Bilkent University

Department of Computer Engineering

Object-Oriented Software Engineering Project

CS 319 Project: Rush Hour

Design Report

Project Group: Quintuple Whopper

Group Members: Doğa Acar, Tunç Zerener, Ahmet Atahan Mutlu, Sarp Tekin, Ahmet Emre Zengin

Course Instructor: Eray Tüzün

December 12, 2018

Design Report

CS 319 Project : Rush Hour

1. Introduction

This design report has been made to clarify the gap between our problem and our solution. We will be referring about our purpose, our design goals, our system's architecture and its subsystem services.

1.1 Purpose of the System

Rush Hour which we are working on is a computer game that is similar to classic "Rush Hour" puzzle game which was invented by Nobuyuki Yoshigahara in 1970s. [1] Our Rush Hour game has the exact same play style with the classic "Rush Hour" puzzle game. However, our game offers more features than the physical Rush Hour game. Our game differs from the classic "Rush Hour" game by having sounds, background music and special levels which have a story behind it. Our purpose is to make a more dynamic and fun game with more content. Our digital game is designed for anyone who enjoys to playing board games, and our purpose is to offer them a more exciting experience.

1.2 Design Goals

There are many non-functional requirements and these shape our design goals. The most crucial ones of our design goals are mentioned below.

End User Criteria:

Ease of Learning: Since our game is designed for diverse group of age, ease of learning is important to play the game correctly by every person from different ages. In order to achieve ease of learning goal, our game has "How to Play" button which explains the game's basics. Moreover, it has a tutorial which is a simple level, tutorial shows how to play it in detail.

User-Friendliness: To attract and encourage users to play the game, our game has many graphical features. All cars in the game and the user interface have a color. Also, by adding background music and various sounds such as car sounds we aim to make the game more enjoyable. Furthermore, special levels and time challenges aim to eliminate the monotony and lack of variety from the game.

Maintenance Criteria:

Reliability: System should be purified from possible critical bugs and system crash problems. In order to achieve that goal, many testings will be done.

Performance Criteria:

Efficiency: Since our game has a time counter there should not be any delay when the user plays it. Therefore, the program won't consume more than 20 MB of device's random access memory. The system is able to do this by not using big pictures such as high quality pictures. It will use picture formats like Joint Photographic Experts Group (.JPG).

Response Time: The system of Rush Hour uses little memory as explained in the efficiency part to run the program efficiently. This helps program to run fast and have small latency.

2. High-Level Software Architecture

In this section, decomposition of the subsystems, mapping of hardware/software, persistent data management, access control and security, and boundary conditions will be explained. Purpose here is to explain the high-level architecture that will be used for the development of the game.

2.1 Subsystem Decomposition

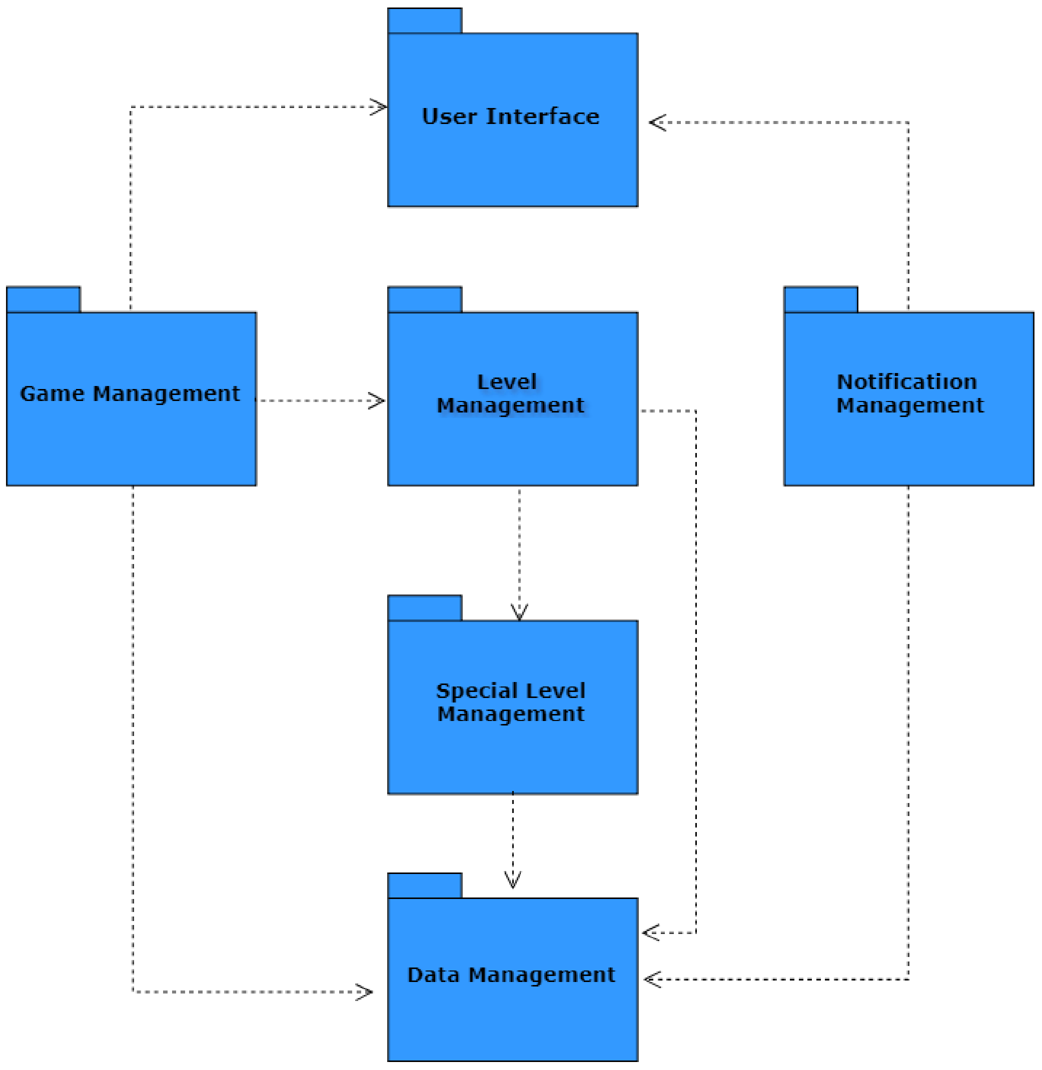

Figure 1 - Subsystem Decomposition

Our design will contain 6 main components that are designed for various features of the system. These are User Interface, Game Management, Level Management, Notification Management, Special Level Management and Data Management as seen in Figure 1.

User Interface will contain GUI materials that are used to display menu, credits, level selection, play view, how to play and tutorial. Data management will store all the model objects,interactions with the database, level objects and notification objects. Level management will contain the map for the game. It will be used to create, update and remove levels. Special level management will do the same thing with level management but special levels will contain extra features like time and move limit. Game management will contain soundtrack and the game. Notification subsystem which has an active connection with the database will show info, alerts or instructions about the game to the user.

2.2 Hardware/Software Mapping

Our game will be developed using Java. Only requirement for the software is Java Runtime Environment. In terms of hardware only a mouse will be required to interact with game menu and play the game by dragging the vehicles. For system requirements, Java Runtime Environment requires Pentium 2 266 MHz processor at minimum. Minimum memory requirement is 128 MB.[2] In terms of graphics Java games do not require advanced GPUs so the game will be appropriate for most of the computer systems.

2.3 Persistent Data Management

The data of the in-game features will be stored in txt files. The user can access the data from the location of the txt files, however, these data can only be changed indirectly by playing the game. No direct access to data will be permitted and any attempt to change or corrupt data will return these data to factory settings.

2.4 Access Control and Security

To play the game user needs to download our application to a computer. Access is straight to the point and there are no special requirements like login/password system to initialize the game. Since this is a computer game security is not a main issue for the system. However there will be a secure system to not give an open door to cheating. For example, if a player could progress without completing previous levels it would lead to an unfair and meaningless game. Therefore in the coding process any chance of cheating both in-game and at level screen will be eliminated.

2.5 Boundary Conditions

Initialization

Rush Hour application can be initialized by clicking the shortcut of the game after downloading the game files to any computer that has Java Runtime Environment. The game can be started by pressing the “PLAY” button in the main screen. The application will be an executable .jar file and no further installation will be needed once the game is downloaded. Rush Hour is an offline game so internet connection won’t be needed when the game is initialized.

Error

The game will give an error if a foreign system(Ex:Cheat Engine) tries to change or corrupt its files. The game will be terminated and all saved data will be deleted immediately.

Termination

Rush hour can be closed by clicking the exit button “X” on the top right side of the window screen. If the player presses the exit button a pop-up window will ask to confirm the termination. The termination of the game won’t result in data loss since once a level is completed all the data is saved to the save file of the game.

3. Subsystem Services

This section contains subsystem services. There are 6 subsystem in the project which are Game Management, Level Management, Special Level Management, Notification Management, User Interface and Data Management.

3.1 Game Management

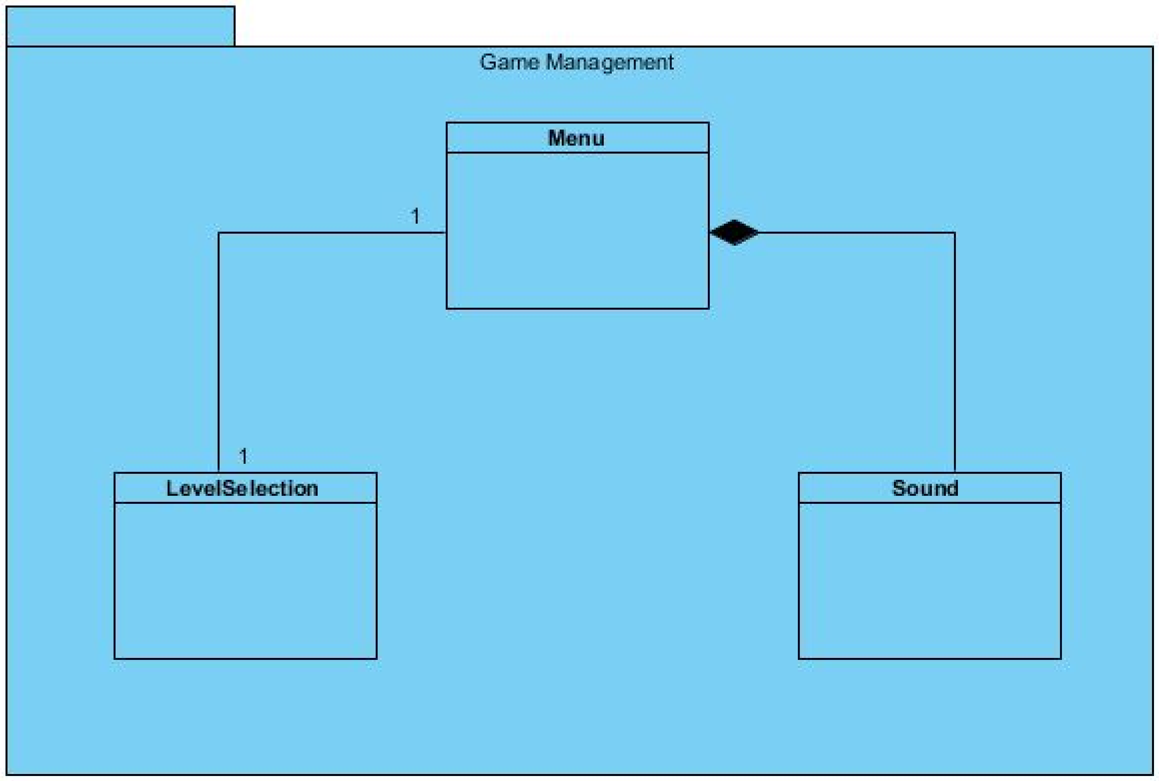

Figure 2 - Game Management Subsystem

Game management has classes to create the game and add soundtrack. It has connection with Level Management, Data Management and User Interface. It will add the created levels from Level Management to the game, use graphics from the user interface to make them all visual. Also it will add soundtrack to the game and create changes in Data Management to keep track of the data.

3.2 Level Management

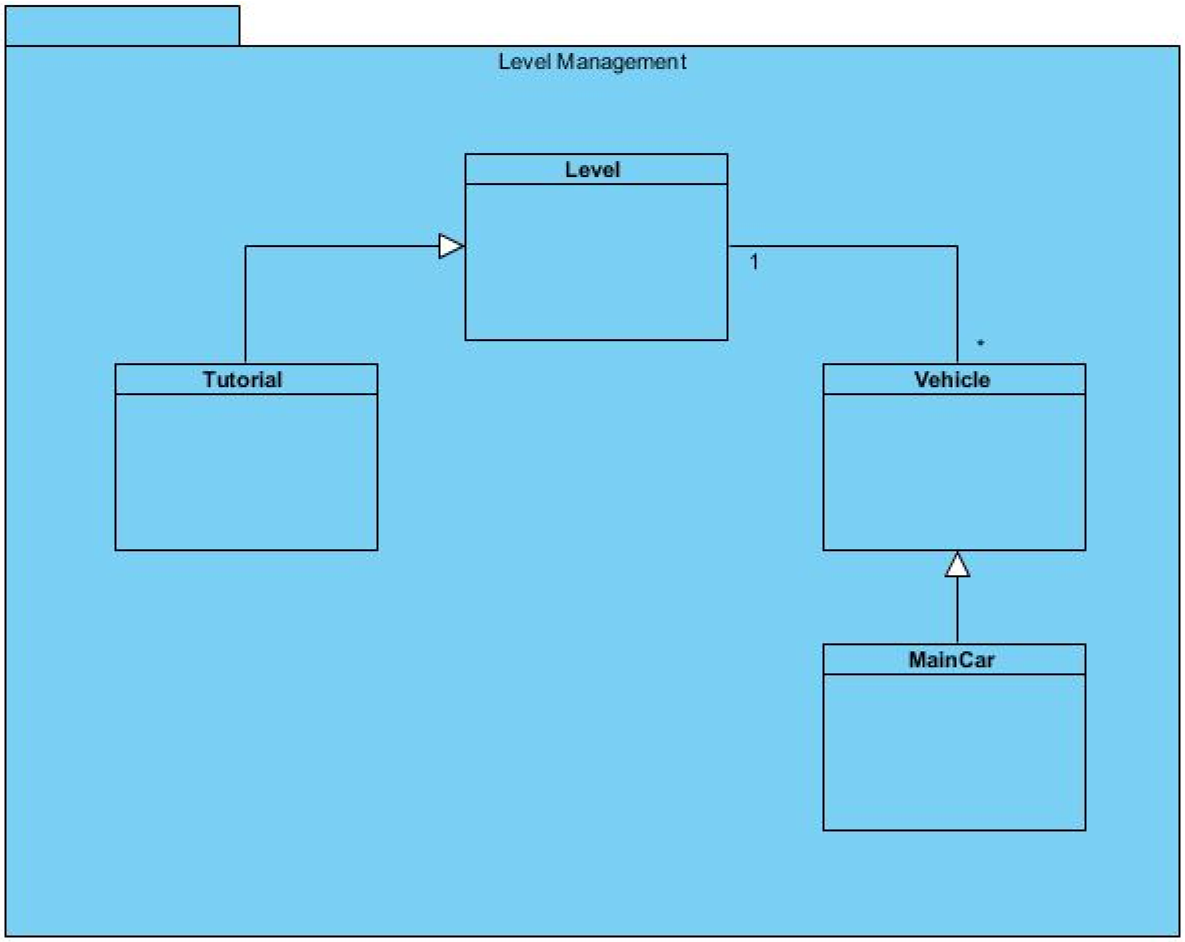

Figure 3 - Level Management Subsystem

Level Management has classes to create, update and remove levels. It will create the map, vehicles and integrate them to each other, creating the level. Also it will have a class to create tutorials for instructive purposes as well. It will give hints on how to play the game. It will have a connection to Data management as well to keep track of the data. So that the players don't lose any progress they made throughout the game.

3.3 Special Level Management

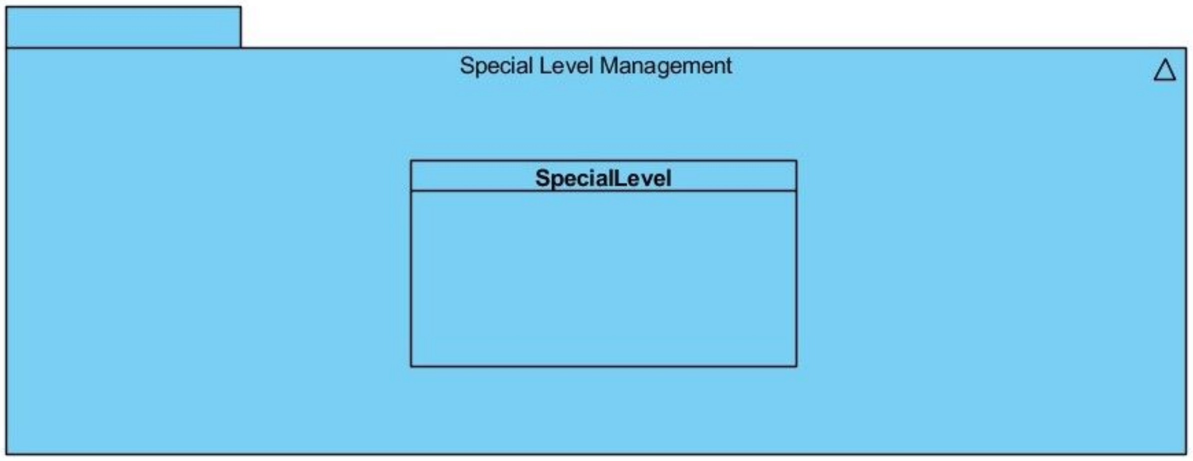

Figure 4 - Special Level Management Subsystem

Special Level Management and Level Management subsystems are connected to each other. When creating a new special level, Level Management subsystem will be used. This subsystem will do some alterations to created level and add some extra features to it like having a move or time limit to beat the level. With its connection to the data management it will also be able to keeps data changes in check.

3.4 Notification Management

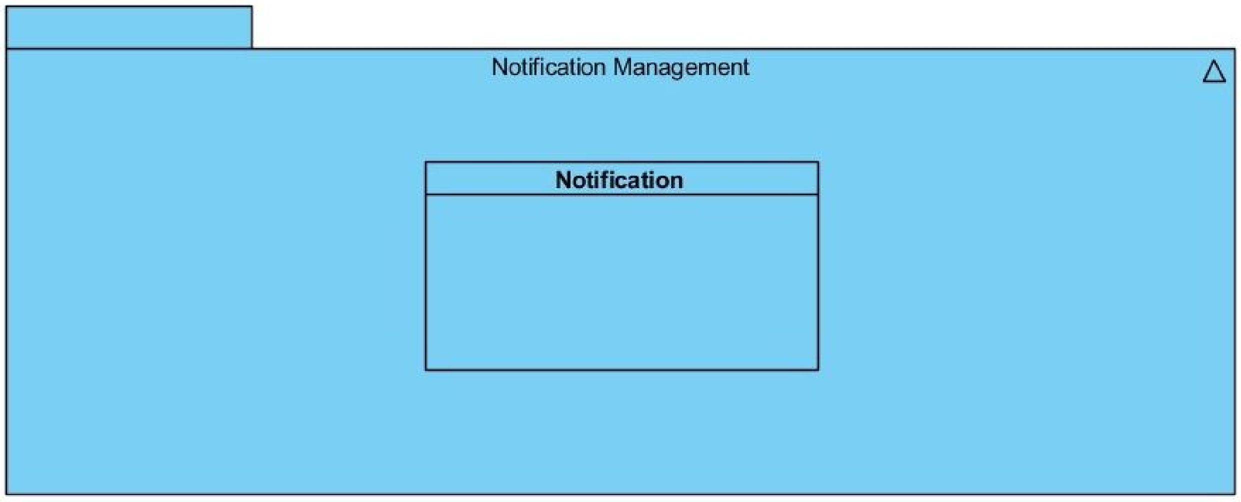

Figure 5 - Notification Management Subsystem

This subsystem will get information about the game from the Data Management and display them to the user with pop-ups including the information. The information provided will be about getting a high-score, passing a level etc.

3.5 User Interface

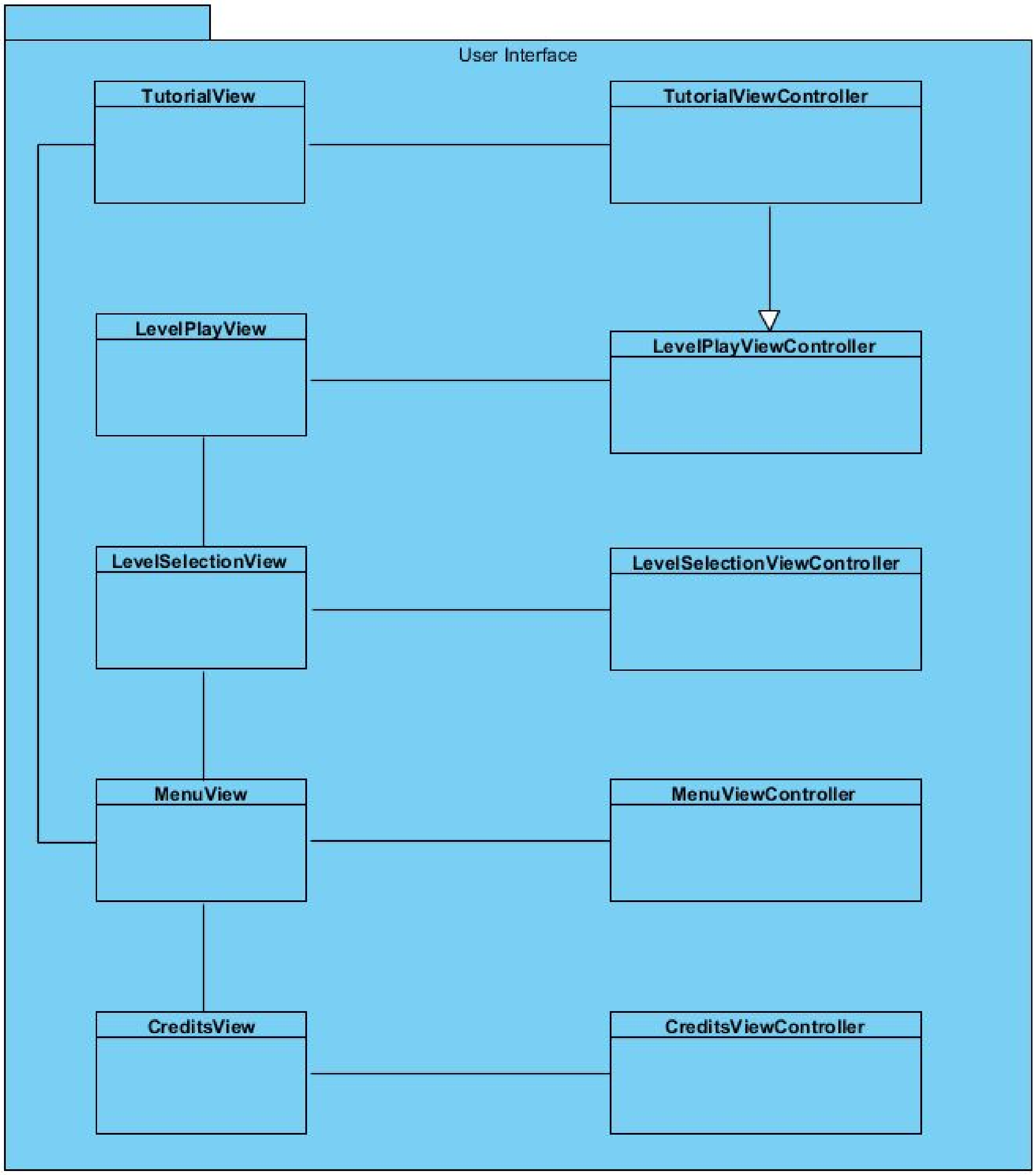

Figure 6 - User Interface Management Subsystem

This subsystem will include the graphics for the program. When the Game Management initializes the game it will get the according graphics from the User Interface and Create a game that can be seen by the user. Also pop-up balloons and animations will be included in this subsystem as well. User interactions in each screen are done with their respective controller classes.

3.6 Data Management

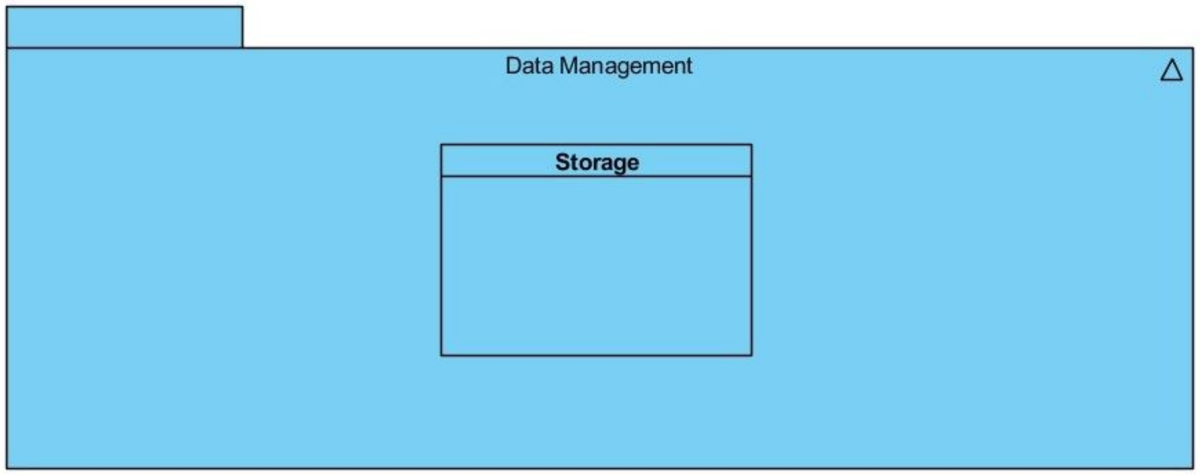

Figure 7 - Data Management Subsystem

This subsystem will be the storage of the program. It will contain information of high scores, game progress and created levels. It will have connection to all of the other subsystems. Others systems will have the ability to change the data stored in this subsystem.

4. Low-Level Design

4.1 Object Design Trade-Offs

Rapid Development vs Functionality: Since we have limited time, rapid development has higher priority in our design. To reduce time for development, instead of other programming languages, Java is prefered for implementation, as it is the programming language that all group members are familiar with. It is decided that too many functionality might create hardship for users, if user friendliness and ease of learning are considered. By this reason, Rush Hour has limited features but our main goal is implement these features in a better way. In addition, the limited time prohibits us from working on numerous features.

4.2 Final Object Design

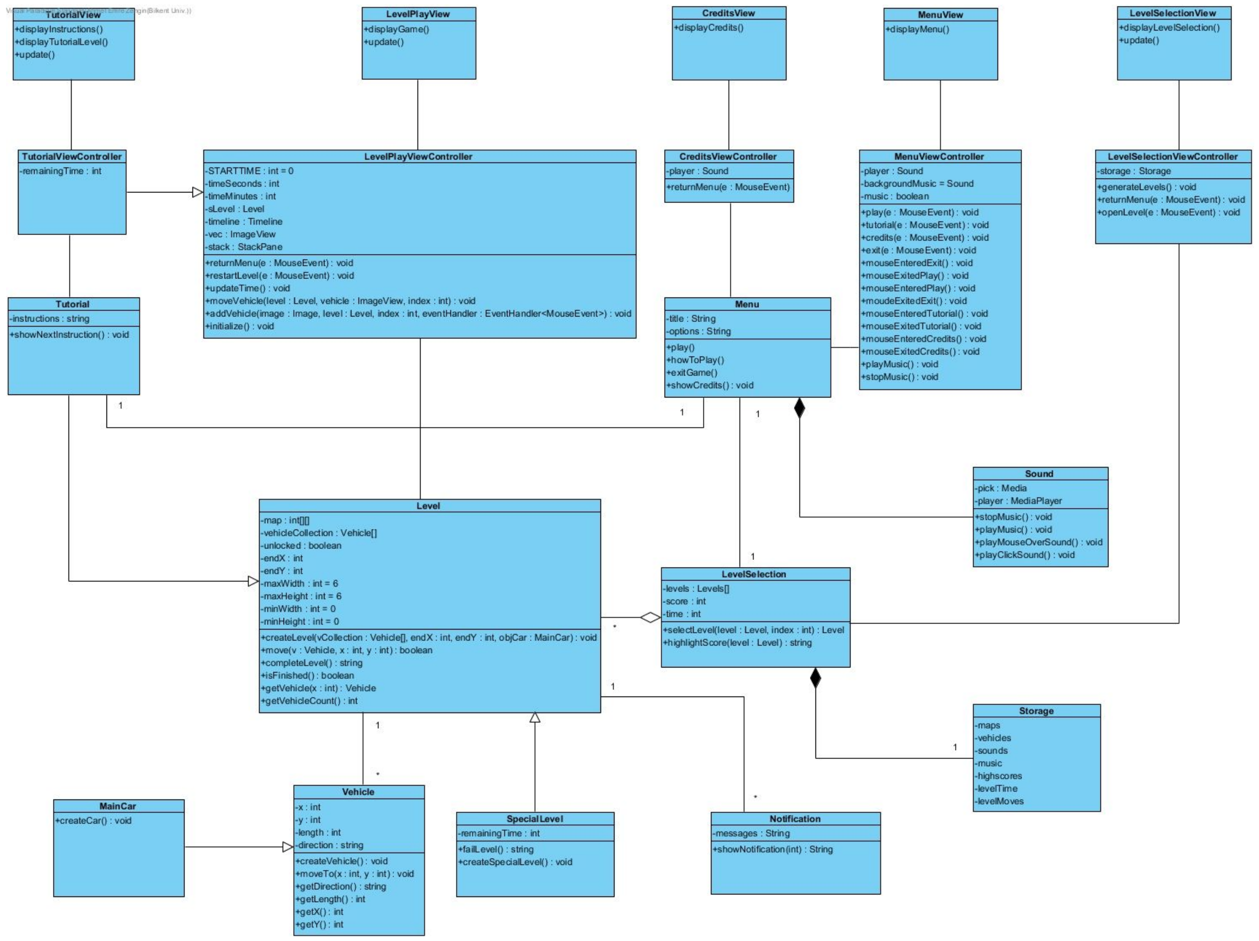

Figure 8 - Final Object Design

4.2.1 Game Management

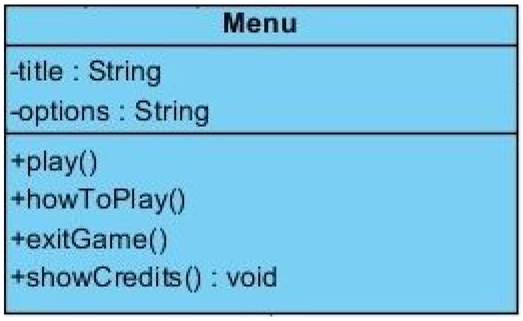

Figure 9 - Menu Class

The menu class is basically the first screen shown to the player once the game is executed. It has a collection of options and a title to be shown to the player. The ‘play()’ function takes the player to the level selection screen. The ‘howToPlay()’ function generates a tutorial level and starts it. The credits are shown by the ’showCredits()’ function. Lastly, the ‘exitGame()’ function terminates the program.

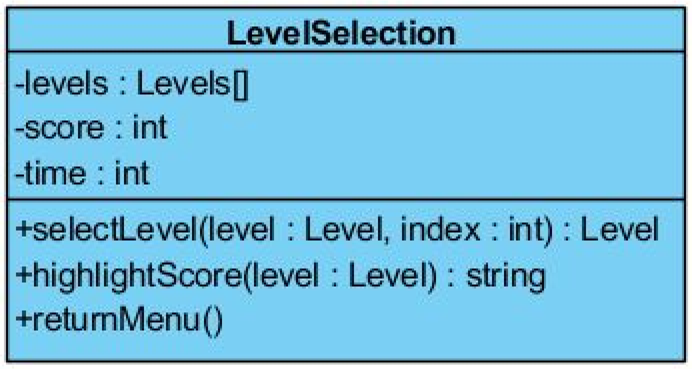
LevelSelection
-levels : Levels[]
-score : int
-time : int
+selectLevel(level : Level, index : int) : Level
+highlightScore(level : Level) : string
+returnMenu()

Figure 10 - LevelSelection Class

The LevelSelection class keeps a collection of levels as a ‘Level’ array. The selected level is opened once the ‘selectLevel()’ function is called. The ‘time’ attribute starts increasing and a ‘score’ is calculated depending on it. When the player finishes a level the score is shown to the player with the ‘’highlightScore()’ function.

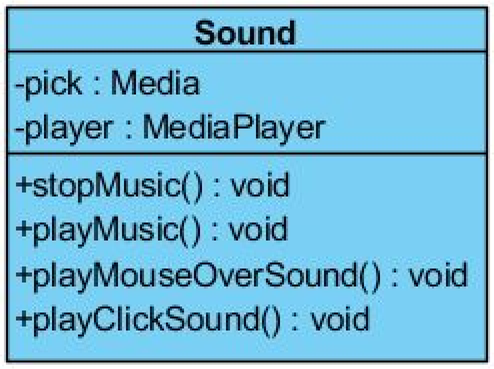

Figure 11 - Sound Class

The sound class contains the background music and sound effects that will be inside the game. It has 2 functions called ‘turnMusic()’ and ‘turnSound()’ to control these functions given proper parameters.

4.2.2 Level Management

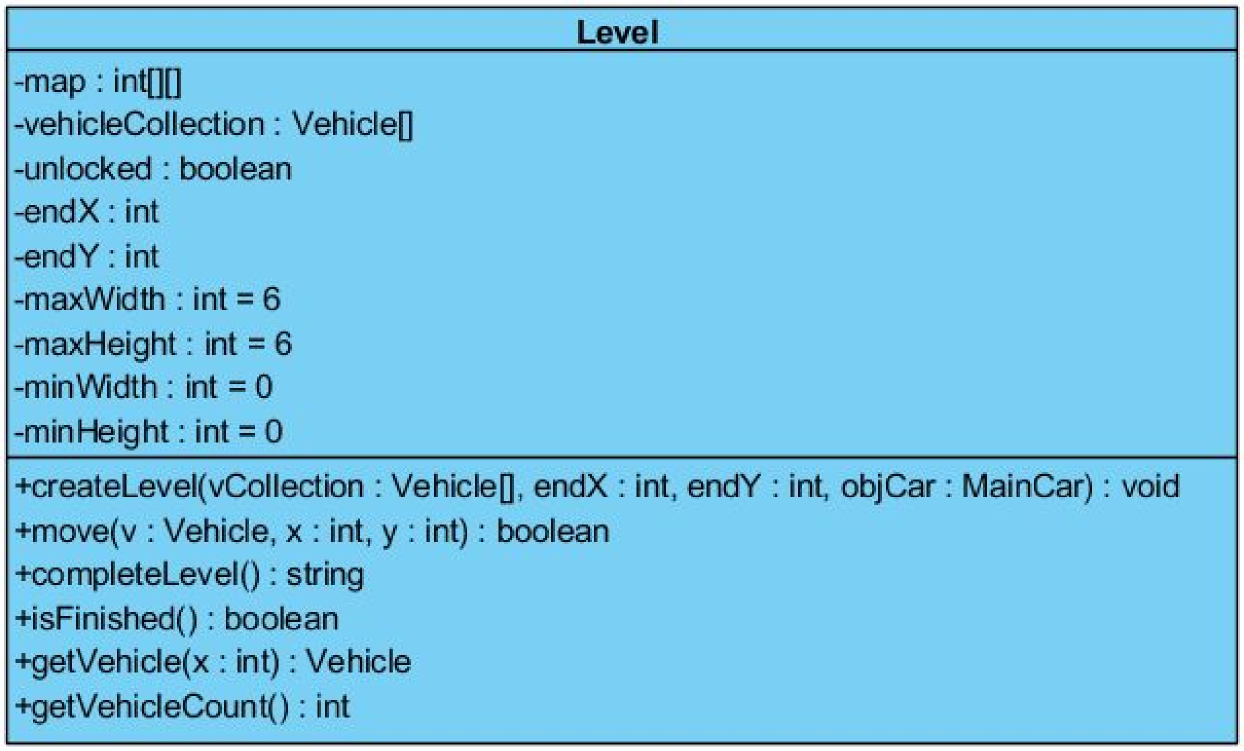
Level
-map : int[][]
-vehicleCollection : Vehicle[]
-unlocked : boolean
-endX : int
-endY : int
-maxWidth : int = 6
-maxHeight : int = 6
-minWidth : int = 0
-minHeight : int = 0
+createLevel(vCollection : Vehicle[], endX : int, endY : int, objCar : MainCar) : void
+move(v : Vehicle, x : int, y : int) : boolean
+completeLevel() : string
+isFinished() : boolean
+getVehicle(x : int) : Vehicle
+getVehicleCount() : int

Figure 12 - Level Class

The level class contains the game grid as a two dimensional array called ‘map’, a collection of ‘Vehicle’ objects, a boolean attribute called ‘unlocked’ that will be used to distinguish locked and unlocked levels and two integers called ‘endX’ and ‘endY’ which specify the location of end point of the game. Lastly four integers called ‘maxWidth’, ‘maxHeight’, ‘minWidth’, ‘minHeight’ indicates the game grid’s size. The ‘createLevel(Vehicle[] vCollection, int endX, int endY, MainCar objcar)’ creates a game field for each level given different size/kind of Vehicle objects and initializes the end point. The ‘move(Vehicle v, int x, int y)’ function indicates whether a vehicle can move to the desired place or not. If it can be moved, function moves the selected car and returns true. If the path of the Vehicle that we want to move is blocked, it returns false. ‘isFinished()’ checks whether the level is finished or not, returning true and false respectively. ’completeLevel()’ prints out the score of the player and congratulates them. ‘getVehicle(int x)’ gets the vehicle that wanted to be moved, this function is required to move the cars efficiently. ‘getVehicleCount()’ returns the number of vehicles on the game grid.

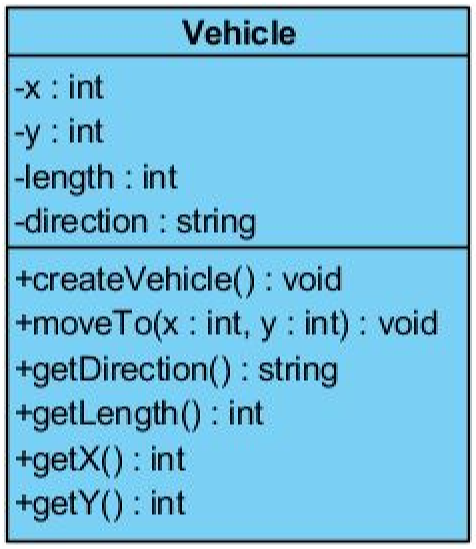

Figure 13 - Vehicle Class

All vehicles have one x and one y value that is used to specify their location. All vehicles can consist of either 2 or 3 units. Each vehicle can be either horizontal or vertical depending on their ‘direction’ attribute. The ‘createVehicle()’ function creates a Vehicle object according the given location, length and direction of a car. ‘moveTo()’ function moves a vehicle to a specified location. The ‘getDirection()’ function returns the direction of a vehicle. ‘getLength()’ function returns the length of a vehicle. ‘getX()’ function returns the x value of a vehicle. ‘getY()’ function returns the y value of a vehicle.

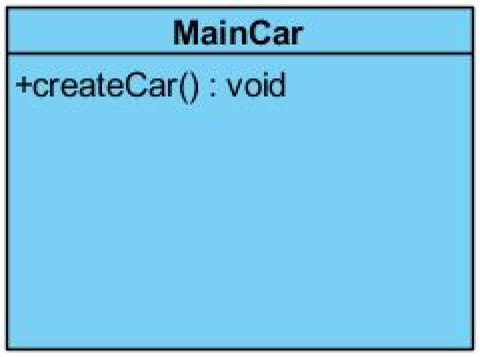

Figure 14 - MainCar Class

The MainCar class is basically the escape Vehicle which must reach the endpoint in order to complete the level.

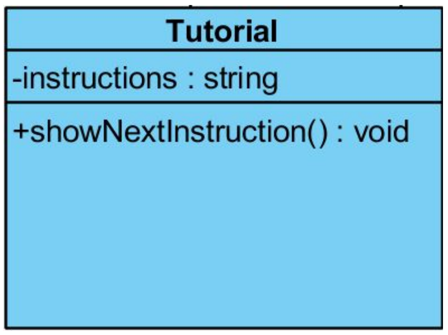

Figure 15 - Tutorial Class

The Tutorial Class contains a function that pops up the next instruction for the player, calling the ‘showNextInstruction()’ function.

4.2.3 Special Level Management

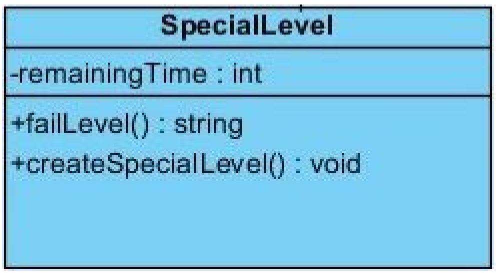

Figure 16 - SpecialLevel Class

Special Level class is inherited by Level class. Special levels have time challenges where a player will have a limited time to complete the level. Therefore, it has an extra attribute called remainingTime. ‘failLevel()’ function is called if the player runs out of time. This function informs the player with a text message. ‘createSpecialLevel()’ creates an instance of a special level.

4.2.4 Notification Management

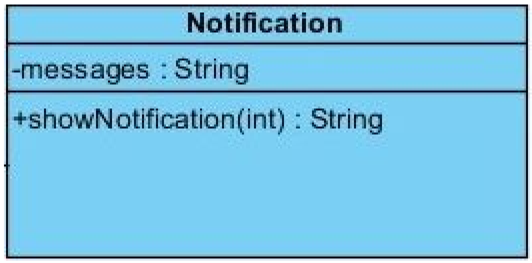

Figure 17 - Notification Class

The Notification class contains ‘messages’ that will be shown to the player by the ‘showNotification()’ function. These messages will include alerts and instructions which will be shown specifically when required.

4.2.5 Data Management

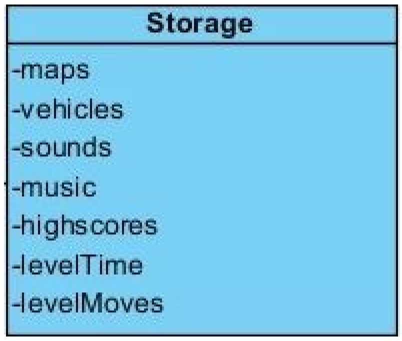

Figure 18 - Storage Class

The Storage class keeps every element inside a game level that is needed to be saved inside the local machine. These are maps, vehicles, sounds and music which are predetermined for each game level, and highscores, levelTime and levelMoves which are calculated with respect to player's score in each game level.

4.2.6 User Interface

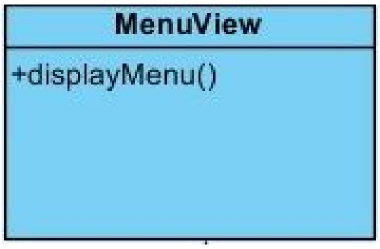

Figure 19 - MenuView Class

MenuView class contains a single method. 'displayMenu()' displays the main menu of the game which contains various options like play, tutorial, credits and exit.

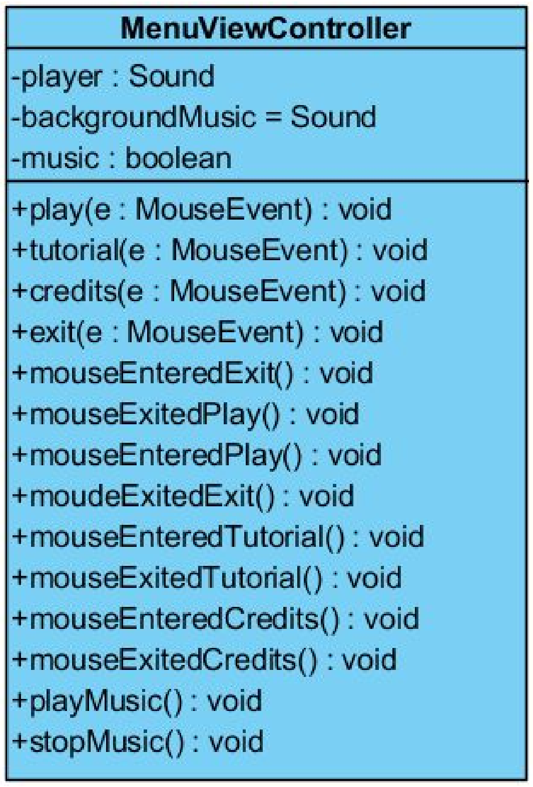
MenuViewController
-player : Sound
-backgroundMusic = Sound
-music : boolean
+play(e : MouseEvent) : void
+tutorial(e : MouseEvent) : void
+credits(e : MouseEvent) : void
+exit(e : MouseEvent) : void
+mouseEnteredExit() : void
+mouseExitedPlay() : void
+mouseEnteredPlay() : void
+moudeExitedExit() : void
+mouseEnteredTutorial() : void
+mouseExitedTutorial() : void
+mouseEnteredCredits() : void
+mouseExitedCredits() : void
+playMusic() : void
+stopMusic() : void

Figure 20 - MenuViewController

MenuViewController class contains 14 functions. ‘play(e: MouseEvent)’, ‘tutorial(e: MouseEvent)’ , ‘credits(e: MouseEvent)’ and ‘exit(e: MouseEvent)’ functions check the clicked option by the player. According to clicked option each function opens selected screen. ‘playMusic()’ function plays the music and ‘stopMusic()’ function stops the music. Rest of the mouse listeners are required for the click sounds of exit, play, tutorial and credits. MenuViewController also contains player for click sounds, backgroundMusic for the music of the game and music checks if the music is running.

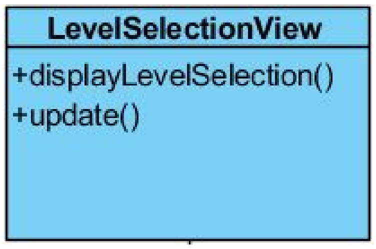

Figure 21 - LevelSelectionView Class

LevelSelectionView class has 2 functions. The ‘displayLevelSelection()’ shows all the levels in the game. ‘update()’ function updates the screen when a player unlocks a new level or breaks a new record for a specific level.

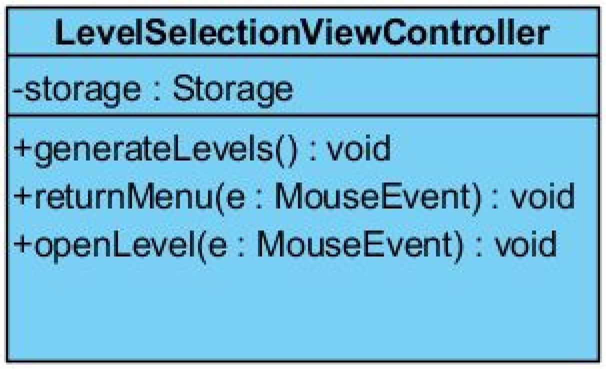
LevelSelectionViewController
-storage : Storage
+generateLevels() : void
+returnMenu(e : MouseEvent) : void
+openLevel(e : MouseEvent) : void

Figure 22 - LevelSelectionViewController

The LevelSelectionViewController class has 3 functions. generateLevels() method’s return type is void and it takes levels from storage and display them to the user. openLevel() methods takes a MouseEvent. When the user click the level on the screen, this method leads the user to the level. Lastly, returnMenu() function exist in order to go back to main menu.

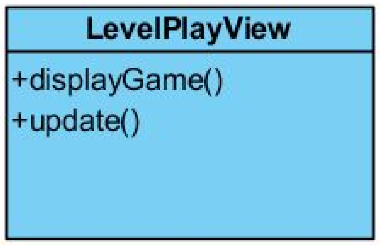

Figure 23 - LevelPlayView Class

The LevelLevelPlayView class contains 2 functions. ‘displayGame()’ method will show the playing screen after the player selects a level. ‘update()’ method will refresh the game screen after each move made by the player.

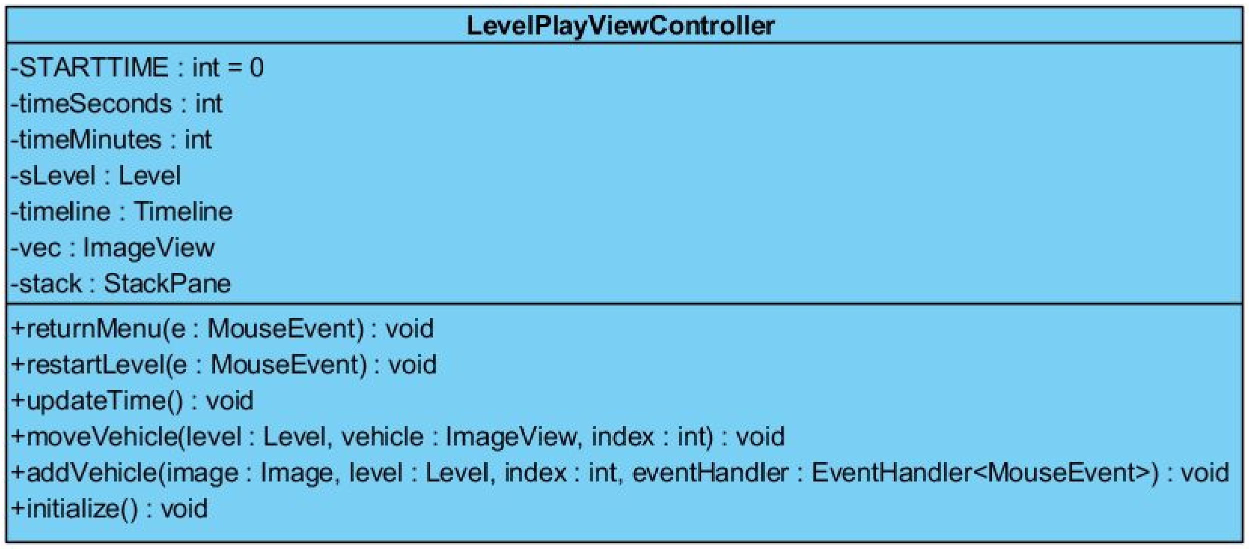
LevelPlayViewController
-STARTTIME : int = 0
-timeSeconds : int
-timeMinutes : int
-sLevel : Level
-timeline : Timeline
-vec : ImageView
-stack : StackPane
+returnMenu(e : MouseEvent) : void
+restartLevel(e : MouseEvent) : void
+updateTime() : void
+moveVehicle(level : Level, vehicle : ImageView, index : int) : void
+addVehicle(image : Image, level : Level, index : int, eventHandler : EventHandler<MouseEvent>) : void
+initialize() : void

Figure 24 - LevelPlayViewController

LevelPlayViewController class has 6 functions. ‘returnMenu(e: MouseEvent)’ function returns player to the menu screen. ‘restartLevel(e: MouseEvent)’ function restarts the level. ‘updateTime()’ function updates the time counter each second. ‘moveVehicle(level:Level,vehicle: ImageView, index:int) function moves the vehicle when its clicked. ‘addVehicle(image:Image, level:Level, index:int, eventHandler:EventHandler<MouseEvent>) function adds vehicles to the level. ‘initialize()’ creates the game screen with respect to level selected. LevelPLayViewController also has attributes like STARTTIME to start the timer at 0, and timeSeconds and timeMinutes to update the timer when seconds is 60. sLevel holds the selected level. The property timeline is used to create the timer. Vec is used to initialize images of each vehicle.

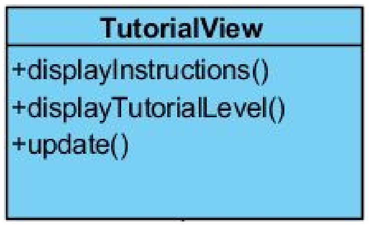

Figure 25 - TutorialView Class

The ‘TutorialView’ class contains 3 functions. The ‘displayTutorialLevel()’ function will display the tutorial level. When the tutorial level is opened, ‘displayInstructions()’ function will display the instructions for the tutorial level. The screen will be updated every time player moves to the next instruction by ‘update()’.

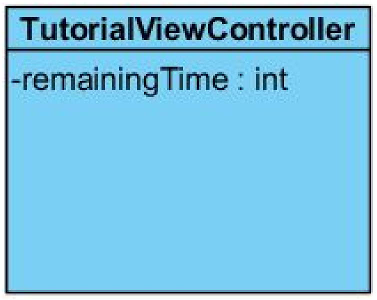

Figure 26 - TutorialViewController

The TutorialViewController has merely an integer as called remainingTime. It holds the remaining time to end of the tutorial.

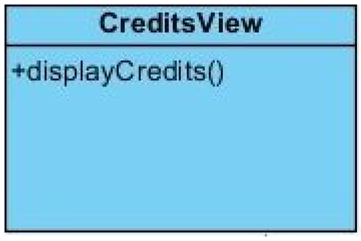

Figure 27 - CreditsView Class

The CreditsView class contains only one function. ‘displayCredits()’ function will show the credits of the game.

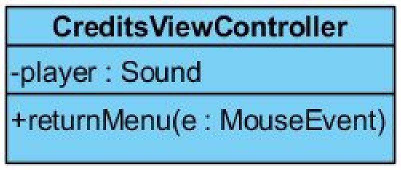

Figure 28 - CreditsViewController

The CreditsViewController class contains only one function. ‘displayCredits()’ function will show the credits of the game.

References

[1] "Rush Hour (Puzzle)", Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Rush_Hour_(puzzle). [Accessed 3 November 2018]

[2] "Windows System Requirements for JDK and JRE", . [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_system_requirements.html. [Accessed 5 November 2018]