Bilkent University

Department of Computer Engineering

CS319 – Object Oriented Software Engineering Project

Project short-name: Donkey Kong Game

Analysis Report (Final Draft)

Group 3E
Fuad Ahmadov
Çağatay Küpeli
Sine Mete
Arkın Yılmaz

Supervisor: Bora Güngören

Analysis Report (Final Draft)
Nov 18, 2017

Contents

Analysis Report (Final Draft)

Project short-name: Donkey Kong Game

1 Introduction

Donkey Kong is a well-known old arcade game which has been an inspiration to many other games from its generation. It is a basic platform-action game. Your main objective in this game is saving the Pauline from Donkey Kong by climbing over platforms and dodging enemies such as barrels and fire elementals.
This report contains an overview of the game, system requirements, use-case models including scenarios, use-case diagrams, class and dynamic models.

2 Current System

Donkey Kong was created by Shigeru Miyamo in 1981. He came up with some characters, Jumpman (soon to be Mario) and Donkey Kong. The idea for the name Kong came from the movie King Kong and the "Donkey" part apparently meant stupid according to a Japanese/English dictionary [1]. Additionally, Donkey Kong is the debut game of Mario and also is notable for being one of the first complete narratives in video game form, told through simplistic cut scenes that advance the story [2], which affected our choice.

3 Proposed System

In addition to original game, we are planning to add some new features and cancel some features from original Donkey Kong. Aim of the game is to reach Pauline by avoiding enemies such as barrels and fire elementals. Main villain of the game is monkey who kidnapped the princess. The game will be desktop application and it will be controlled by a keyboard.

3.1 Overview

3.1.1 Gameplay

The gameplay focuses on controlling the Jumpman across platforms while dodging and jumping over obstacles. Player in each level had to save Pauline, pink dressed female character, from giant ape Donkey Kong.

To complete the level, Jumpman must reach Pauline without hitting any obstacles. To do so, the player should utilize timely jumps, ladder climbing skills, navigating correct path, collecting power-ups and collecting coins to get additional points.

Donkey Kong requires great amount of skill and little bit of luck, especially dealing with barrels. For example, a barrel might fall the ladder you are climbing at that moment. During the game, Donkey Kong will throw some barrels which might fall without rolling or roll and fall from a random ladder or at the end of the platform. Moreover barrels spawn fire elementals which is another type of enemy can climb ladders.

3.1.2 Enemy Types

Rolling Barrel (Yellow): Rolling barrel can both roll and fall; however, it needs ladder or empty space at the end of the platform to fall.

Falling Barrel (Blue): Falling barrel can both roll and fall. Behavior of falling barrels is very similar to rolling barrel; however, it can also fall from the platform without ladder or empty space.
Fire Elemental: Fire elemental spawns when barrel hits oil. Its movement pattern is similar to Jumpman. It can go right and left. Moreover it can go up and down from ladders. It cannot be destroyed by hammer.

3.1.3 Enhancements/Power-ups

- Hammer: Hammer allows Jumpman to destroy barrels in front of him. To be able to use hammer, user must reach the hammer icon on the screen. It will only last small amount of time.
- Extra Life: Player starts with 3 life and they will lose one every time they hit by any obstacles.
- Coin: Allow user to earn more score.

3.1.4 Other Objects

- Jumpman: Jumpman is a user controlled unit. To complete the level, user must reach Pauline without hitting any obstacles. It can jump, climb and strike.
- Pauline: Pink dressed female character.
- Oil: Oil spawns fire elementals when it is hit by a barrel.
- Monkey: There is no point reaching monkey. If you get so close, you will die.
- Ladder: Jumpman uses ladders to climb other platforms.

3.2 Functional Requirements

- User can start from any level they unlocked in their earlier runs. However at the end they might not reach enough score to hit the high score table in the end.
- User should be able to control the Jumpman with keyboard inputs such as W, A, S, D. W is for jumping and climbing. A is for going left direction. S is for doing down from a ladder. D is for going right direction.
- User can see their score during game. There will be a score panel that shows current score of the player during gameplay.
- User should be able to pause the game with keyboard input such as ESC button.
- User can move around in the menu by using keyboard inputs such as W and S. W is for going up and S is for going down.
- If user can reach a high score, game should store it with their name or nickname.
- The game should contain an option menu that contains settings, help and credits.
- User should be able to turn on and off the game sound.
- Help should provide information about how to play the game.
- High score table should show at least 5 records.

3.3 Non-functional Requirements

- The game should be easily understandable. After reading how to control the character, there should be conflict about how to control the character and what to do during gameplay. There will be a help section inside option menu.
- The game should have a low and efficient response time so that user play the game with minimal delay. Otherwise user might end up dying without purpose.
- The game should have a high frame rate in order to allow user to do jumps very easily.

- Realistic gameplay. Gravity should increase while player is not on the ground and it should fix at some point.
- Jumpman's animations and actions should be smooth and realistic. It should not teleport.
- Source code design pattern should fit into Model View Controller Pattern
- Source code should be well organized and usable in order to add new features easier.
- Source code generate any given level without making another class for them. Therefore there should be no difficulty of adding extra levels.

3.4 Pseudo Requirements

All related software associated with Donkey Kong Game, should be written using Java.

3.5 System Models

3.5.1 Use-Case Model

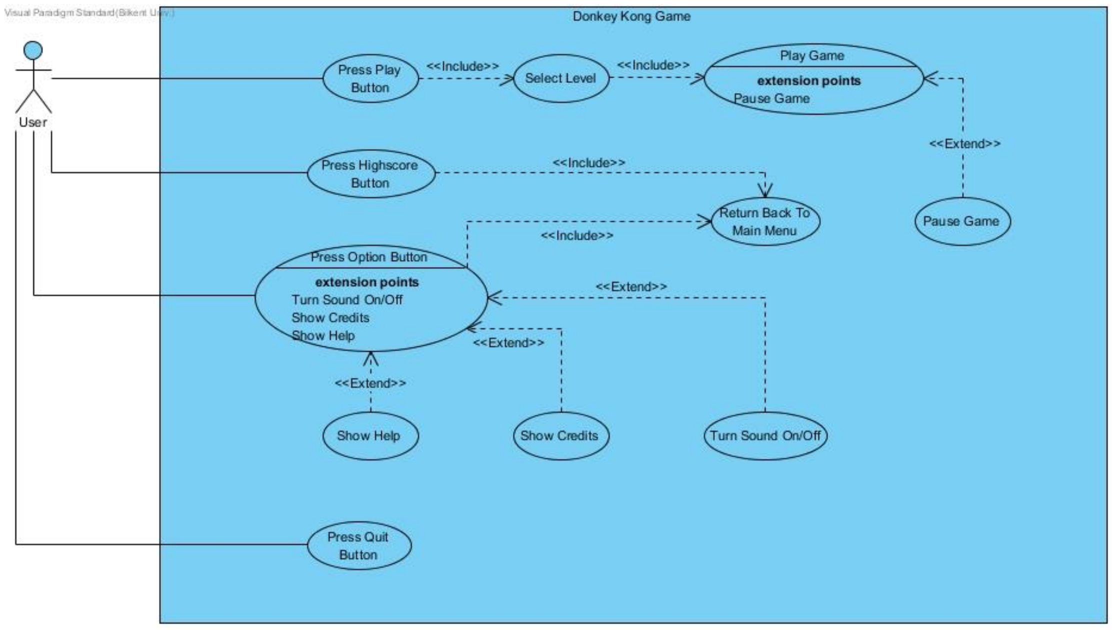


Figure 3.5.1.a: Use-case Model

3.5.2 Use-Case Descriptions

Use Case #1

Use case name: Press Play Button

Participating actors: Initiated by User

Flow of events
1. User presses Enter while the indicator points Play.
2. System comes up with Level Select Menu.

Entry condition:
- User has opened Main Menu via opening the game.
- User has pressed Return Back to Main Menu option inside Highscore or Option Menu.

Exit condition:
- User has to use default window close operation. In other words, User has to press red X button on the window.

Use Case #2

Use case name: Select Level

Participating actors: Initiated by User

Flow of events
1. User selects a level which he or she unlocked before.
2. User presses Enter.
3. System comes up with Game Window.

Entry condition:
- User has opened Level Select Menu by pressing Enter on Main Menu while the indicator points Play in Main Menu.

Exit condition:
- User has to choose a level to start the game.
- User has to use default window close operation. In other words, User has to press red X button on the window.

Use Case #3

Use case name: Play Game

Participating actors: Initiated by User

Main flow of events
1. User start playing the game.
2. User complete all the levels.
3. System comes up with Main Menu.

Alternative Flow of events
1. User start playing the game.
2. He or she gets bored and clicks red X button on the window.

Alternative Flow of events
1. User start playing the game.
2. User complete all the levels.
3. Game Windows pops a message saying that "New High Score"
4. User enters a name.
5. System comes up with Main Menu.

Entry condition:
- User has opened Game Window by pressing Enter on Level Select Menu while the indicator points an unlocked level or 1st level.

Exit condition:
- User has to complete all the levels.
- User has to lose all of his or her lives.
- User has to use default window close operation. In other words, User has to press red X button on the window.

Use Case #4

Use case name: Pause

Participating actors: Initiated by User

Main flow of events

1. User start playing the game.
2. User presses ESC.
3. Game pauses.

Entry condition:

- User has opened Game Window by pressing Enter on Level Select Menu while the indicator points an unlocked level or 1st level.

Exit condition:

- User has to press ESC again to continue playing.
- User has to use default window close operation. In other words, User has to press red X button on the window.

Use Case #5

Use case name: Press Highscore Button

Participating actors: Initiated by User

Main flow of events

1. User presses Enter while the indicator points Highscore.
2. System comes up with Highscore Menu.

Entry condition:

- User has opened Main Menu via opening the game.
- User has pressed Return Back to Main Menu option inside Highscore or Option Menu.

Exit condition:

- User has to press Enter while the indicator points Return Back to Main Menu.
- User has to use default window close operation. In other words, User has to press red X button on the window.

Use Case #6

Use case name: Press Option Button

Participating actors: Initiated by User

Main flow of events

1. User presses Enter while the indicator points Options.
2. System comes up with Option Menu.

Entry condition:

- User has opened Main Menu via opening the game.
- User has pressed Return Back to Main Menu option inside Highscore or Option Menu.

Exit condition:

- User has to press Enter while the indicator points Return Back to Main Menu.
- User has to use default window close operation. In other words, User has to press red X button on the window.

Use Case #6

Use case name: Return Back to Main Menu

Participating actors: Initiated by User

Main flow of events

1. User presses Enter while the indicator points Return Back to Main Menu.
2. System comes up with Main Menu.

Entry condition:

- User has pressed Enter while the indicator points Highscore or Options inside Main Menu

Exit condition:

- User has to use default window close operation. In other words, User has to press red X button on the window.

Use Case #7

Use case name: Show Help

Participating actors: Initiated by User

Main flow of events

1. User presses Enter while the indicator points Help.
2. User reads the instructions shown by the system.
3. System comes up with Option Menu.

Entry condition:

- User has pressed enter while indicator points Help inside Option Menu.

Exit condition:

- User has to press Enter while indicator points Return Back to Main Menu.
- User has to use default window close operation. In other words, User has to press red X button on the window.

Use Case #8

Use case name: Show Credits

Participating actors: Initiated by User

Main flow of events

1. User presses Enter while the indicator points Credits.
2. User reads the information shown by the system.
3. System comes up with Option Menu.

Entry condition:

- User has pressed enter while indicator points Credits inside Option Menu.

Exit condition:

- User has to press Enter while indicator points Return Back to Main Menu.
- User has to use default window close operation. In other words, User has to press red X button on the window.

Use Case #9

Use case name: Turn Sound On/Off

Participating actors: Initiated by User

Main flow of events

1. User presses Enter while the indicator points Credits.
2. User reads the information shown by the system.
3. System comes up with Option Menu.

Entry condition:

- User has pressed Enter while indicator points Options inside Main Menu.

Exit condition:

- User has to use default window close operation. In other words, User has to press red X button on the window.

Use Case #10

Use case name: Press Quit

Participating actors: Initiated by User

Main flow of events

1. User presses Enter while the indicator points Quit.
2. Program is closed.

Entry condition:

- User has opened Main Menu via opening the game.
- User has pressed Return Back to Main Menu option inside Highscore or Option Menu.

Exit condition:

- User has to use default window close operation. In other words, User has to press red X button on the window.

3.5.3 Scenarios

3.5.3.1 Scenario #1 (Start Level 1)

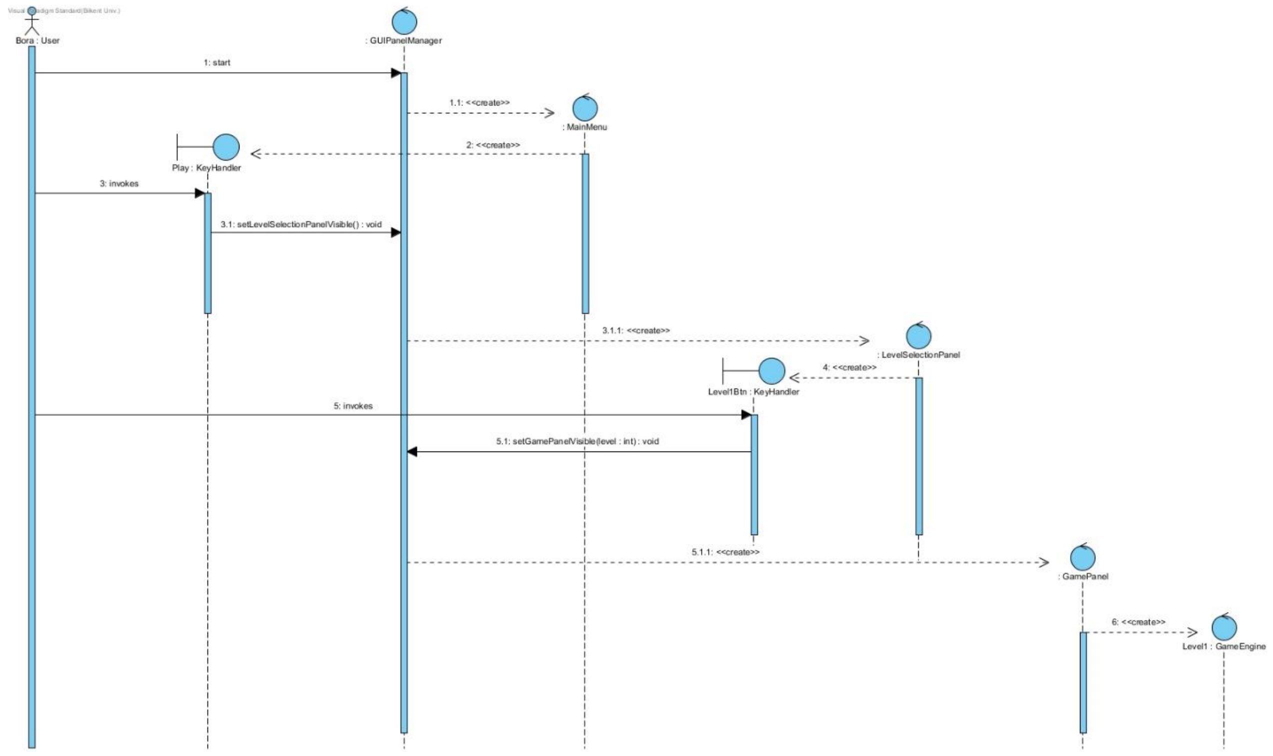


Figure 3.5.3.1.a: Scenario #1 (Start Level 1)

- User starts the game. Then game starts by creating GUIPanelManager. GUIPanelManager creates an instance of MainMenu.
- MainMenu creates Play object which is an instance of KeyHandler that interact with user.
- If user invokes Play object, it calls setLevelSelectionPanelVisible() function inside GUIPanelManager. Then GUIPanelManager creates a LevelSelectionPanel and make previous panel invisible and setContent as LevelSelectionPanel. At this point user sees LevelSelectionPanel.
- LevelSelectionPanel creates Level objects which are instances of KeyHandler that interact with user.

Remark: For this particular scenario user invokes Level1 object. For some other example it can invoke Level2 etc.

- If user invokes Level1 object, it calls setGamePanelVisible(1), 1 indicates the level number, inside GUIPanelManager. Then GUIPanelManager creates a GamePanel and make previous panel invisible and setContent as GamePanel. At this point user sees GamePanel.
- GamePanels creates appropriate GameEngine according to the level it takes as an input in its constructor.

3.5.3.2 Scenario #2 (Play Game)

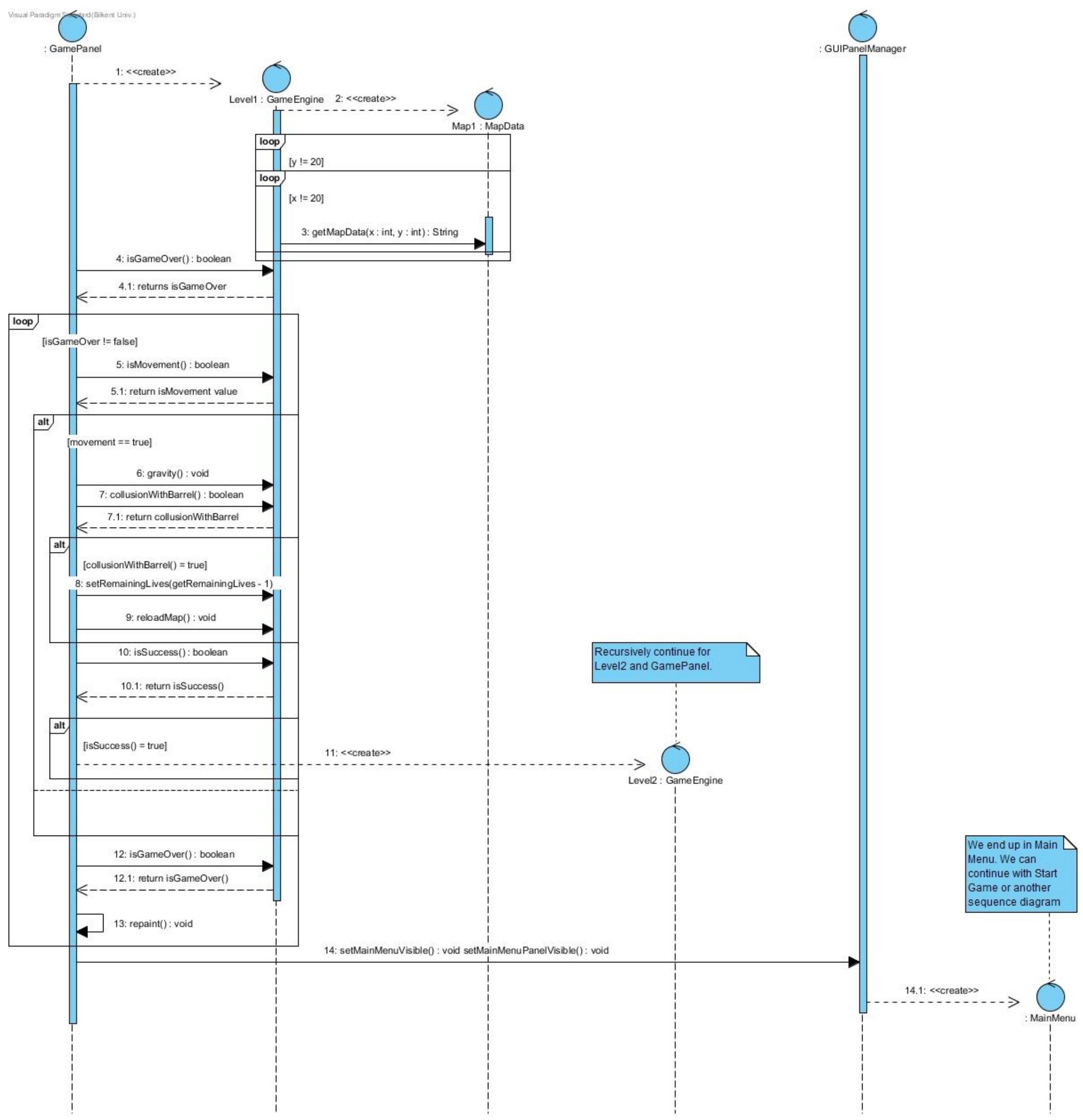

Visual Paradigm Standard(Bilkent Univ.)
: GamePanel
: GUIPanelManager
1: <<create>>
Level1 : GameEngine
2: <<create>>
Map1 : MapData
loop
[y != 20]
loop
[x != 20]
3: getMapData(x : int, y : int) : String
4: isGameOver() : boolean
4.1: returns isGameOver
loop
[isGameOver != false]
5: isMovement() : boolean
5.1: return isMovement value
alt
[movement == true]
6: gravity() : void
7: collusionWithBarrel() : boolean
7.1: return collusionWithBarrel
alt
[collusionWithBarrel() = true]
8: setRemainingLives(getRemainingLives - 1)
9: reloadMap() : void
10: isSuccess() : boolean
10.1: return isSuccess()
alt
[isSuccess() = true]
11: <<create>>
Level2 : GameEngine
Recursively continue for Level2 and GamePanel.
12: isGameOver() : boolean
12.1: return isGameOver()
13: repaint() : void
14: setMainMenuVisible() : void setMainMenuPanelVisible() : void
14.1: <<create>>
: MainMenu
We end up in Main Menu. We can continue with Start Game or another sequence diagram

Figure 3.5.3.2.a: Scenario #2 (Play Game)

Remark: This sequence diagram take place right after Start Game diagram.

- GamePanel creates appropriate GameEngine according to the level it takes as an input in its constructor.

Remark: For this particular scenario, we start from level1.

- GameEngine creates appropriate MapData according to the level it takes as an input in its constructor.

- GameEngine loads Map first level.
- Next loop is representation of run() function inside GamePanel.
- GamePanel realize whether the game is over or not with isGameOver() function.
- GamePanel realize whether the game is paused or not with isMovement() function.
- 1st part of the 1st alt is presentation of when the game is running. Its 2nd part is presentation of when the game is paused. Currently we do not have initiative to make a pause menu; but if we do we will write down the code at there.
- 2nd alt decides if the Jumpman hit a barrel or not with collusionWithBarrel() function.
- 3rd alt decides if user successfully finished the game with isSuccess() function. Then creates a following level. For this particular scenario, it is level 2.
- It uses repaint() function to call paintComponents(g) function. Its purpose is to rendering the screen.
- If the user loses isGameOver() becomes true and the system send user to a new constructed MainMenu.

3.5.3.3 Scenario #3 (Change Sound)

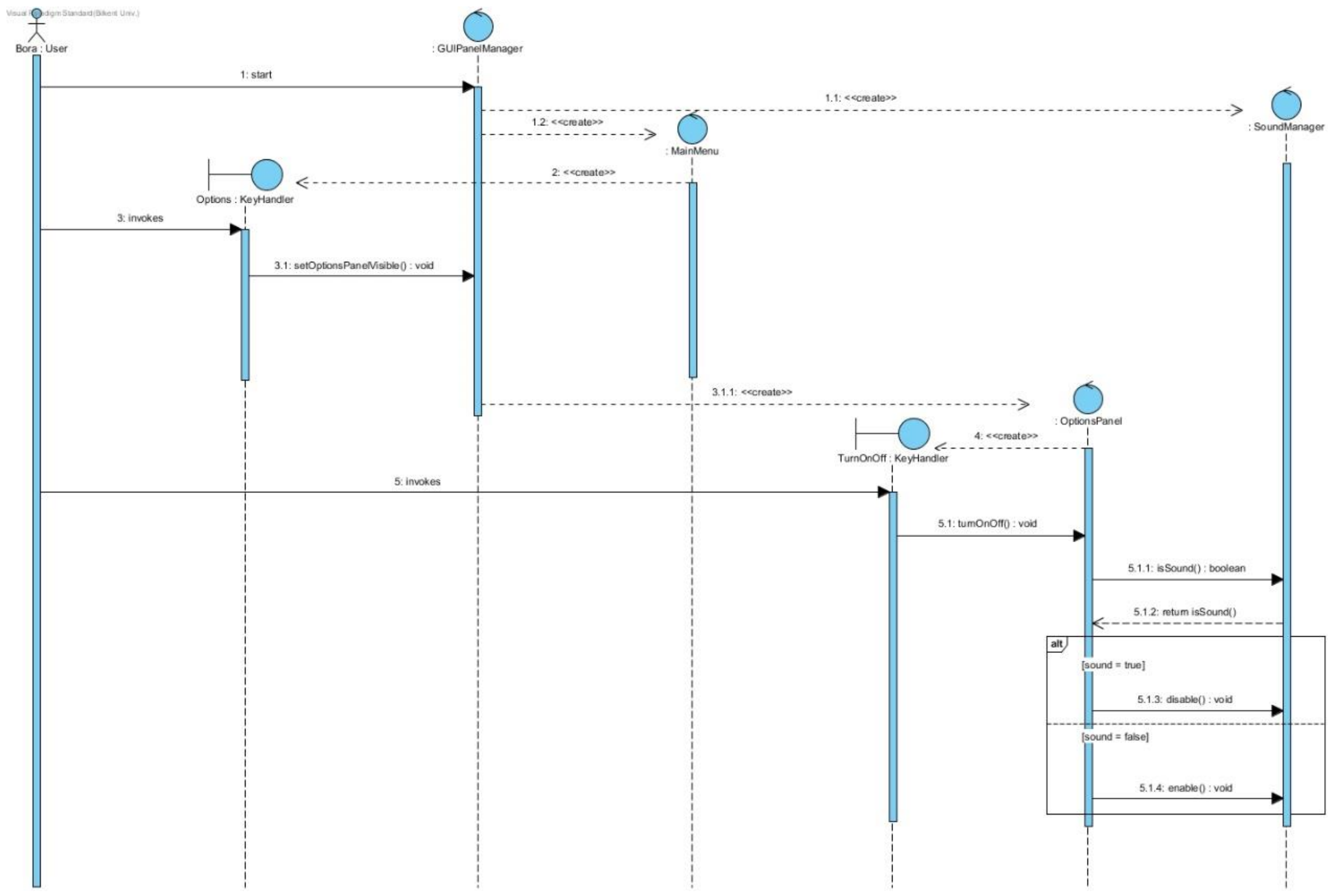
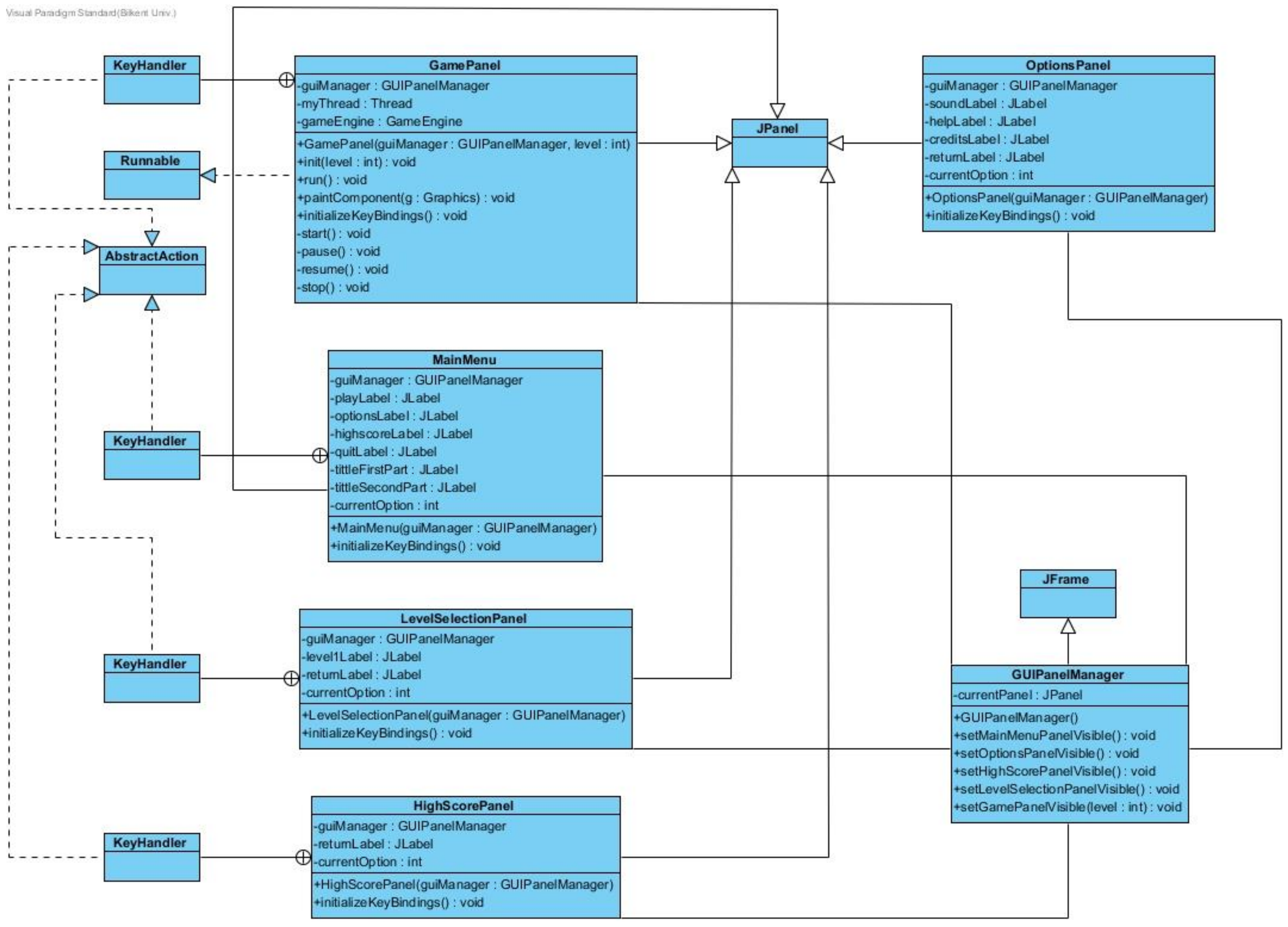


Figure 3.5.3.3.a: Scenario #3 (Change Sound)

- User starts the game. Then game starts by creating GUIPanelManager. GUIPanelManager creates an instance of MainMenu.
- MainMenu creates Play object which is an instance of KeyHandler that interact with user.
- If user invokes Option object, it calls setOptionPanelVisible() function inside GUIPanelManager. Then GUIPanelManager creates an OptionPanel and make previous panel invisible and setContent as OptionPanel. At this point user sees Option Menu.
- OptionPanel creates TurnOnOff object which is an instance of KeyHandler that interact with user.
- If user invokes TurnOnOff object, it calls turnOnOff() function inside OptionPanel that interacts with SoundManager.
- If the sound is true, the system calls disable(). If the sound is false, the system calls enable() functions.

3.5.4 Object and Class Mode



Visual Paradigm Standard(Bilkent Univ.)
KeyHandler
Runnable
AbstractAction
GamePanel
-guiManager : GUIPanelManager
-myThread : Thread
-gameEngine : GameEngine
+GamePanel(guiManager : GUIPanelManager, level : int)
+init(level : int) : void
+run() : void
+paintComponent(g : Graphics) : void
+initializeKeyBindings() : void
-start() : void
-pause() : void
-resume() : void
-stop() : void
JPanel
OptionsPanel
-guiManager : GUIPanelManager
-soundLabel : JLabel
-helpLabel : JLabel
-creditsLabel : JLabel
-returnLabel : JLabel
-currentOption : int
+OptionsPanel(guiManager : GUIPanelManager)
+initializeKeyBindings() : void
MainMenu
-guiManager : GUIPanelManager
-playLabel : JLabel
-optionsLabel : JLabel
-highscoreLabel : JLabel
-quitLabel : JLabel
-tittleFirstPart : JLabel
-tittleSecondPart : JLabel
-currentOption : int
+MainMenu(guiManager : GUIPanelManager)
+initializeKeyBindings() : void
JFrame
LevelSelectionPanel
-guiManager : GUIPanelManager
-level1Label : JLabel
-returnLabel : JLabel
-currentOption : int
+LevelSelectionPanel(guiManager : GUIPanelManager)
+initializeKeyBindings() : void
GUIPanelManager
-currentPanel : JPanel
+GUIPanelManager()
+setMainMenuPanelVisible() : void
+setOptionsPanelVisible() : void
+setHighScorePanelVisible() : void
+setLevelSelectionPanelVisible() : void
+setGamePanelVisible(level : int) : void
HighScorePanel
-guiManager : GUIPanelManager
-returnLabel : JLabel
-currentOption : int
+HighScorePanel(guiManager : GUIPanelManager)
+initializeKeyBindings() : void

Figure 3.5.4.a: Class Diagram (GUI Components)

- **GUIPanelManager** extends JFrame. It creates our main frame and manages which panel will be shown.
- **MainMenu** extends JPanel. It creates our main menu components and according to the user inputs allow user to move around main menu. Therefore it provides inputs for GUIPanelManager to show which panel will be visible.
- **LevelSelectionPanel** extends JPanel. It creates our level selection components and according to the user inputs allow user to move around level selection menu. Therefore it provides inputs for GUIPanelManager to show which level will be loaded.
- **OptionPanel** extends JPanel. It creates our option menu components and according to the user inputs allow user to move around option menu. Therefore it provides inputs for GUIPanelManager to show which option will be visible.
- **Highscore** extends JPanel. It creates our high score menu components and provides highscores. This class also communicate with ScoreData class which will be explained below to show high scores stored.
- **GamePanel** extends JPanel. It interacts with GameEngine class which will be explained below to render game state. This class is a visual presentation of GameEngine class.

- **KeyHandler** is Keybinding class. Every class which extends JPanel, also implements this class as a nested class.

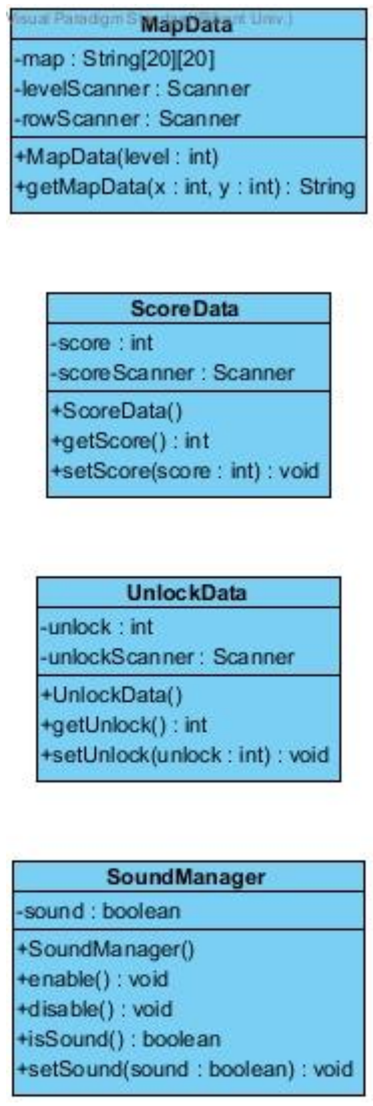
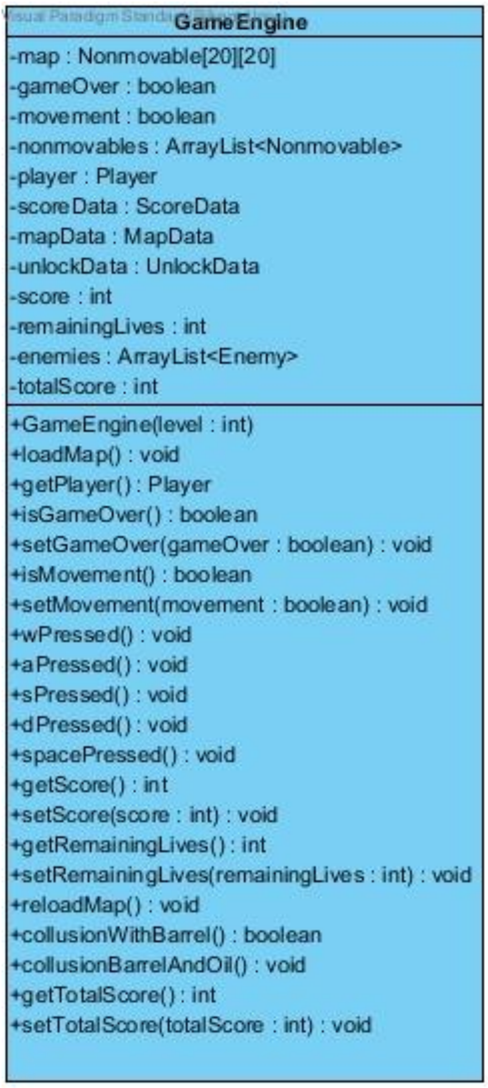


Figure 3.5.4.b: Class Diagram (File Management)

- **MapData** stores the map layout at the start of the levels which cannot be changed.
- **ScoreData** stores the highest scores and current score of the game.
- **UnlockData** stores the levels which are unlocked by user.



```
GameEngine
-map : Nonmovable[20][20]
-gameOver : boolean
-movement : boolean
-nonmovables : ArrayList<Nonmovable>
-player : Player
-scoreData : ScoreData
-mapData : MapData
-unlockData : UnlockData
-score : int
-remainingLives : int
-enemies : ArrayList<Enemy>
-totalScore : int
+GameEngine(level : int)
+loadMap() : void
+getPlayer() : Player
+isGameOver() : boolean
+setGameOver(gameOver : boolean) : void
+isMovement() : boolean
+setMovement(movement : boolean) : void
+wPressed() : void
+aPressed() : void
+sPressed() : void
+dPressed() : void
+spacePressed() : void
+getScore() : int
+setScore(score : int) : void
+getRemainingLives() : int
+setRemainingLives(remainingLives : int) : void
+reloadMap() : void
+collusionWithBarrel() : boolean
+collusionBarrelAndOil() : void
+getTotalScore() : int
+setTotalScore(totalScore : int) : void
```

Figure 3.5.4.c: Class Diagram (GameEngine Class)

- **GameEngine** takes the map layout from the MapData and score from the ScoreData. The game will be played inside this class and it will rendered by GamePanel. This class takes user inputs by using GamePanel class. Every game mechanic and collusion will be implemented inside this class.

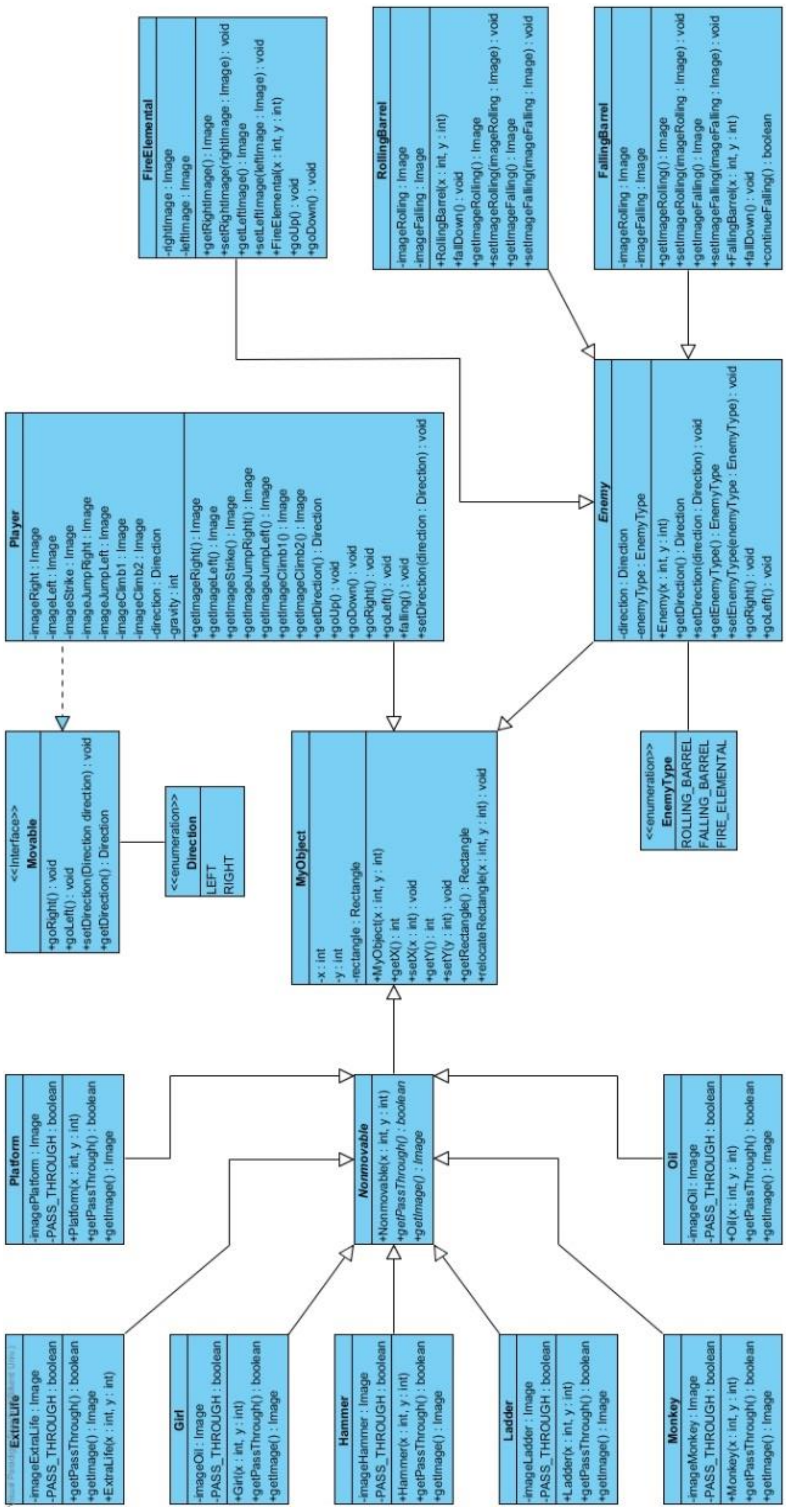
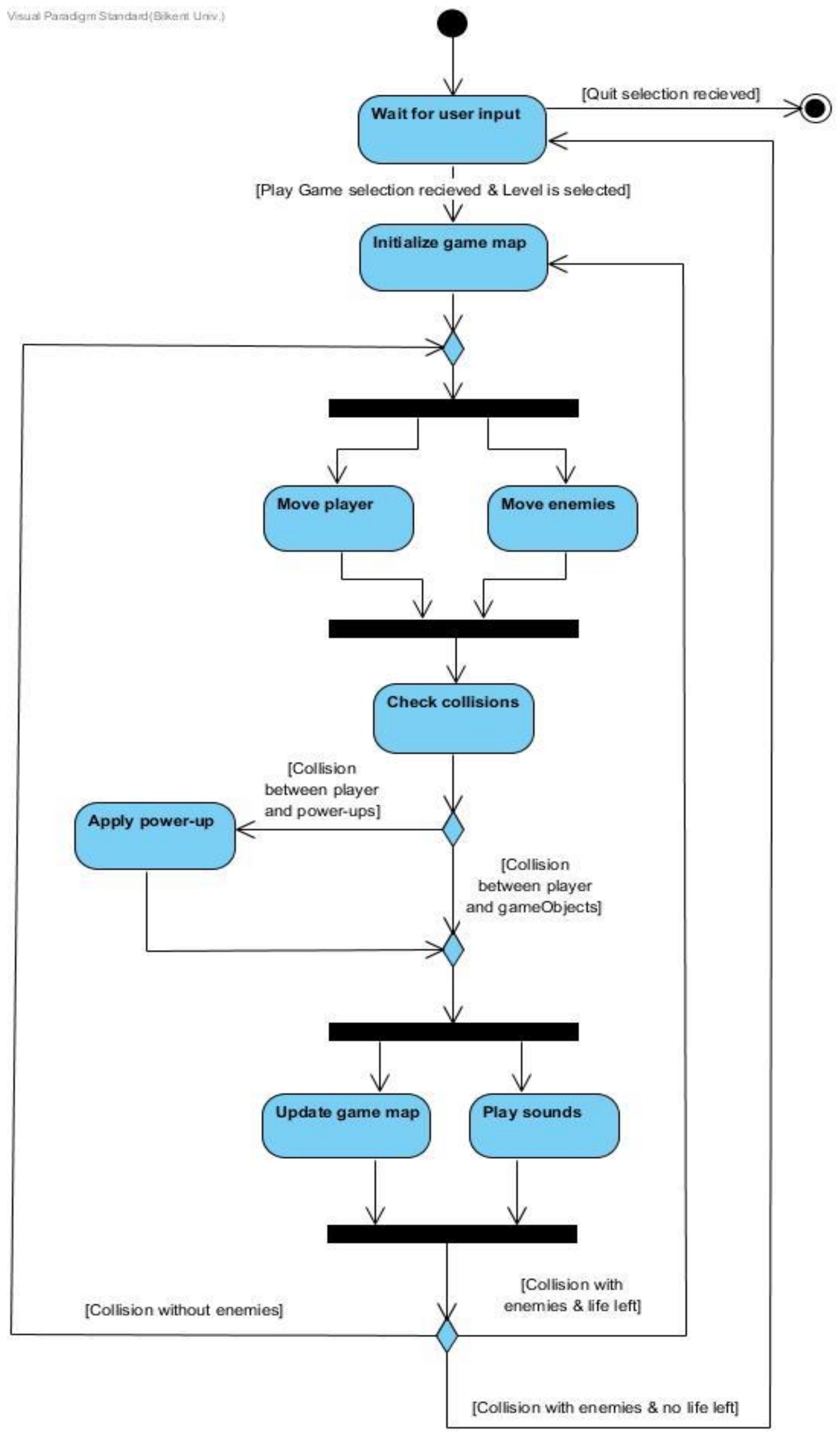


Figure 3.5.4.d: Class Diagram (Game Entities)

- **Movable** is an interface for Player class.
- **Direction** and **EnemyType** are enum types.
- **MyObject** class is parent of all the other classes in this subsystem. Every objects has a rectangle inside that will help to detect collusions. Every subclass has different methods to match with their functions.

3.5.5 Dynamic Models



Visual Paradigm Standard(Bilkent Univ.)
Wait for user input
[Quit selection recieved]
[Play Game selection recieved & Level is selected]
Initialize game map
Move player
Move enemies
Check collisions
[Collision between player and power-ups]
Apply power-up
[Collision between player and gameObjects]
Update game map
Play sounds
[Collision with enemies & life left]
[Collision without enemies]
[Collision with enemies & no life left]

Figure 3.5.5.a: Activity Diagram

The diagram above shows how the system maintains the gameplay. When user opens the game, the system displays a main menu for the user and waits for the user input. If user selects Play and selects appropriate level to play, the system initializes the game map and the game starts.

According to the user inputs, the system tries to move the main character and the enemies concurrently. Before updating their positions, the system have to check collisions. At first, the system checks collisions between Jumpman and the game objects. There are two types of this collision. First type is between the main character and power-ups such as hammer, extra life or bonus coin. If this collision appears, the system applies power-ups. The second type is between the main character and the other game objects rather than the power-ups. The system updates game map after checking these collisions because without checking them, the system cannot figure whether the desired movement is possible or not. Also, the system play sounds while it is updating the game map. After this point, the system determines the course of events. If Jumpman doesn't collide with enemies, the system tries to handle the next movement of the main character and enemies. If the main character collides with enemies but player have remaining lives, the system initializes the game map and starts the game again. If the main character collides with enemies and player doesn't have remaining lives, the system returns to main menu and wait for the user input again. User can play the game again or he/she can select Quit to exit game.

3.5.6 User Interface

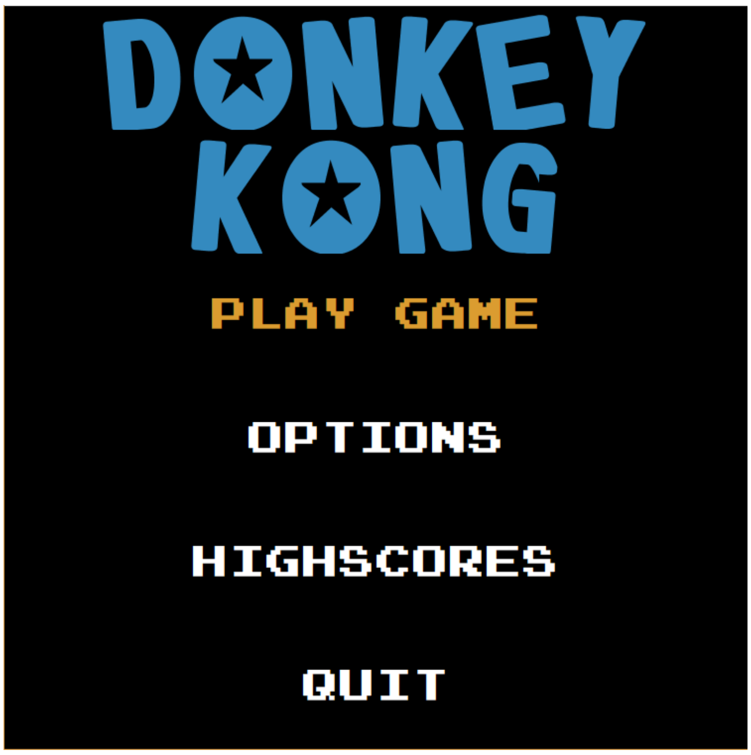


Figure 3.5.6.a: Main Menu

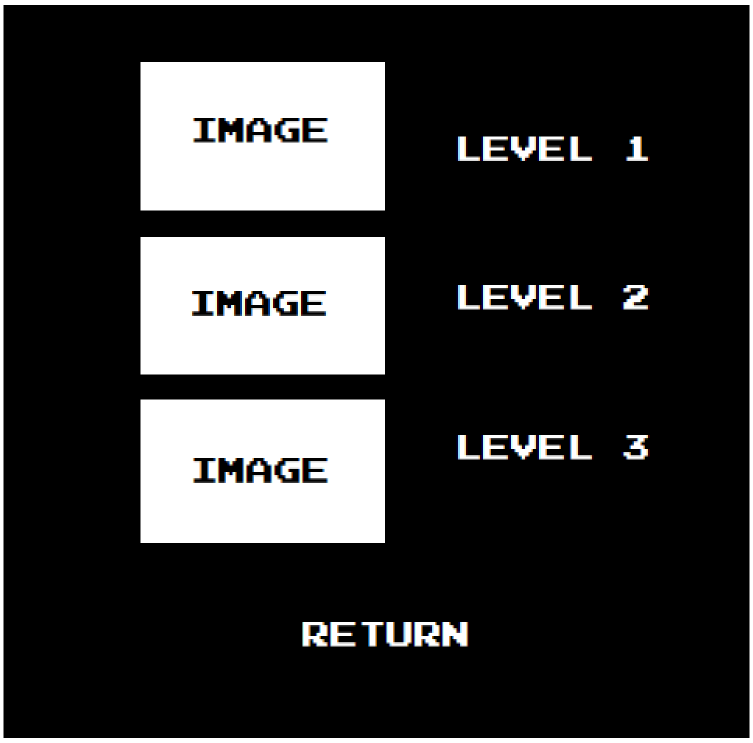


Figure 3.5.6.b: Level Selection

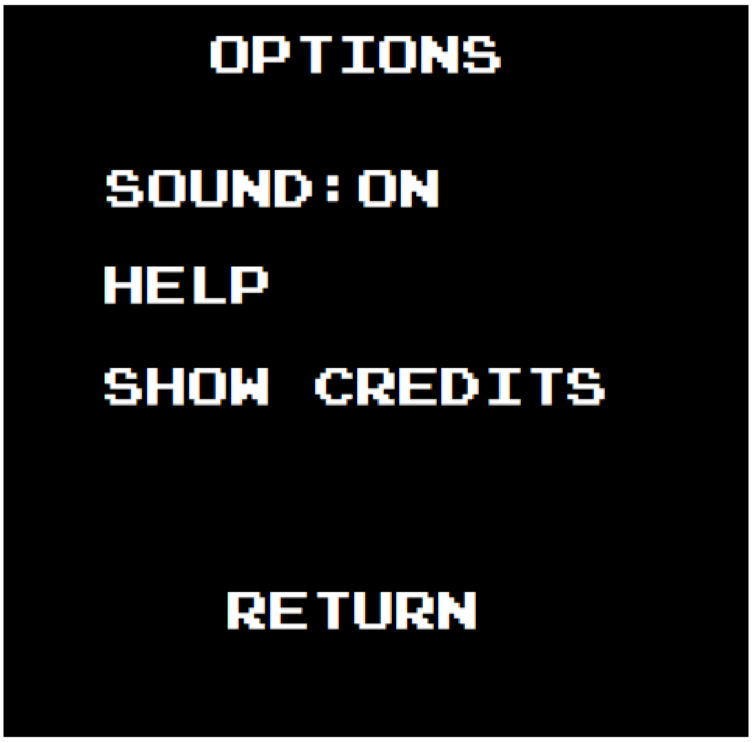
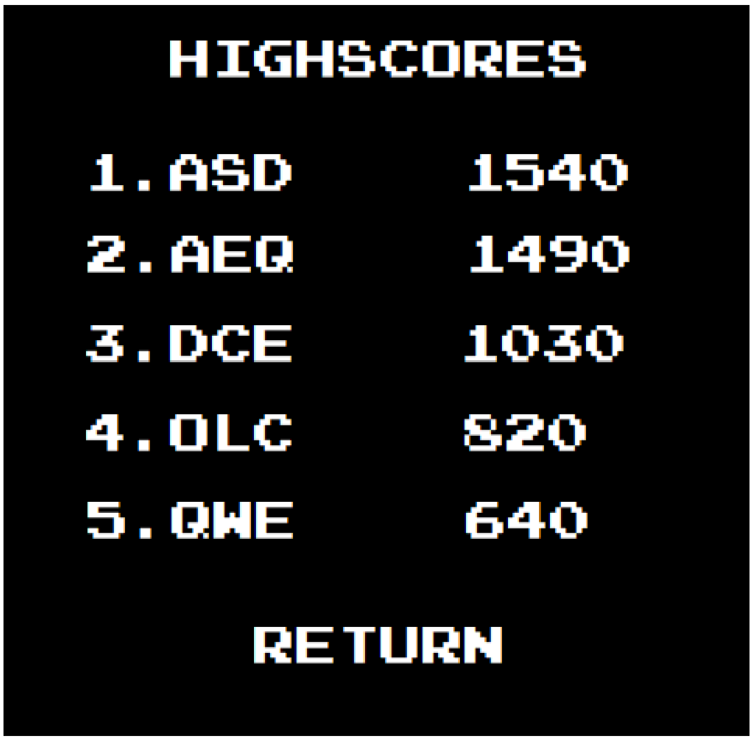


Figure 3.5.6.c: Option Menu



HIGHSCORES

1.ASD	1540
2.AEQ	1490
3.DCE	1030
4.OLC	820
5.QWE	640

Figure 3.5.6.d: Highscore Panel

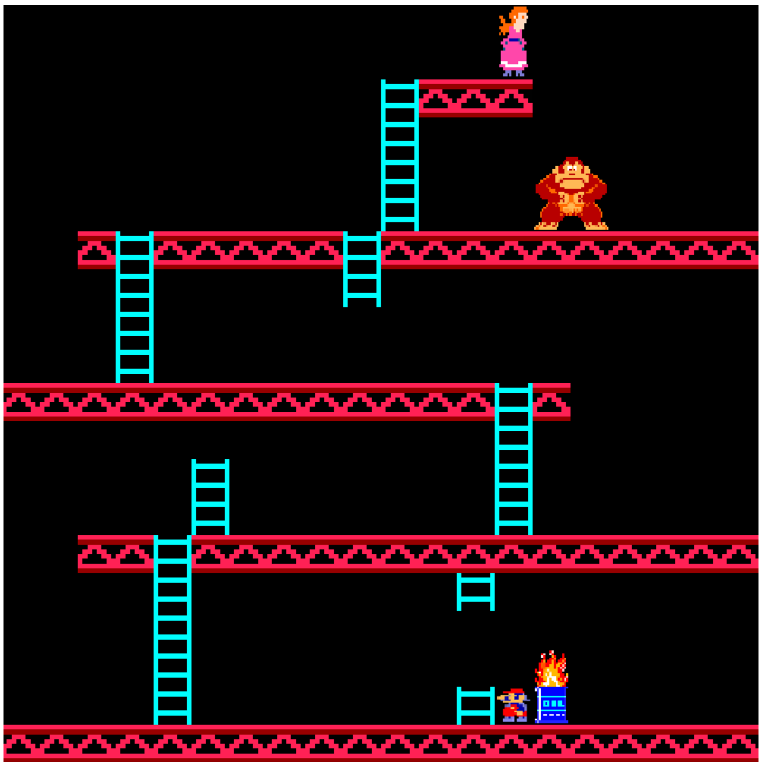


Figure 3.5.6.e: Game Panel

Remark: Game Panel is still under construction. This is an unfinished example. At the end level will make sense and it will show score and remaining lives.

4 Glossary

- **Runnable** is an interface which is defined in Java Standard Library. For more information, use the following link: https://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html

- **JFrame** is an extended version of Java.awt.Frame that adds support for Swing Library. For more information, use the following link: https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html

- **JPanel** is a generic lightweight container which is defined in Java Standard Library. For more information, use the following link: https://docs.oracle.com/javase/7/docs/api/javax/swing/JPanel.html

- **JLabel** displays an area for text, an image which is defined in Java Standard Library. For more information, use the following link: https://docs.oracle.com/javase/7/docs/api/javax/swing/JLabel.html

- **AbstractAction** provides default implementations for the JFC Action interface which is defined in Java Standard Library. For more information, use the following link: https://docs.oracle.com/javase/7/docs/api/javax/swing/AbstractAction.html

- **Rectangle** specifies an area in a coordinate space that is enclosed by the Rectangle object's upper-left point (x, y) in the coordinate space, its width, its height. For more information, use the following link: https://docs.oracle.com/javase/7/docs/api/java/awt/Rectangle.html

- **Image** is the superclass of all classes that represent graphical images. For more information, use the following link: https://docs.oracle.com/javase/7/docs/api/java/awt/Image.html

5 References

[1] "The History of Donkey Kong." *Classic Gaming,* http://www.classicgaming.cc/classics/donkey-kong/history

[2] "Donkey Kong." *MobyGames,* www.mobygames.com/game/donkey-kong