# CS 319 - Object-Oriented Software Engineering

# Analysis Report

Dave Davesson

<u>Group 6</u>

**Ali Atlı** – 21302442

**Berk Türk** – 21302570

**Sefa Gündoğdu -** 21301794

**Burak Özmen** – 21200918

**TABLE of CONTENTS**

## 1. *Introduction*

In our project, we are going to implement a game called Dave Davesson. Dave Davesson is a Java based platform game that is inspired by the old fashioned MSDOS game Dangerous Dave. Like the original, the main goal is to find the key to open the door and advance into the next level.

**Link to Dangerous Dave game:** http://playdosgamesonline.com/dangerous-dave.html

Apart from the original, there will be some different features like multiplayer mode, different weapon types and various maps. In multiplayer mode, two players will start the game from opposite sides of the map and the focus is to eliminate the opposing player with the guns given by the game.

Dangerous Dave will mainly focused on Player-to-Object, but there will be some Object-to-Object focuses, for example obtaining a key to open the doors, killing enemies with the guns' projectiles. Lastly, the game will be played with a keyboard for both players.

## 2. *Overview*

Dave Davesson is a modern day clone of the original Dangerous Dave platform game, with a few additions that will make the game more fun. The original version of the game featured single-player mode where the players would find the cup that is in an uncanny spot in the map, then would go on to open the door to a new map, while also collecting the goods (diamonds, lives, guns etc.) to further increase the propitiousness of the player. This part of the game persists in this version too, with changes in the graphics.

The new version that we will add is the multiplayer mode, where the goal of the game is to kill the opponent with the guns that are placed on the map. The players have health points and the guns have different damage points and different recoil times to allow players to develop distinct play styles.

### *2.1.* ***Game-Play***

The game is played using arrow and 'W', 'A', 'S', 'D' keys on the keyboard. More specifically, the single-player mode will allow users to play with either of them, whereas the multi-player version will assign at the beginning of the game the controller depending of the spawn position of the Dave, the right most Dave will be controlled by arrow keys and vice versa. The Daves will be distinguished by their colors to ease visual understanding of the game.

## 2.2. ***Maps***

The maps in the single-player mode are levelled with respect to their difficulty in the game play. The first map will be very straightforward to pass, a player will be able to pass this map just jumping on the bricks without being exposed to any dangerous elements. As the levels proceed the game will introduce different obstacles ranging from fire, water, bushes to monsters that will cause the loss of life. Some maps are wide and therefore cannot be visually examined at once. This problem will be solved by introducing a border that is positioned close to the continuation of the map whose crossing will bring the remaining part of the map.

The multi-player maps will not have cups or keys but obstacles that will enable users to cover from bursts of guns, moreover, they will feature fire, water and maybe even monsters. The maps will contain guns and health stations to restore health. The multi-player maps will be designed to fit in to a single screen, so as not to further complicate the matter.

## 2.3. ***Dave***

Dave's will have three lives at the beginning of the every map. They will be represented by different graphics in case of jumping, running and gunning. They will have distinct colors for multi-player version. They are the single elements that can be controlled by the players. They are vulnerable to fire, water and bushes. They will be able to jump from bricks that are hung on the map and collect items upon positioning themselves on the items. There will only be one type of brick, as we see no reason to change the functionality of the bricks. The limit for carrying items non-existent except for guns. The maximum number of lives will not exceed three and a hundred health point for single-player and multi-player modes respectively. The speed of jumps and jogs are constant at all times and can be arranged from the settings menu.

## 2.4. ***Gun Types***

There will be three types of guns: sniper, shotgun and pistol.

***Sniper:*** The most powerful weapon in the game. But it has one drawback, its long duration to reset. This gun will bring the opponent to a standstill in one shot. It will fire one bullet per shooting. We plan to hide it in a very hard place in the map.

***Shotgun:*** It bursts three bullets per shooting, and one hit gives only 34 points of health damage. Recoil time is shorter than that of Sniper. The bullets are spread upon firing, so a close shut would suck all the life points of the enemy.

***Pistol:*** The fastest of all guns. It requires five hits to kill the enemy. This gun will be abundant and relatively worthless.

All guns will have unlimited bullets.

## *3. Requirement Specification*

### *3.1 Functional Requirements and Screen Mock-ups*

-In the first view of the game there are 3 options.

*Play Game

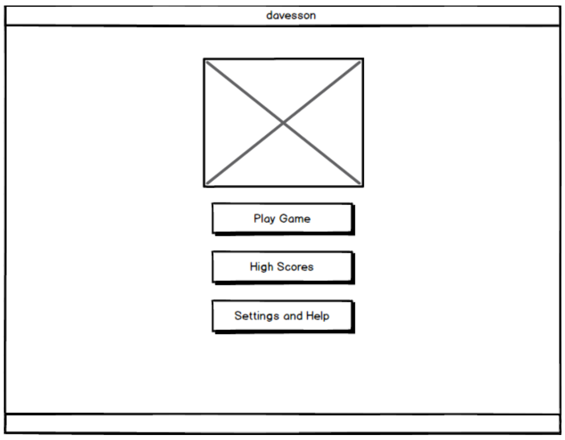
*High Scores

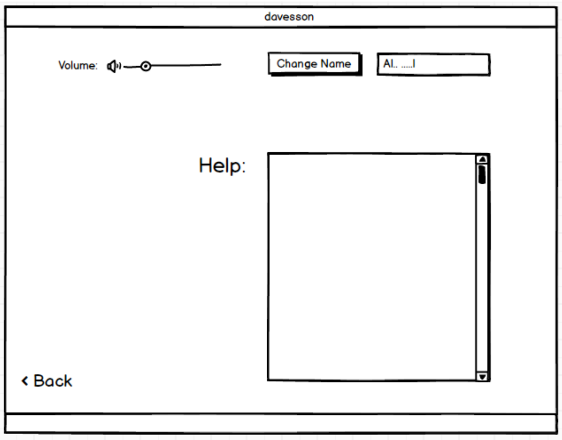
*Settings and Help

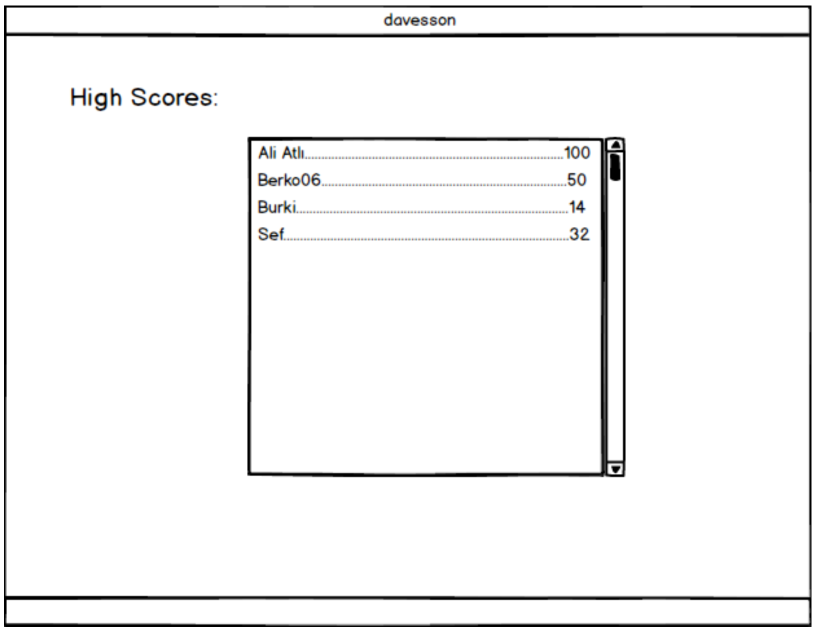
Play Game: This is the option that player starts the game.

High Score: This is the option that player can see and track his/her high scores.

Settings and Help: This is the option that player can read the help that includes instructions about the game and change sound volume, name etc.







| | |
|---|---|
| Ali Atlı | 100 |
| Berko06 | 50 |
| Burki | 14 |
| Sef | 32 |

In addition to play game option, it will provide new options to users like,

-Single Player

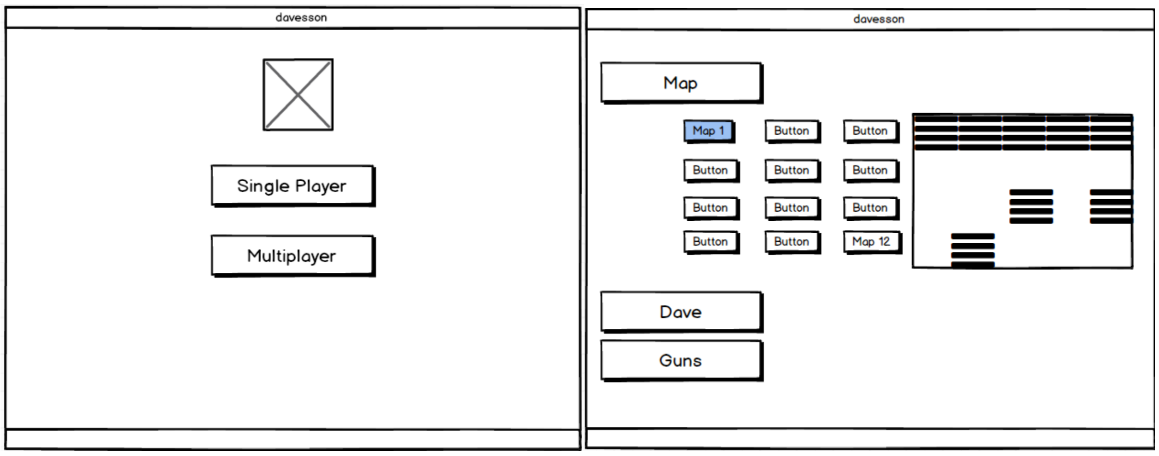
-Multiplayer

These options are important because game will prepare mouse keys, map, and number of characters in the game, the objects of the game such as guns, armor, boss monsters based on these options.

If user choose single player, he/she will play the mandatory game which is the same as real Dave game. Namely, user can't choose guns, map, and Dave versions.

But if user choose multiplayer, he/she will play the game which was chosen by the user.

Also, the user will be able to save and load his/her game.



*(Kind of)

## *3.2 Non-Functional Requirements*

- Game Performance

The game performance is important to have a great gaming experience. So to make it as smooth as possible, we intend to get low latency from keystrokes (<100 ms). Also making the game with better optimization helps to give a good performance, because while playing, there will be objects on higher amounts and collusion between them.

- Graphics Performance

Dangerous Dave is a retro game, so the main goal is to make the graphics look old-fashioned, while giving the feeling that graphics are not look outdated. Also, we intend to use the most efficient graphics for playing the game fluently for computers with different specs, because the graphics play a major role for latency.

- User-friendly Interface

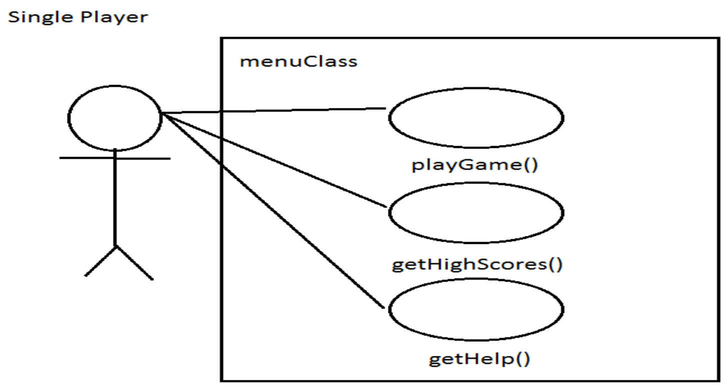
The game has a basic user interface and needs little number of buttons to make the players comfortable while playing the game with ease. There will be little to no extra information despite points, remaining lives and number of level. Our main goal is to make Dangerous Dave an enjoyable game.

- Reusability

Since the game will be implemented in Java, we will try to code Dangerous Dave as simple as possible for the possibility to rework or update it in future. In addition, level design will be easy to code for new players to make their levels to enjoy different experience of playing.

# *4. System Model*

## *4.1. Use Case Model*



| ***4.1.1 Use Case Name :*** | playGame |
|---|---|
| Participating Actors | Clicked by user<br>It communicates with menuClass user interface panel. |
| Flow of events | 1. User clicked to button on menuClass panel and activates the playGame function.<br>2.menuClass responds with creating a Game(panel) object.<br>3.User makes initializations by adjusting the Game object's variables and Dave Object's variables such as number of Player, Player Name, Map, Gun.<br>4.After adjusting, user clicked to Start Game button which activates menuClass's start function<br>5.Start function understands and calls the gamePanel container and starts the game. |
| Entry Condition | After user clicks desktop button and clicked playGame function in menuClass. |
| Exit Condition | 1. The game continues until the players click exit button.<br>2. One player dies, whether the game mode.<br>3. User passes all the levels in singleplayer mode. |

| | |
|---|---|
| Quality Requirements | ---------- |

| | |
|---|---|
| ***4.1.2 Use Case Name:*** | getHighScores |
| Participating Actors | Clicked by user<br>It communicates with menuClass user interface panel. |
| Flow of events | 1. User clicked to button on menuClass Panel and activates getHighScores function.<br>2. Function activates and read scores.txt and displays the high scores. |
| Entry Condition | After user clicked desktop button and clicked getHighScores function in menuClass . |
| Exit Condition | The game continuous until the player clicked exit button to change the panel. |
| Quality Requirements | ---------- |

| | |
|---|---|
| ***4.1.3 Use Case Name:*** | getHelp |
| Participating Actors | Clicked by user<br>It communicates with menuClass user interface panel. |
| Flow of events | 1. User clicked to button on menuClass Panel and activates getHelp function.<br>2. Function activates and read help.txt and displays the help. |
| Entry Condition | After user clicked desktop button and clicked getHelp function in menuClass . |
| Exit Condition | The game continuous until the player clicked exit button to change the panel. |
| Quality Requirements | ---------- |

| ***4.1.4 Use Case Name:*** | gamePanel |
|---|---|
| Participating Actors | Clicked to Start Game button by user<br>It communicates with gamePanel container user interface panel. |
| Flow of events | 1. User clicked to button on menuClass Panel and activates gamePanel container.<br>2. This container creates the map, dave, monsters in the screen. Also, detects the keys that user input such as arrow and 'w', 'a', 's',d' keys.<br>3.Every time user push any of those keys dave moves and gamePanel Activates revalidate function to refresh container.<br>4.Game continuous, until the all maps finish or user click to the exit button. |
| Entry Condition | After user clicked Start Game button on menuClass panel. |
| Exit Condition | The game continuous until the player clicked exit button to change the panel. |
| Quality Requirements | ---------- |

## *4.2. Dynamic Models*

### *4.2.1 Sequence Diagrams*

**Scenario 1:**

Ali double clicks the game icon in the desktop and open the game. He sees the Main Menu and clicks the High Score button and looks at the high scores. Then, in order to play a single player game, he selects Play Game from Main Menu and Single Player from Mode Select menu and starts a new game.

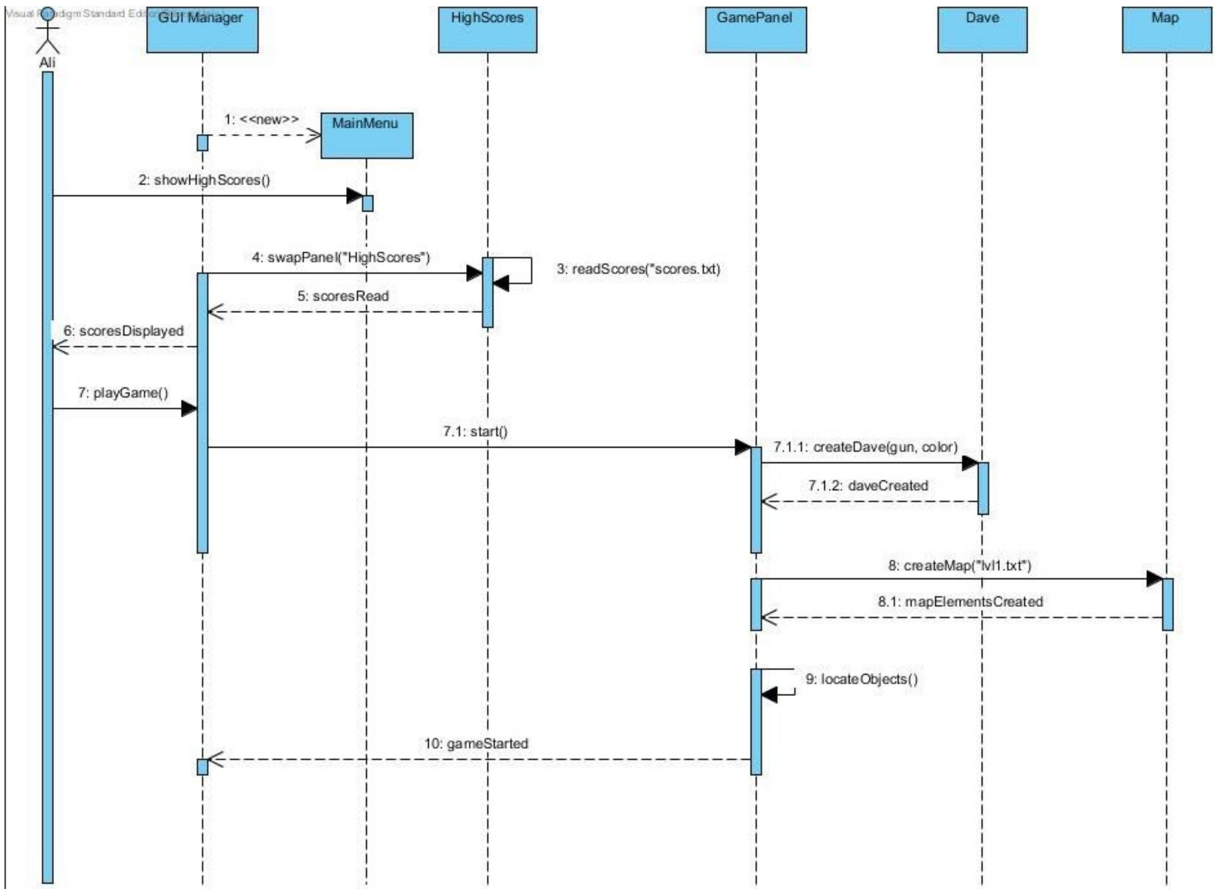
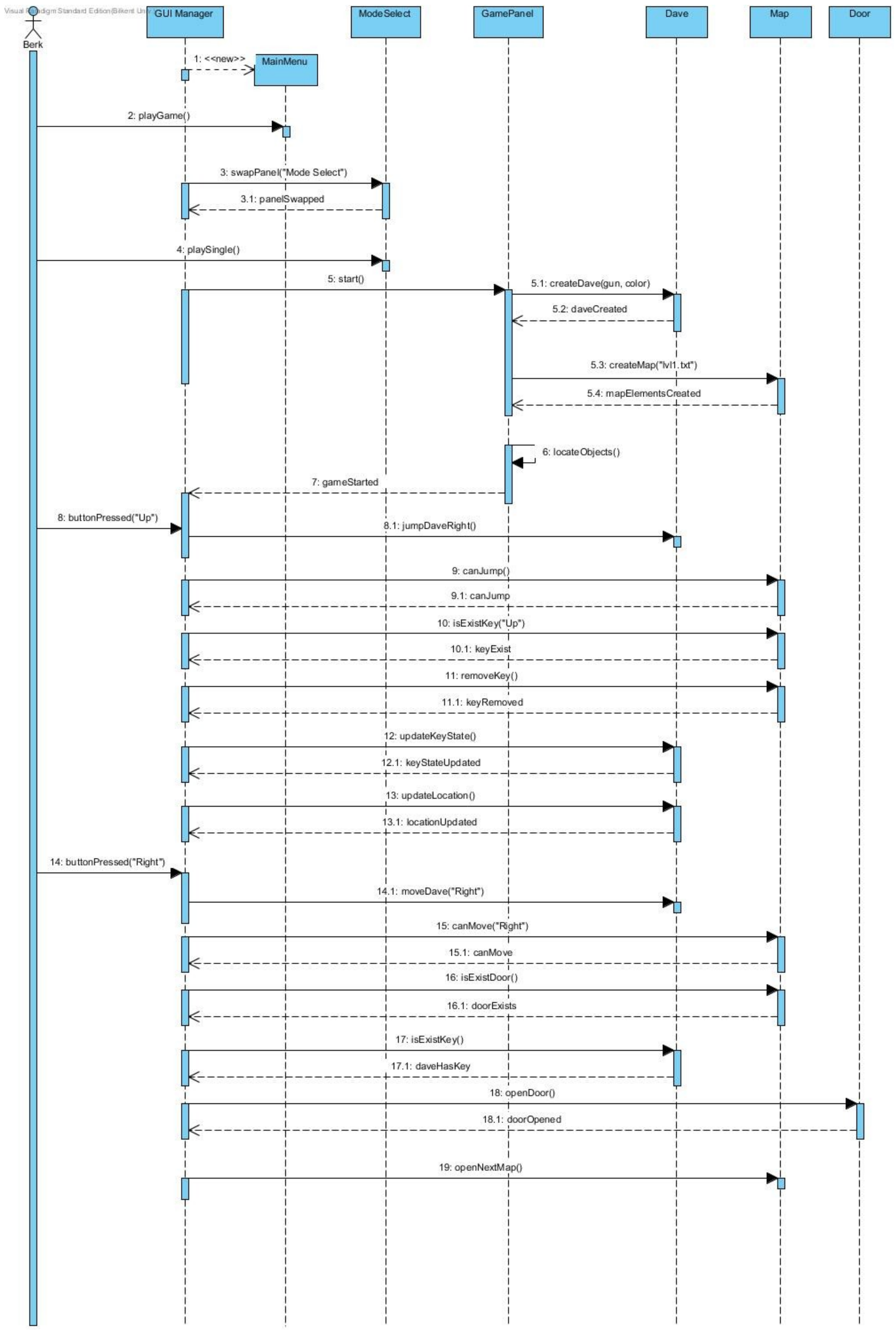


Figure 1) Sequence diagram for the use case described below.

GUI Manager handles to swap panels (Main menu, Settings, High Scores etc.). Therefore, when Ali executes the game, GUI Manager opens Main Menu which has “Play Game”, “High Scores” and “Settings” buttons. When Ali wants to see High Scores and clicks it, GUI Manager swaps panel and opens High Scores, then High Scores objects reads scores from a text file. After that, when Ali wants to play a game and clicks it, GUI Manager swaps the panel to Mode Select which has two options (“Single Player”, “Multiplayer”). Ali chooses Single Player and GUI Manager tells Game Panel to start a new game. Game Panel creates Dave object and first map of the game from “lvl1.txt” file. Game becomes ready to play.

**Scenario 2:**

Berk starts a new Single Player Game by choosing Play Game from Main Menu and Single Player from Mode Select menu. He jumps Dave and gets key. Then, he moves right. Since he gets the key and there is a door, door is opened and he completes that level successfully.



Visual Paradigm Standard Edition(Bilkent Univ
Berk
GUI Manager
ModeSelect
GamePanel
Dave
Map
Door
MainMenu
1: <<new>>
2: playGame()
3: swapPanel("Mode Select")
3.1: panelSwapped
4: playSingle()
5: start()
5.1: createDave(gun, color)
5.2: daveCreated
5.3: createMap("lvl1.txt")
5.4: mapElementsCreated
6: locateObjects()
7: gameStarted
8: buttonPressed("Up")
8.1: jumpDaveRight()
9: canJump()
9.1: canJump
10: isExistKey("Up")
10.1: keyExist
11: removeKey()
11.1: keyRemoved
12: updateKeyState()
12.1: keyStateUpdated
13: updateLocation()
13.1: locationUpdated
14: buttonPressed("Right")
14.1: moveDave("Right")
15: canMove("Right")
15.1: canMove
16: isExistDoor()
16.1: doorExists
17: isExistKey()
17.1: daveHasKey
18: openDoor()
18.1: doorOpened
19: openNextMap()

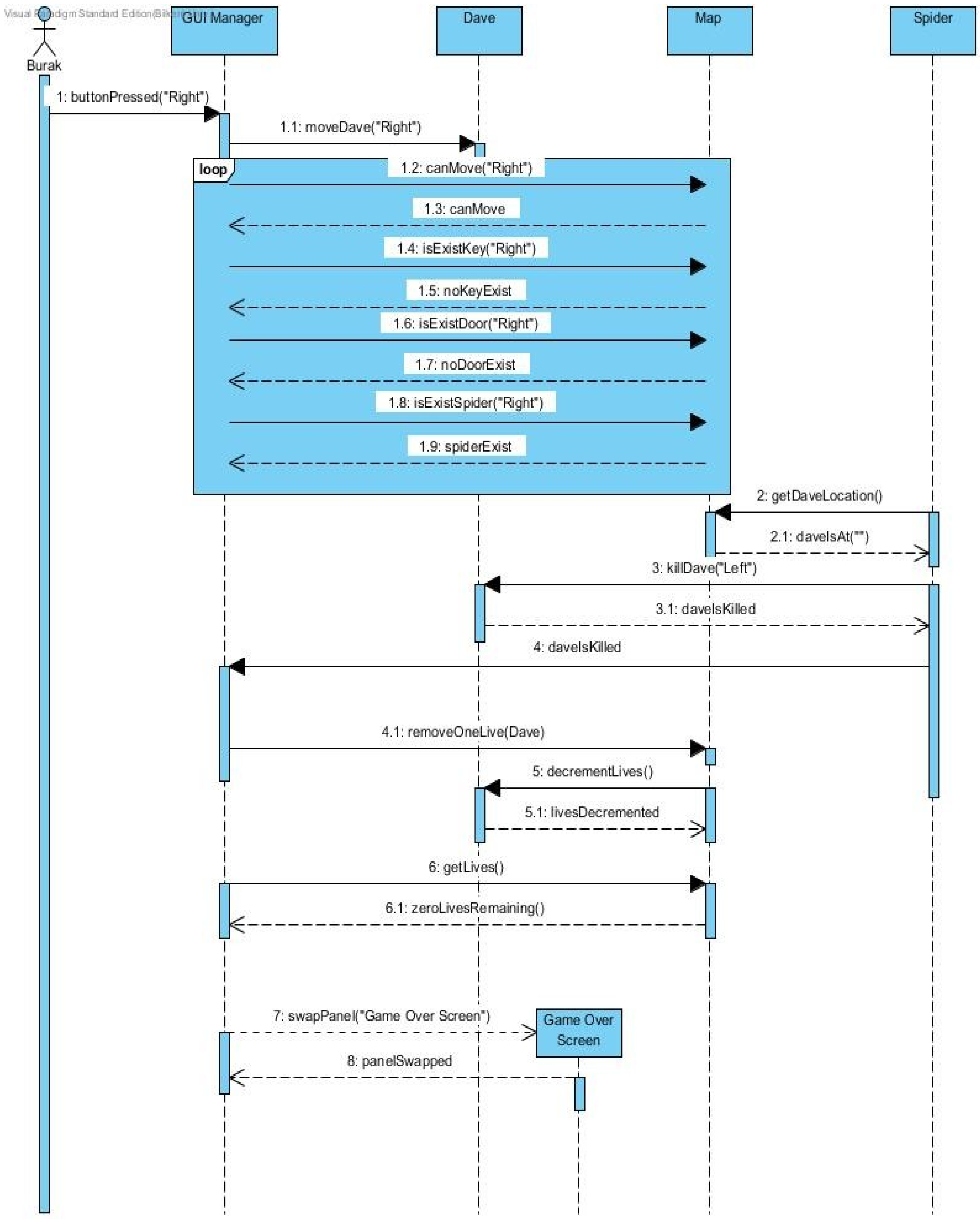
Figure 2) Sequence diagram for the use case describe below

As it is described in the description of first scenario, game is created started from GUI Manager. When Berk presses Up button and wants to jumps, GUI Manager checks from map that if he can jump or not, key exists or not and monsters or waters. In this scenario, there exists a key, therefore key is removed from map and the key state and the location of Dave is updated. After that, when Berk presses Right button, again GUI Manager checks everything from map. It is realized that there is a door, GUI Manager also checks that Dave has got the key or not. Since Berk has already got the key, door is opened and passes that level.

**Scenario 3:**

Burak plays a Dave game and has last one chance (life). Then, he wants to move right where there is a spider. Spider kills Dave and since he had only one life, game over screen displayed and game over.

In this scenario, Player Burak is already in the game and has his only one life. When he presses Right, GUI Manager checks everything from map again. When map tells GUI Manager that there exists a spider, spider gets Dave's location via getDaveLocation() method. Spider kills Dave and sends a message to GUI Manager. When Dave is killed, his total life is decremented by one. At the end, GUI Manager checks that if he has zero life or more. In this scenario, since he had no life, GUI Manager swaps the panel to Game Over Screen and game over.


Visual Paradigm Standard Edition(Bilkent
Burak
GUI Manager
Dave
Map
Spider
1: buttonPressed("Right")
1.1: moveDave("Right")
loop
1.2: canMove("Right")
1.3: canMove
1.4: isExistKey("Right")
1.5: noKeyExist
1.6: isExistDoor("Right")
1.7: noDoorExist
1.8: isExistSpider("Right")
1.9: spiderExist
2: getDaveLocation()
2.1: daveIsAt("")
3: killDave("Left")
3.1: daveIsKilled
4: daveIsKilled
4.1: removeOneLive(Dave)
5: decrementLives()
5.1: livesDecremented
6: getLives()
6.1: zeroLivesRemaining()
7: swapPanel("Game Over Screen")
Game Over Screen
8: panelSwapped

### *4.2.2 Activity Diagram*

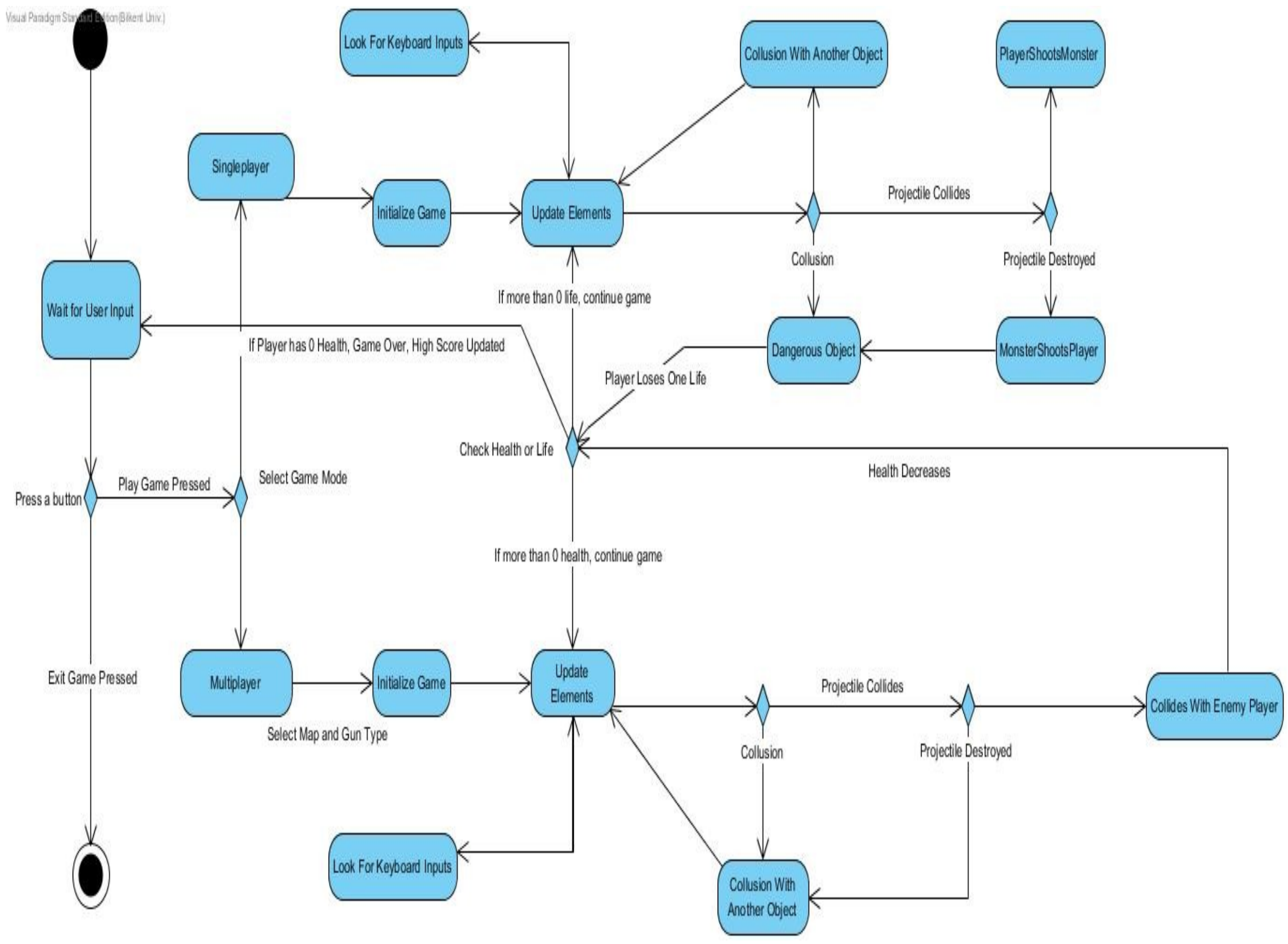


***Figure 4 – The Activity Diagram of Play Game Mode***

Before starting the game, users have to click the desktop icon and run the game. After running the game, the menu appears in first. There will be 3 options that user can prefer Play Game, High Scores, Help. Play Game option provides user to choose game mode. Game mode consists of two parts “Single and Multiplayer”. This option identifies the future of the game. If player prefers Single mode it starts directly from first level of the game because game machine adjusts the map and Dave’s objects (gun, color) automatically for the user. However, in the Multiplayer mode game gives chance for selecting map, guns and colors of the Dave’s.

After selecting the game mode, the game is started. All elements includes Dave, map’s objects is updated by the game in every action happened by the player. For example, Dave includes health, location properties. Because of the movements such as fighting with the monsters or falling into fire needs to refresh the health and location of Dave. Also, after user capture the key, game needs to refresh itself (containers) again.

***T***he flow of the game is mostly same as the above. But there is an important element that needs to explain. It is collusion. Hence, game needs to be fluent so collusions have to be fast. To achieve this aim, game makes updates after every movement and it has function that identifies the collusions. In singleplayer, the user can lose its life points via the terrain. In some places of the maps, there will be fires or cliffs and when Dave collides with those terrains and this will be a cause for losing life points. Also, when an enemy projectile collides will Dave, he will lose another life point and if Dave's projectile hits an enemy, the enemy will die and game elements will update themselves. There are less important collusions like Dave's walk through the terrain, or any projectile that hits a wall. For multiplayer, it is similar to singleplayer but both Dave's have health points and projectiles will diminish their health not instantly, but slowly based on the type of weapon.

## *4.3. Object and Class Model*

Note: The class diagram was too big and would not fit here. Therefore we included it in the github directory with the name 'class_diagram.jpg'.

(Link to the class diagram:
https://github.com/aliatli/davesson/blob/master/class_diagram.jpg)

Class diagram of the program is shown in the link. As for our estimates, the game will have more than 20 classes.

'Game' class will have the main method. It will create an instance of GamePanel class which is responsible for setting the size of the window and the fps. GamePanel initialization will create an instance of GUIManager class. Every time the GamePanel class updates its state by calling the update() function, the call will propagate through to the classes down in the hierarchy. For example, a call to draw() method will initiate a call to the draw() method of the GUIManager, which will also provoke other methods that does drawing in the classes that represents the maps such as Level1, Level2, Level3 and etc. These map classes will have objects representing the elements in the game, such as tiles, water, fire, Dave which also have draw() method.

The classes that represent the JPanel shown to the user are extended from an abstract class, GameState. The class will have a protected method called createMap(string) which will create the map by reading from a text file that specifies the positions of the MapObjects.

MapObject is an abstract class as well, representing the elements in the game such as Water, Fire, Dave, Gun, Bullet, Key, Door, RedDiamond, Vine, Crown, RedBush and etc. For instance in a  case that involves a Key object which is a requirement for  a Dave object to pass through a Door object, all the objects mentioned will be extended from a MapObject. We have chosen to implement the elements as an extension to MapObject, so as to store the elements in a MapObejct[] array and check for collisions in a much simpler way. More specifically, method checkTileMapCollision(MapObject, MapObject) will check if the two instances are located at the same position or if the boundary's of them are colliding. This explains why the MapObject has attributes to represent the rectangle that circumscribes the boundary's of MapObjects instances.

The classes that represent maps are also extended from an abstract class called GameState. GUIManager will command which map to create by invoking createMap(string) method with respect to the input from the user. This class has lots of instances of MapObjects. The diagram omits some of the relations between the MapObjects and GameStates, as there are lots of MapObjects and lots of GameStates, the arrows make it hard to comprehend. For example, Level2 is a class extended from GameState, which has a Crown object in it, yet we omitted it. This applies to other GameState objects as well.

This diagram's making took a lot of reviews of Java platform game codes.  The patterns across the different games then merged into creating the diagram.

## *5. Conclusion*

In conclusion, this report in its first part describes the requirement specification and second part is about our system model.

We believe that the cases we covered for specification part are addressed by the later system design discussions. We tried to build the models so as to be able to add new functionalities whenever needed.

System model involves use case models, dynamic models such as sequence diagrams and activity diagrams and class and object diagrams. The mock-up of the user interface part of the project is discussed in the requirements part to describe the minimal functionality of the system. The dynamic models are created so as to express much of the functionality by using only few cases. The sequence diagrams, for instance, are, therefore, very indicative about the inner workings of the system. Activity diagram acts as an intermediary between our use cases and class diagrams. Finally our class diagram lays out our class level solution that will help us meet our requirements within the Object-Oriented framework. We think that using intricacies of Object-Oriented Programming helped us to resolve otherwise very complicated relations between objects. For instance, abstract classes helped us to create an abstraction that is open to high level ideas and easily convertible to technical details. We believe that the solution we propose in this report will be of great importance when we start the implementation process.