CS 319 Object Oriented Software Engineering Final Report Iteration - 2

RUSH HOUR
GROUP 3.B

Yusuf Dalva
Kuluhan Binici
Musab Gelişgen
Melih Ünsal
Ahmet Avcı

1. Introduction

For the current implementation of the project, most of the functionalities of the game are up and working. Different from the Demo presented in the first iteration of the game, there are 8 different levels implemented for the singleplayer game and three levels for the multiplayer game. The functionalities offered in the analysis report are mostly implemented (some are discarded such as optional music, remaining time and more exciting features are brought such as hint, undo, etc). For the visual part of the game, some improvements in the flexibility of the interface are provided. Selecting the theme and the escape car is enabled to the user. For the visuality of the game, the assets used in the game are also revised.

Specific for the SinglePlayer game, the background image of the grid and the car that the user wants to use as the escape car are adjustable by using "Select Theme" and "Select Car" screens. In addition to the playability of the levels in the single-player mode, the player is now awarded stars only depending on the moves made to pass the level. The amount of stars given is between 1 and 3. The stars collected by a player is now stored locally (using serialization).

For the Multiplayer game, the wildcard system is implemented which determines the moves that can be made by the players. The turns of the players also occur in a sequence (turn-based). In the multiplayer mode, the first person to get his car out of the map wins. Just like the single player mode of the game, the cars can be moved in the current implementation and the three parts of the map can be shifted accordingly. The wildcard system is also another feature of the game. The type of the move that the player does is determined by the wildcard.

For the visuals used in the game, a considerable amount of revision has been made to contribute to the visual representation of the game. The cars used in the game are updated with assets that have higher resolution. New themes for the maps are brought. The theme is changeable now, which can be adjusted by using "Select Theme" option of the game. There are currently 4 different selections for the theme of the grid. The escape car now also has 4 different skins which can be adjusted by using "Select Car" theme of the game. The button styles are also updated to provide a better user experience.

2. Design Changes

After the first iteration of the game, by considering the logistics that the implementation of the game provides to us, the design of the game changed drastically. With the design patterns (both high and low level) learned through the course, the object design changed in a considerable amount. In the first proposal of the object design, the game aimed to follow a design pattern which is completely fitting the MVC design pattern which is not the case anymore (due to the complexities with the game engine we are using). The current design of the game benefitted from the Model-View-Controller (MVC) design pattern but it is not a complete reflection of the pattern. In the overall design of the application, with the hint functionality brought to the game, a new class for finding the shortest path is added.

For the scoring system of the game, the high score is now determined by the move count of the player and the optimal move count for that specific map. Then, the player is awarded stars as a result of this move count (from 1 to 3). For the UI components, the structure of the classes used in the implementation has changed (due to the game engine we are using) which resulted in the change in the structure of the User Interface package. Through the application, with the library used, the UI components used could be held in a container which the BaseScreen class provides. This container is now held inside the BaseGame class which is inherited by all of the user interface components. The relationship between the screens is now changed to be inheritance instead of composition.

In the Game Logic package of the application, the functionalities of the classes also have changed. The moving capabilities of the cars are controlled by the Car class located in this package. With the help of this class, when the user attempts to move the car the verification of the move is done by this class. Different than the design proposed in the first iteration, car class is not just the logical representation of the car with x and y coordinates, it also controls the moves that are done with the car. For the logical representation of the levels, the levels in the single player mode and the levels in the multiplayer mode are rearranged that they both inherit the Levels class which provides a general view about the levels with the UI information that it has and the logical information that it has. With respect to this information, Levels class also formed a relationship between the Game Logic package and the User Interface package. By this model of inheritance, the information that the classes about mode specific levels hold independently is reduced which made these classes much more meaningful and understandable.

As a new feature, in the single-player mode, there is a hint option provided which finds about the shortest path to solve the puzzle according to the current status of the user in the level. This path is found by the A* algorithm and implemented inside the Game Logic package. The algorithm works with the logical representation of the game grid and the cars on it, so it was a logical decision to include it inside the Game Logic package.

With respect to all of these decisions, the low-level design of the project is revised as well as the variable names by replacing them with more meaningful ones.

3. Lessons Learnt

One of the main problems in the first phase of the project was of course lack of communication and knowledge. Although we scheduled regular meetings with the team members, some content kept getting away from our attention and we forgot to mention some of our ideas with others. This resulted in two very different class diagrams from each other (the class diagram from the design report and the current project's class diagram). However, we did not let this divergence get ahead of us and completed the project with optimal conditions.

The second lesson we learned might be not being intimidated by the number of requirements we put on the analysis report because we realized that it is easy to complete many of the requirements in a couple of hours. We have already added some features that we did not mention about. Although we skipped on some small features (e.g. playing with a variety of music), the features we completed are far better than the ones we had put on our first requirements report (e.g. hint for single-player, undo for single-player, star system for single-player).

Another lesson we had learned is the importance of using a version control system. We had begun using GitHub as a place where we put our reports on, and then started sharing code within the group. However, the lack of experience and knowledge we had on this system sometimes had slowed us down because of conflicts while pushing, working on too many branches, working on same files simultaneously (which also shows us we had not designed our classes careful enough according to the S.O.L.I.D principles). The lesson to get from this example is that we should be notifying our teammates about the changes we are doing to the classes that might be related to/relevant with what they are doing. Another lesson is that we should keep every class so simple that no more than one person will be working on it at the same time (then, this will help us reduce conflicts and pushing/pulling problems in a branch)

4. User's Guide

This section contains information about the system requirements for playing Rush Hour at its best, installation of the game and a short user's guide to teach beginners how to play the game.

4.1 System requirements & installation

Recommended System Requirements:

- A Desktop Computer or Laptop (Windows 8 or higher / Linux / MacOS X.

- **128 MB** memory or higher.
- The computer screen size should support at least **640 x 480**
- An Intel processor with **1.80 GHz** or higher (preferably 4th generation or higher)

Installation:

The setup steps expect git to be installed on the system.

First, check out the repository. This process might expect you to have access to this private project, so make sure you are logged in to your GitHub account from your terminal.

~~~
```bash
git clone https://github.com/musabgelisgen/RushHour
```
~~~

Then, import this project to an IDE as an existing gradle project (preferably Eclipse since this is also our development environment).

Make sure you are using JavaSE-1.8. Then go to project properties and set the working directory of the project as

`${workspace_loc:RushHour-core/assets}`

Then, go to DesktopLauncher.java file and run the project.

p.s. These installation steps are just used for demonstration purposes. We plan on creating a .jar file to open game easily when the project is finished.

4.2 How to use

When the game launches, you see the main menu which you can choose options from. Once you click on the Single Player button, you get to choose from 8 different levels. At first, only the first level is unlocked and you unlock other levels as you complete the previous levels. The optimal move count for a map is shown at the top and your current move count is shown below the optimal move count label. Click on the car you want to move and place it to a different location by dragging it. Once you complete the level, you can continue to the next level.

In the main menu, there are 2 configuration options, one for car skin and one for map theme. When you click Select Theme or Select Car, you are presented with 4 options. You can click on any of them and choose the car and theme to go with.

If you click Multiplayer, you choose whether you want to play against a human or the computer. The computer version is powered by an AI. In the multiplayer game, there are 3 levels. You can choose any of them to start with. It is turn-based and the first player to get his car out of the map wins.

You can quit the game by clicking the X button on the top.

Appendices

Some screenshots and low-level design diagram is given below

1 - Main Menu Screen: This screen shows up when the user runs the game. It includes the navigation buttons for the game. These buttons navigate through the Single Player game, Multiplayer game, Select Car screen, Select Theme screen and Credits screen. The visual for the Main menu screen is given below.

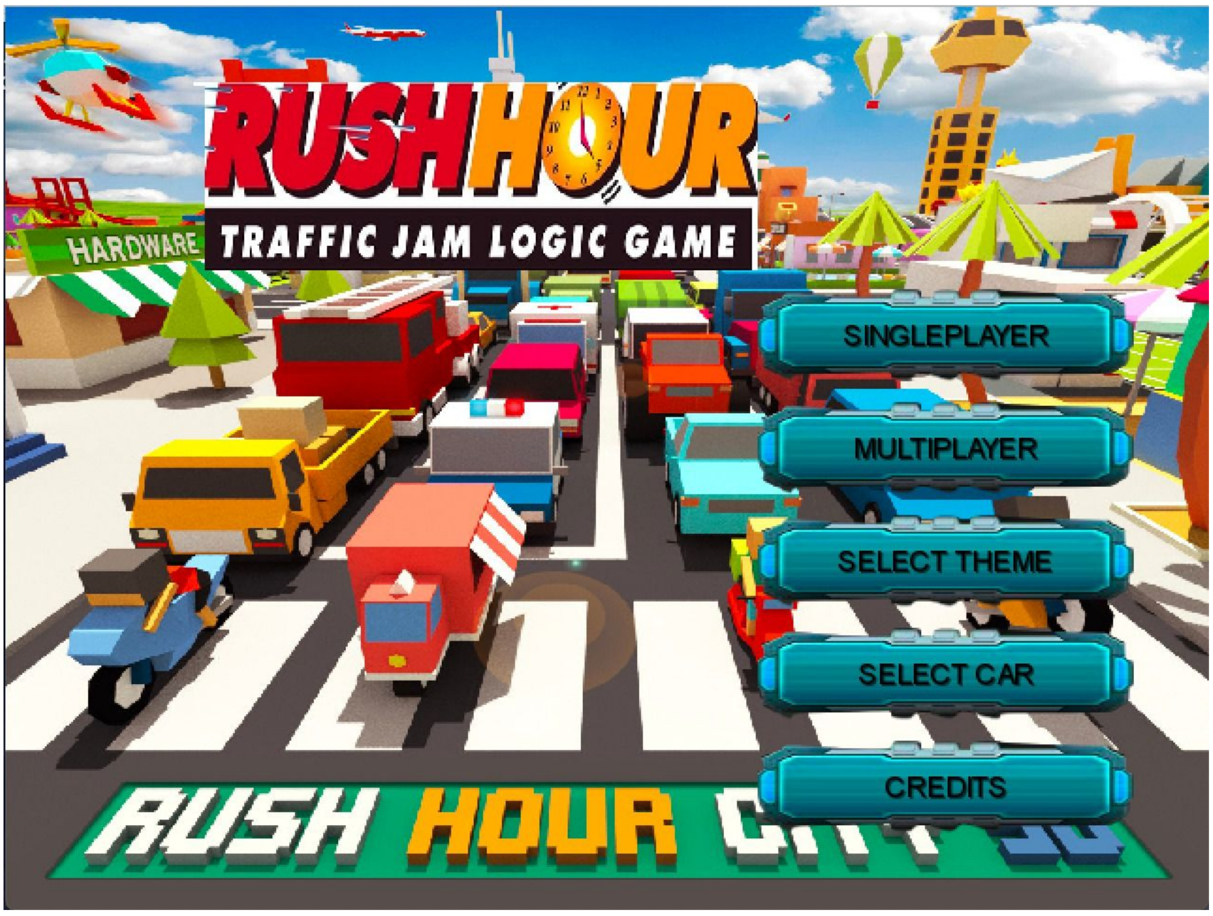

Figure 1: Main Menu Screen

2 - Map Selection for Single Player mode: The player can select a map to play the single player game with the given screen. As levels are passed the new levels become selectable. The stars collected from each level is also illustrated in this map selection screen. The corresponding visual is given below.

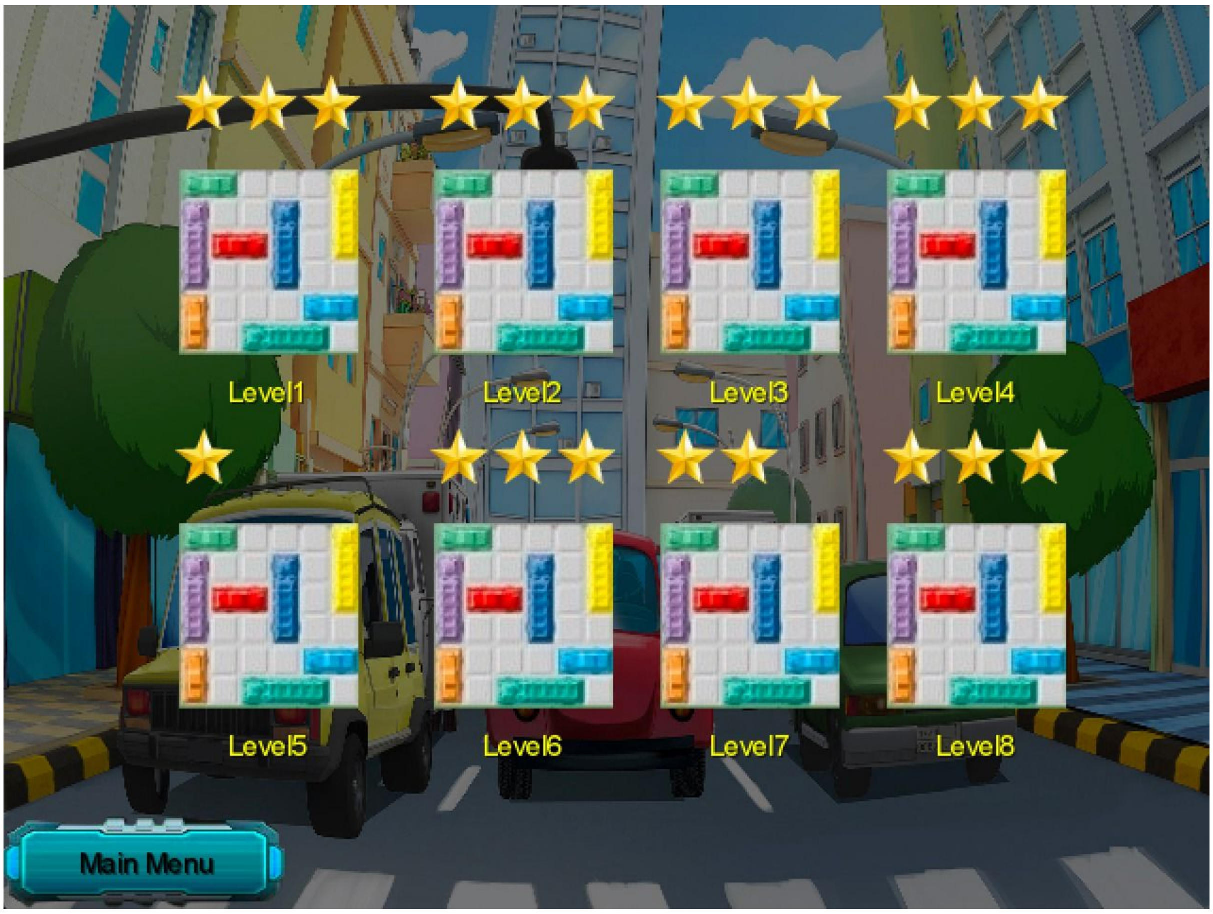

Figure 2: Map Selection for Single Player mode

3 - Single Player In game: The user can play the single player mode of the game using this screen. The move count for the player and the target move count to earn 3 stars from the level is provided in the screen to the user. The taxi shows the escape car and this screen is valid until the player successfully escapes from the map. The screen for the gameplay of level 1 is given below.

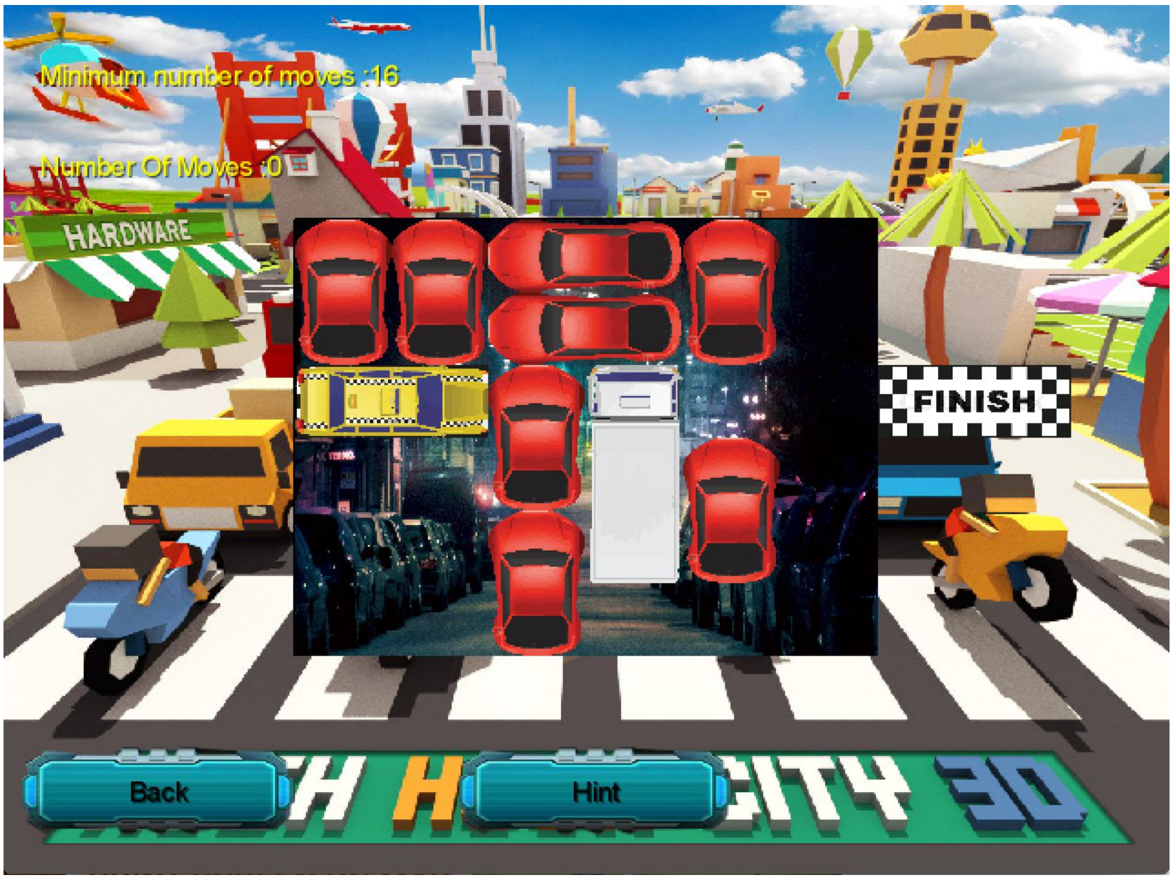

Figure 3: Single Player In game

4 - Single player End of Level: When the player successfully escapes from the map, the illustration shown to the user changes. The user is given the option to continue with the next level or exit from the level. The stars earned from the level is also illustrated. The corresponding visual is provided below.

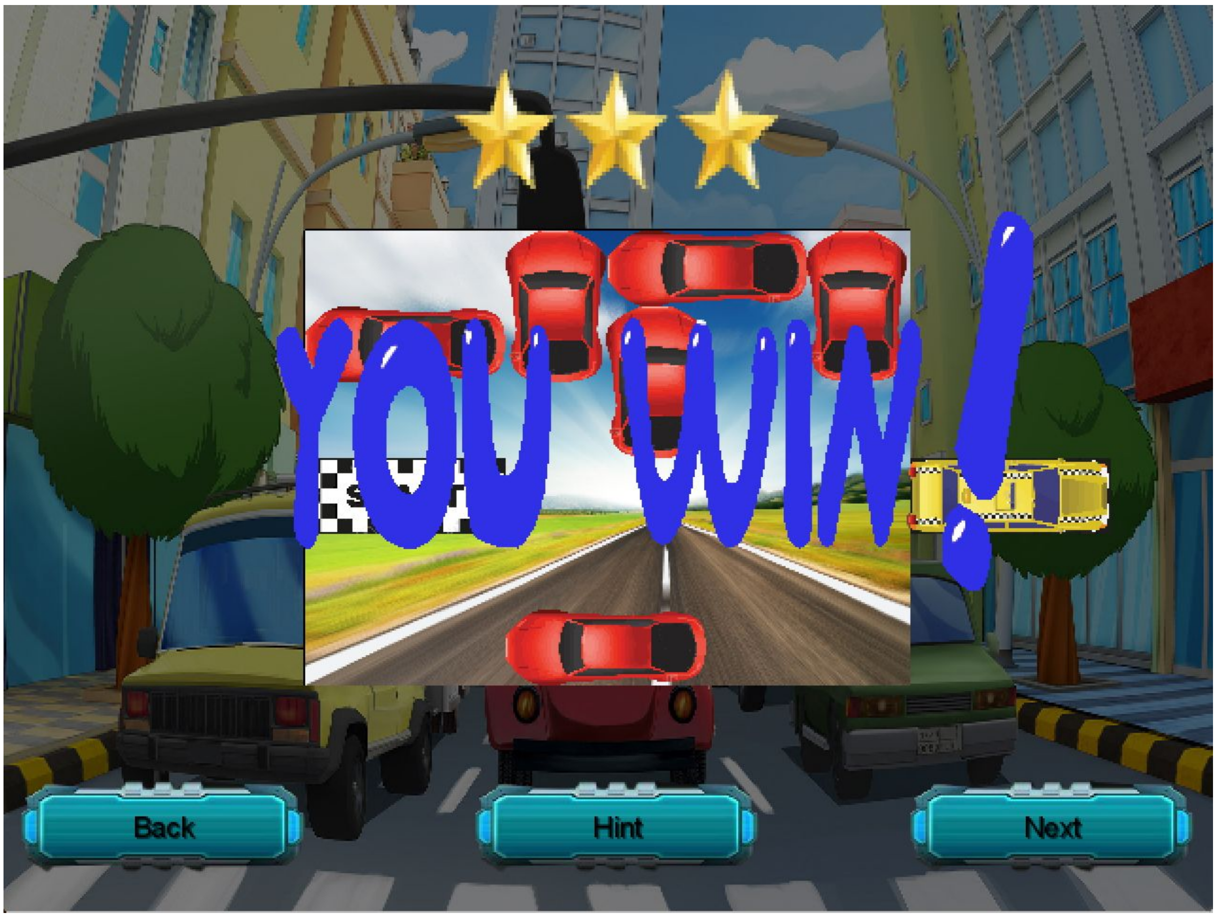

Figure 4: Single player End of Level

5 - Final Low-level design diagram: According to the design changes explained in this report, there are changes in the object model of the application. The changes resulted mainly from the functionalities used from the LibGDX library, which is used for the user interface design. The corresponding UML diagram is given below. The packages inside the object model are provided separately.

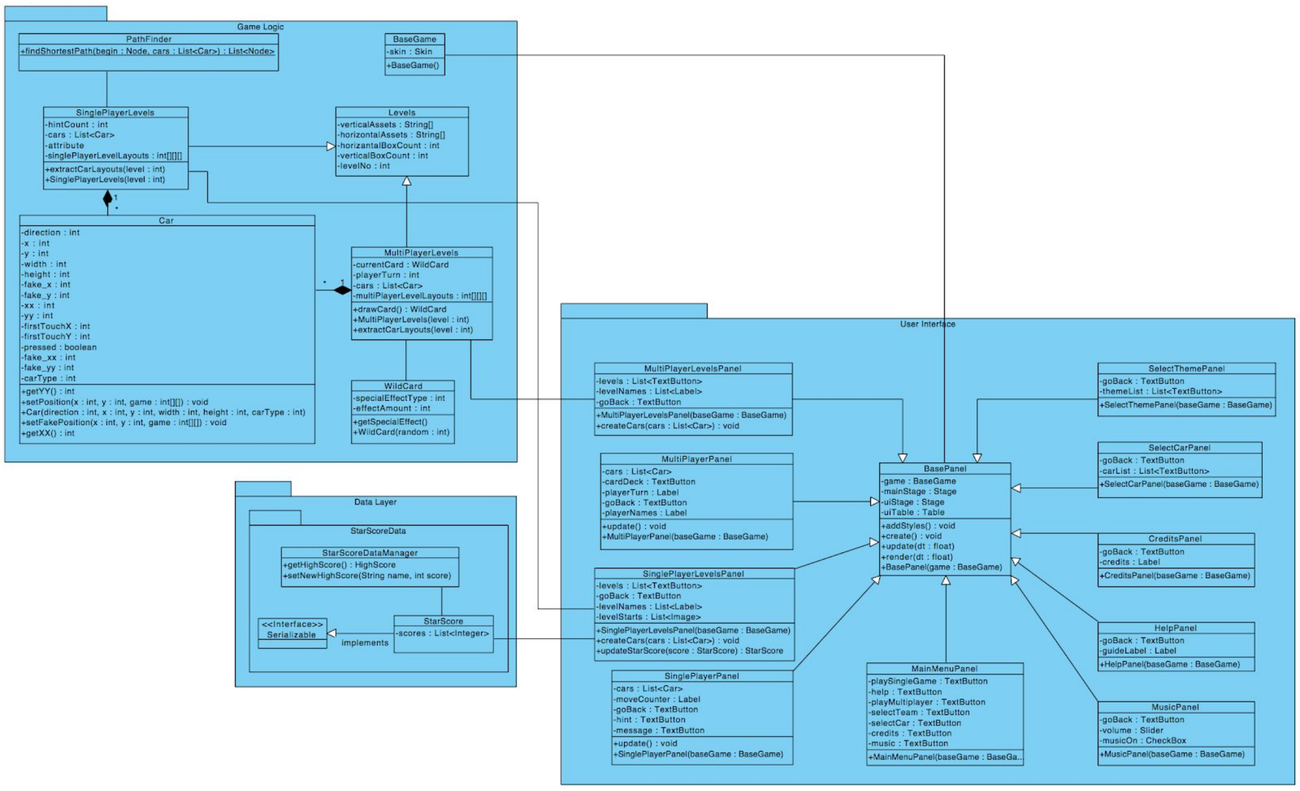

Figure 5: Final Low-level design diagram

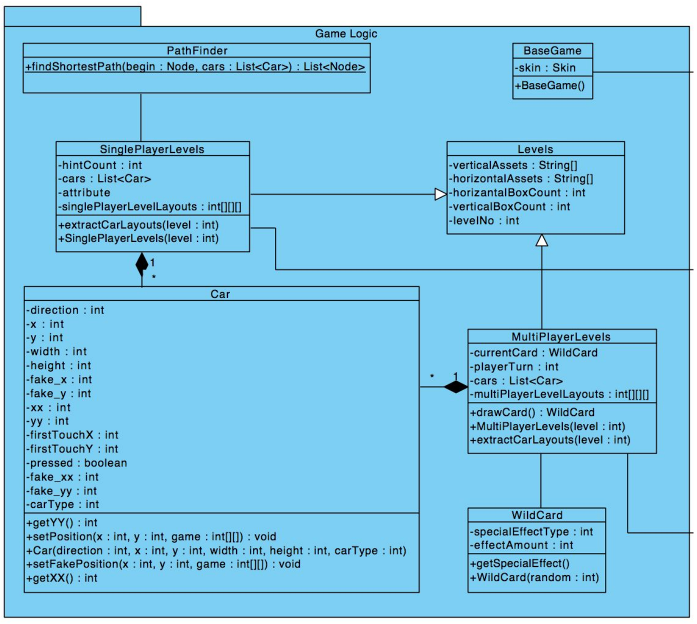

Figure 6: GameLogic

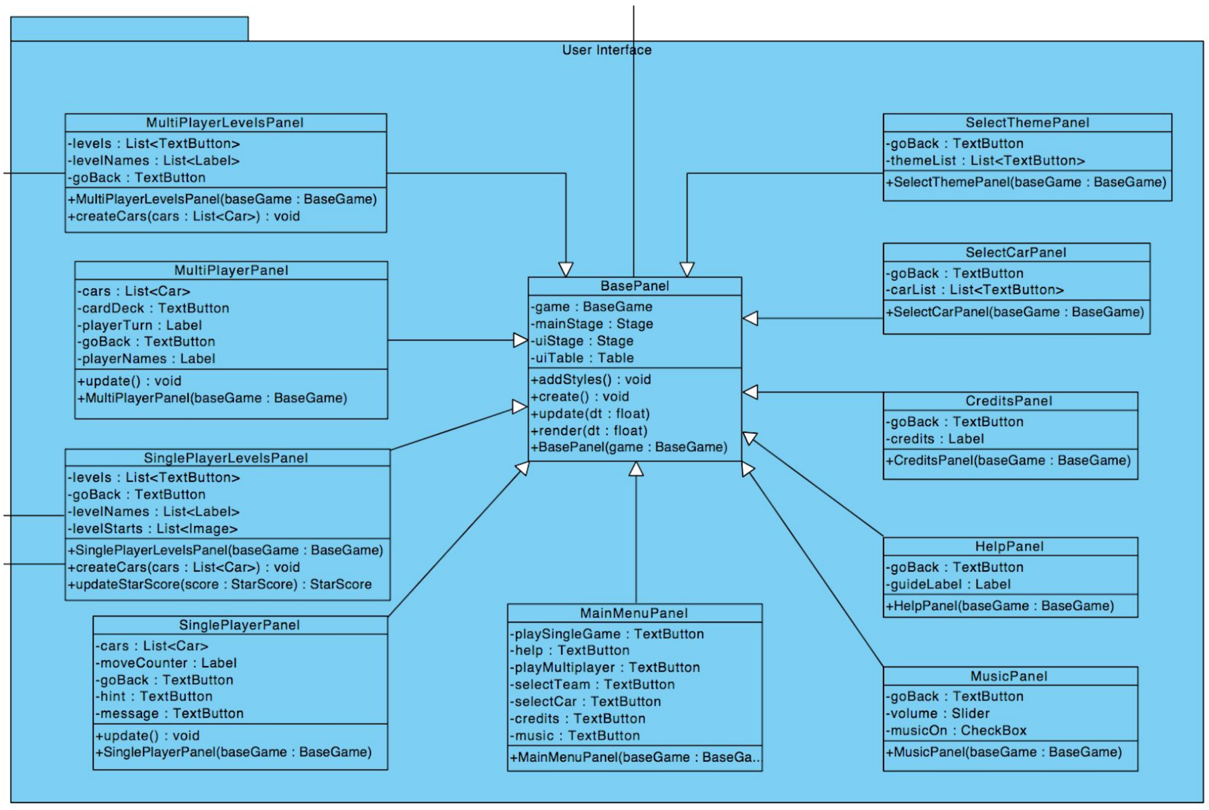

Figure 7: UI

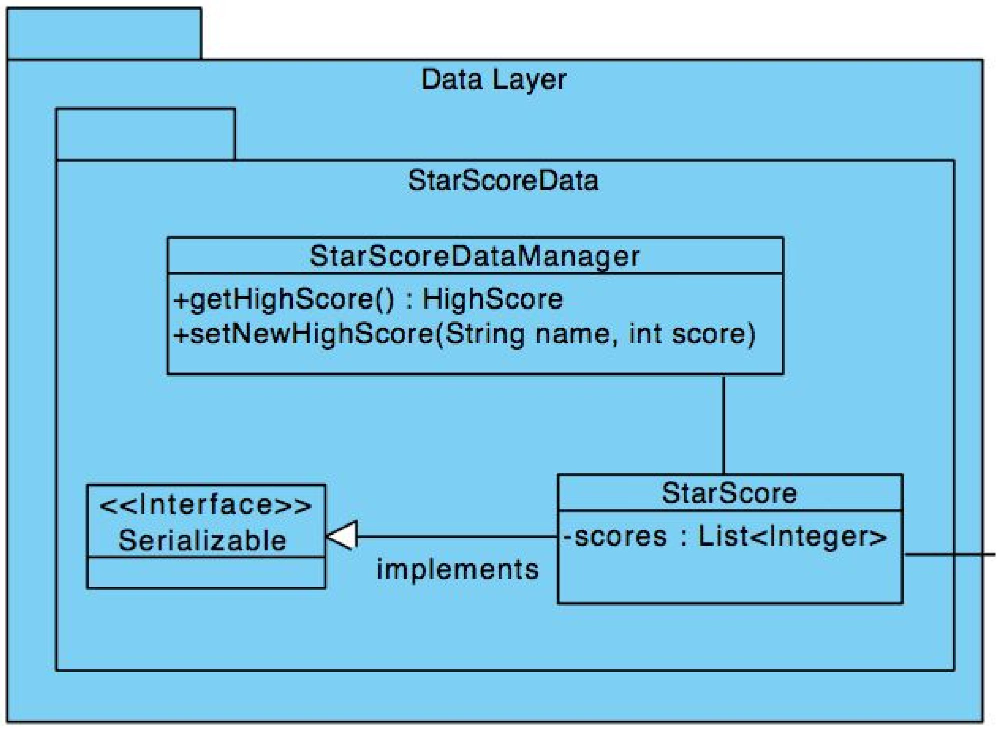

Figure 8: Data Layer