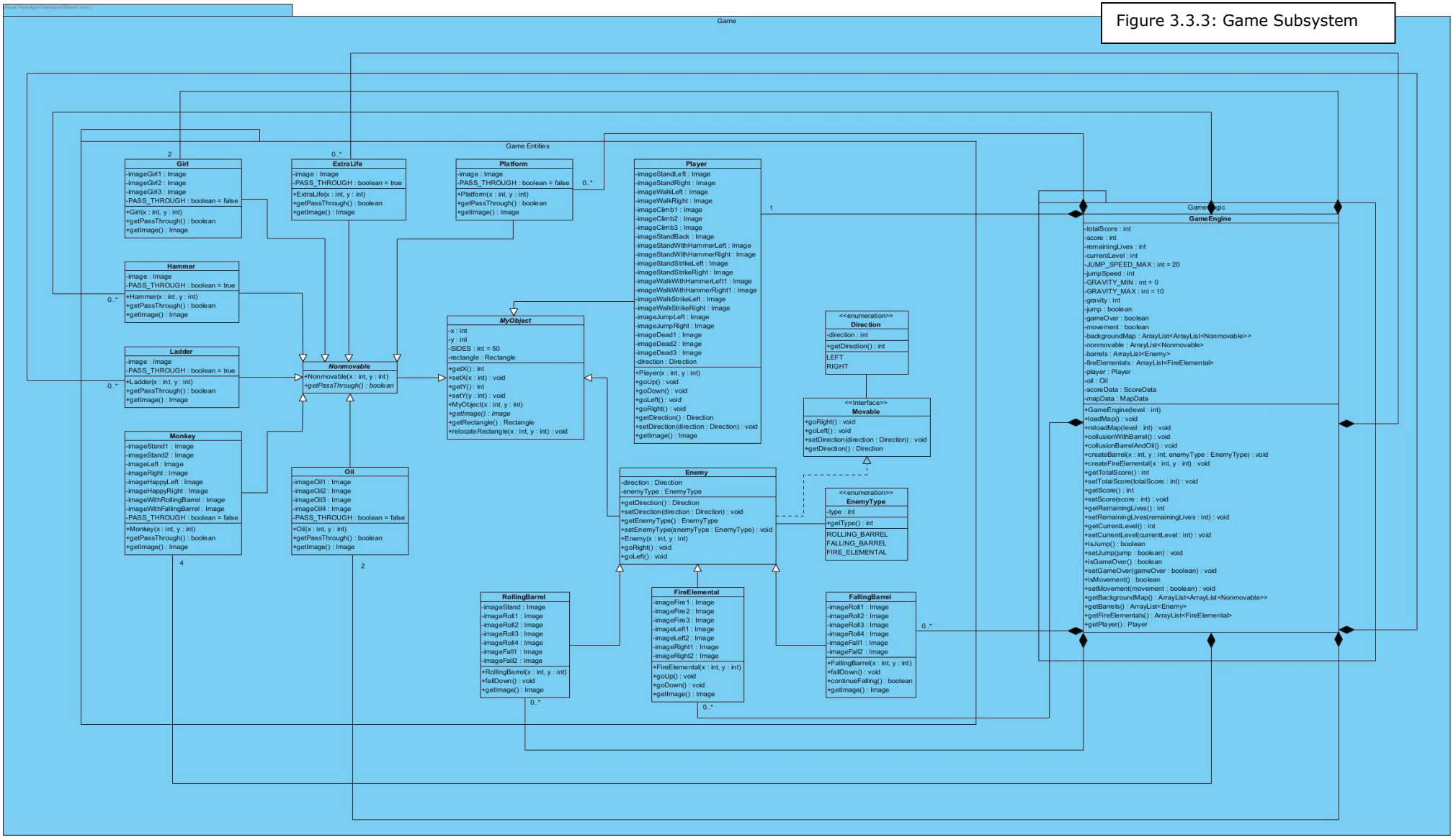

Figure 3.3.3: Game Subsystem

3.3.3.1 MyObject Abstract Class

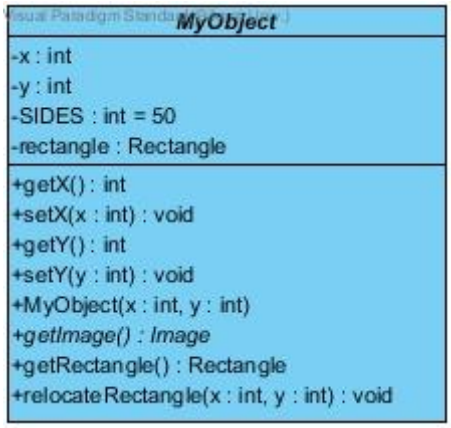

Figure 3.3.3.1: MyObject Abtract Class

Attributes

private	x	int	X-coordinate of the object
private	y	int	Y-coordinate of the object
private	SIDES = 50	int	Images are rectangles and each side is 50 pixel
private	rectangle	Rectangle	Help us to detect collisions

Constructors

public	MyObject(x : int, y : int)	Child classes inherit the implementation

Methods

public	getX()	int	Returns x value
public	getY()	int	Returns y value
public	setX(x : int)	void	Sets x value to parameter
public	setY(y : int)	void	Sets y value to parameter
public	getImage()	Image	Abstract method
public	getRectangle()	Rectangle	Returns rectangle
public	relocateRectangle(x : int, y : int)	void	Change the location of rectangle

3.3.3.2 ExtraLife, Girl, Hammer, Ladder, Monkey, Nonmovable, Platform, Oil Classes

Nonmovable is the parent class of all other classes which are listed in the tittle. ExtraLife, Girl, Hammer, Ladder, Monkey, Oil use the same implementation. The reason they are different is that this allow us to use instanceof method inside GameEngine class and our current plan is to create animations inside their respective classes.

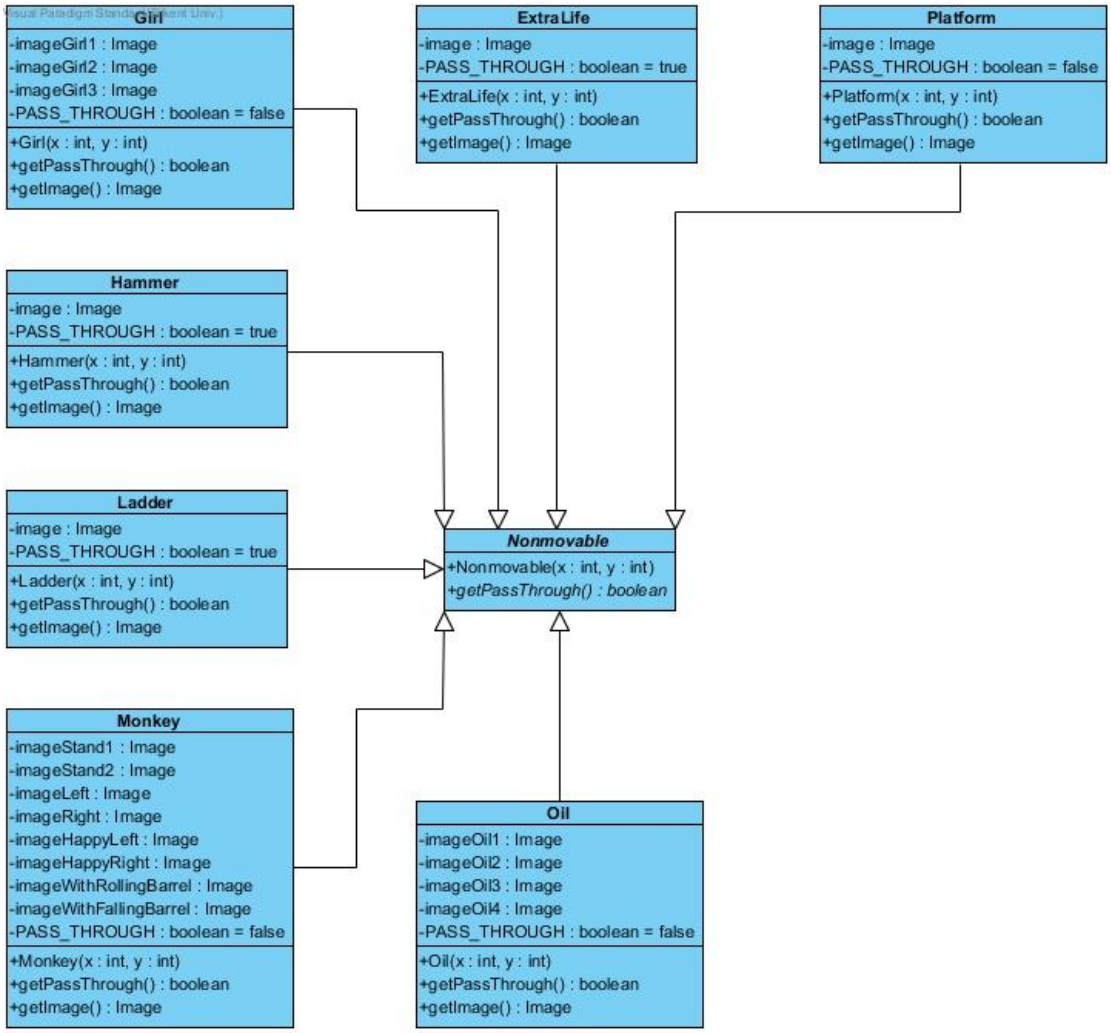

Figure 3.3.3.2: ExtraLife, Girl, Hammer, Ladder, Monkey, Nonmovable, Platform, Oil Classes

Attributes (ExtraLife, Girl, Hammer, Ladder, Monkey, Platform, Oil)

private	PASS_THROUGH	boolean	Determines whether or not Jumpman can able to pass them through.

- Images will be used for animations. Therefore they will not be explained in this document.

Constructors

- All of the classes in this section call their super's contractor which is MyObjects' constructor.

Methods (Nonmovable)

public	getPassThrough()	boolean	Abstract Method

Methods (ExtraLife, Girl, Hammer, Ladder, Monkey, Platform, Oil)

public	getPassThrough()	boolean	Returns PASS_THROUGH
public	getImage()	Image	Returns their respective image

3.3.3.3 Enemy Abstract Class

Figure 3.3.3.3: Enemy Abtract Class

Attributes

private	direction	Direction	Enum type that indicates the object's current direction such as RIGHT or LEFT
private	enemyType	EnemyType	Enum type that indicates the object's type such as ROLLING_BARREL or FIRE_EMENETAL

Constructors

public	Enemy(x : int, y : int)	Child classes inherit the implementation

Methods

public	getDirection()	Direction	Returns direction
public	setDirection(direction : Direction)	void	Sets direction to parameter
public	getEnemyType()	EnemyType	Returns enemyType
public	setEnemyType(enemyType: EnemyType)	void	Sets enemyType to parameter
public	goRight()	void	Increase the X-coordinate of Enemy object
public	goLeft()	void	Decrease the X-coordinate of Enemy object

3.3.3.4 RollingBarrel & FallingBarrel Classes

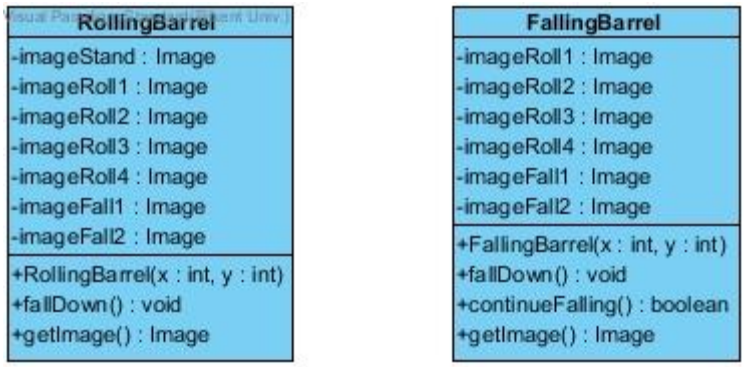

Figure 3.3.3.4: RollingBarrel & FallingBarrel Classes

Attributes

- Images will be used for animations. Therefore they will not be explained in this document.

Constructors

public	RollingBarrel(x : int, y : int)	Create a RollingBarrel object
public	FallingBarrel(x : int, y : int)	Create a FallingBarrel object

Methods

public	fallDown()	void	Increase the Y-coordinate of barrel object*
public	continueFalling()	boolean	Only difference between RollingBarrel and FallingBarrel is that FallingBarrel can falldown immediately. Thus, this method give a Boolean value to decide if it will fall or roll.
public	getImage()	Image	Returns their respective image

- (*) JFrame take upper-left side as (0, 0) and to go down on the frame we increase Y value.

3.3.3.5 FireElemental Class

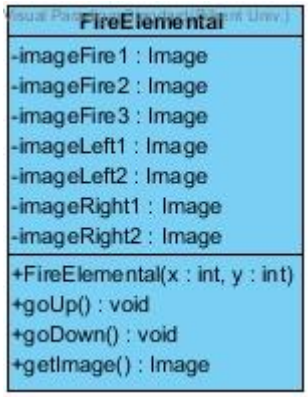

Figure 3.3.3.5: FireElemental Class

Attributes

- Images will be used for animations. Therefore they will not be explained in this document.

Constructors

public	FireElemental(x : int, y : int)	Create a FireElemental object

Methods

public	goUp()	void	Decrease the Y-coordinate of FireElemental object(*)
public	goDown()	void	Increase the Y-coordibate of FireElemental object(**)
public	getImage()	Image	Returns the current image

- (*) & (**) JFrame take upper-left side as (0, 0) and to go down on the frame we increase Y value.

3.3.3.6 Player Class

Player class represent Jumpman character in the game.

Figure 3.3.3.6: Player Class

Attributes

private	direction	Direction	Enum type that indicates the object's current direction such as RIGHT or LEFT

- Nonmovable, barrels, fireElementals and player has a instance inside this class in order to reach them paintComponent(g : Graphics) method easily. Otherwise there is no reason to keep them as objects.

Constructors

public	Player (x : int, y : int)	Calls super class' contractor which is MyObjects' constructor.

Methods

public	goUp()	void	Decrease the Y-coordinate of Player object
public	goDown()	void	Increase the Y-coordinate of Player object
public	goLeft()	void	Decrease the X-coordinate of Player object
public	goRight()	void	Increase the X-coordinate of Player object
public	getDirection()	Direction	Returns direction
public	setDirection(direction : Direction)	void	Sets direction to parameter
public	getImage()	Image	Returns the current image

4. 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.