**Analysis Specification on**

**Cars Game**

**Bibek Tamang**

**00163021**

**Computing Project**

**Level 5 Diploma in Computing**

**Softwarica College of IT & E-Commerce**

**Kathmandu, Nepal**

**April 23, 2017**

**Contents**

[**Introduction 3**](#_Toc480843177)

[**Requirements 4**](#_Toc480843178)

[**Functional Requirements 4**](#_Toc480843179)

[**Non-functional Requirements 6**](#_Toc480843180)

[**Prioritization 8**](#_Toc480843181)

[**Use cases 10**](#_Toc480843182)

[**Architecture 14**](#_Toc480843183)

[**System Architecture 14**](#_Toc480843184)

[**Initial Class Diagram 16**](#_Toc480843185)

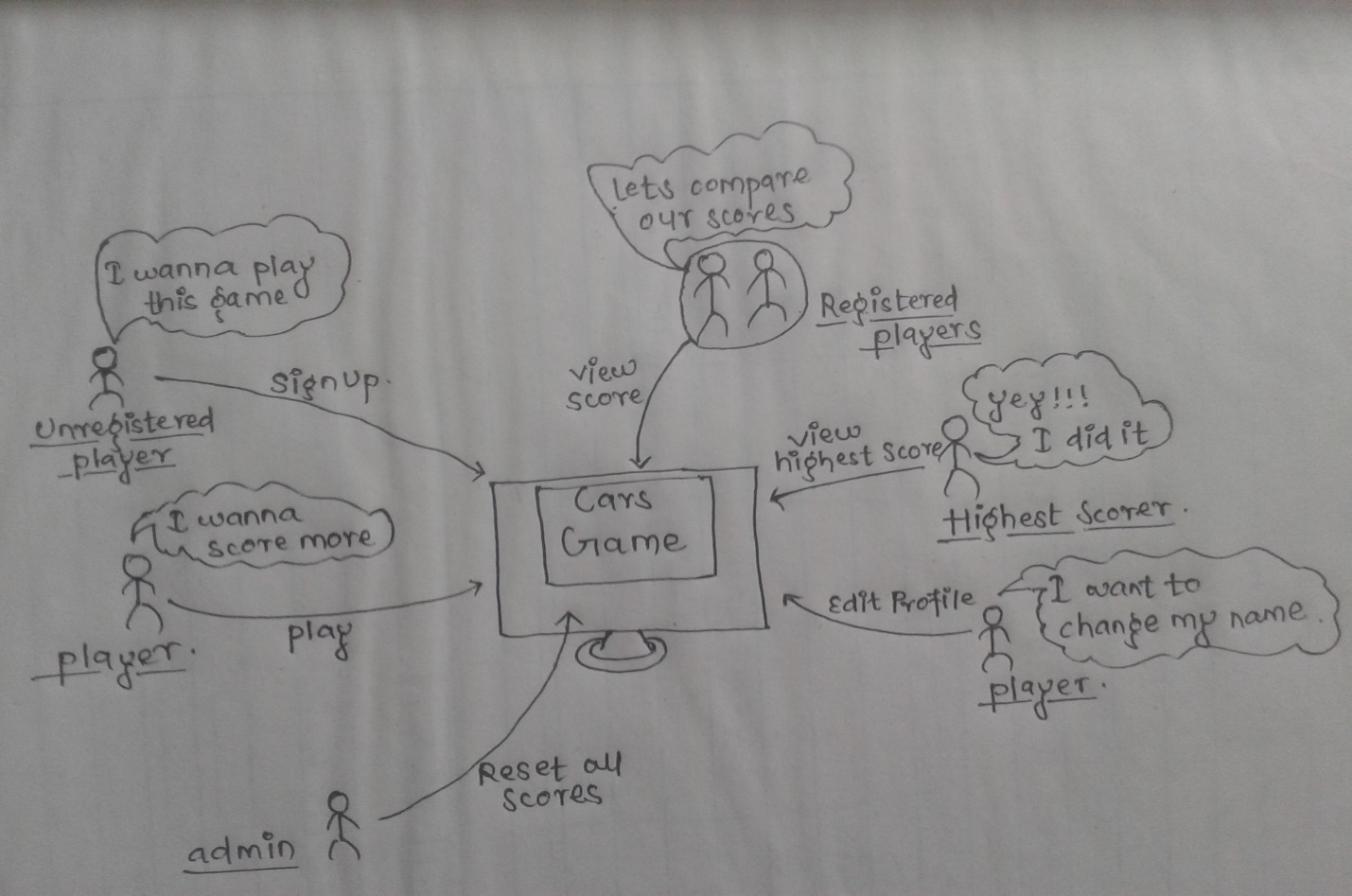
[**Conclusion 17**](#_Toc480843186)

[**References 18**](#_Toc480843187)

# **1) Introduction**

Analysis has always been the most important part in the software or system development. It is the first and the foremost step during the development process. Even in the waterfall model of development methodology, ‘Analysis and Requirements Gathering’ is the first stage. Here in this stage, the project is analyzed to explore the requirements that need to be fulfilled by the system when it is developed. Those requirements which a client or user requires from the system. Those requirements which a developer must keep in mind during the further phases of the system development such as design and development phase. These requirements will be separated as functional and non-functional. Use case diagrams are produced to provide functional overview of the system describing how user interacts with different parts of the system. Now, based upon the requirements and use cases, an architecture of the proposed system will be designed called ‘System Architecture’. Initial class diagram will also be produced to support the design of system architecture.

I am going to perform analysis for my ***Cars Game*** project. First of all, requirements (functional and non-functional) will be understood and gathered. MoScoW prioritization technique will be used to prioritize the requirements. Now, on the basis of specified requirements, use case diagram will be produced which may be one or more. Each interaction presented in the diagram will be described briefly one by one just after the diagram. Then, our final section will be the system architecture where a proposed system is modeled by drawing initial class diagram. This architecture provides initial design of the proposed system which may not include elements that will be added during the design and development stage of our development process.



*Diagram (1): Rich picture of Cars Game*

# **2) Requirements**

Requirement can be defined as the singular documented physical and functional need that a particular product or service must be or perform. It is a raw need which requires external inputs or efforts to deliver a final product. Requirement analysis is the process of analyzing, discovering, defining and documenting the requirements. This involves measuring of each and every functional, non-functional, technical and operational requirement.

It is the discussion and formal agreement between the client and the development agency regarding what is expected of the product. This stage of waterfall model has been the important step in Software Development Life Cycle (SDLC) because this clearly states what will be in the scope and what is out of the scope. Here we have performed analysis and discovered the functional and non-functional requirements for the ***Cars Game***. The requirements identified are clearly stated and described in the following sections. ([Hershey, 2012](#requirements))

## **2.1) Functional Requirements**

These requirements are the desired functionality that the client want us to build and be delivered to them. A functional requirement describes the interaction of system with its environment like what are the inputs to the system, what are the outputs from the system considering all the external and internal factors involved. The functional requirements for the ***Cars Game*** were analyzed and identified. They are described below in detail: -

**Both player and admin**

**ID:** FR1

**Title:** Log In

**Description: -**The administrator or a registered player both should be able to login to the game using their own login credentials. Appropriate message should be provided to them while logging in.

**Rational:** To maintain privacy and security of player accounts

**Dependencies:** FR3

**ID:** FR2

**Title:** Log Out

**Description:** The administrator or a player both should be provided with the functionality to log out from the game. They can simply do this by pressing ‘Log Out’ button from the dashboard or main menu.

**Rational:**

**Dependencies:** FR1

**Only player**

**ID:** FR3

**Title:** Sign Up

**Description:** A new player should be able to sign up with the game in order to create personal profile or account for the game. The player must provide some information for the sign up as requested.

**Rational:** To acquire player credentials for login credentials

**Dependencies:** --

**ID:** FR4

**Title:** Edit Player Profile

**Description:** A player should be able to edit his/her profile in the game. Appropriate message should be provided after the profile has been edited to the player.

**Rational:** To allow users to update their details for the profile in the game

**Dependencies:** FR1

**ID:** FR5

**Title:** View the Highest Score

**Description:** A player should be able to view the highest score scored so far and the name of the player who scored it.

**Rational:** To create a competitive environment in the game

**Dependencies:** FR1

**ID:** FR6

**Title:** View Personal High Score

**Description**: A player should be allowed to view his/her personal best score so far in the game and compare with the highest score of the game.

**Rational:** To allow player to compare personal score with highest score of the game

**Dependencies**: FR1

**ID:** FR7

**Title:** Start Game

**Description:** A registered player should be able to start the game from the main dashboard. They should be navigated to the main game panel when he/she starts the game.

**Rational:** To navigate player to the main game panel.

**Dependencies:** FR1

**ID:** FR8

**Title:** Control Cars

**Description:** The game consists of 2 cars and 2 tracks respectively. A player should be allowed to control these two cars at the same time through separate keystrokes for each car in order to change the lane.

**Rational:** To allow player to control the cars in the game through keystrokes.

**Dependencies:** FR1, FR7

**ID:** FR9

**Title:** Collect Circles

**Description:** During the game runtime, a player should be able to collect circles on the track. Circles can be collected by controlling the cars to change the lanes for each track.

**Rational:** To add difficulty in the game to control car.

**Dependencies:** FR1, FR7, FR8

**ID:** FR10

**Title**: Avoid Squares

**Description**: While playing the game, a player should be able to avoid squares on the track. Squares can be avoided by controlling the car to change the lanes for each track.

**Rational**: To add difficulty in the game to control car.

**Dependencies**: FR1, FR7, FR8

**ID**: FR11

**Title**: Pause Game

**Description**: A player playing the game should be able to pause the game. All the moving objects and the game time should be paused whenever a player pauses the game. The game can be paused using a ‘Pause’ button or any special keystroke specified by the game.

**Rational**: To allow players to pause the game.

**Dependencies**: FR1, FR7

**ID**: FR12

**Title**: Resume Game

**Description**: When a player pauses the game, the player should also be able to resume it. All the paused objects and the game time should continue just after the player resumes the game. The game can be resumed using ‘Resume’ button.

**Rational**: To allow users to resume the paused game.

**Dependencies**: FR1, FR7, FR11

**ID**: FR13

**Title**: Exit to Dashboard

**Description**: After a player pauses the game, the player should be allowed to exit the game and navigate to the dashboard. He/she can exit to the main menu by clicking ‘Exit’ button displayed after the game has been paused.

**Rational**: To allow users to stop playing and navigate to dashboard.

**Dependencies**: FR1, FR7, FR11

**Only admin**

**ID**: FR14

**Title**: Reset Scores

**Description**: The administrator of the game should be allowed to reset all the scores scored by the players in the game. The admin can do this by clicking ‘Reset’ button from the admin dashboard.

**Rational**: To allow admin to reset all the scores scored by the players.

**Dependencies**: FR1

## **2.2) Non-functional Requirements**

Non- functional requirements are the untold parts of the project which are not communicated but really understood as a global standard. They are always supporting the functional requirements. They can also be defined as constraints on a system like performance speed, security and many more which can be optimized to a certain level of extent. The non-functional requirements for the project were identified. They are described below in detail: -

**ID**: NFR1

**Title**: Accuracy

**Description**: The game should calculate and save the scores accurately. It should also display accurate high score from database when a player views it.

**Rational**: Provide accurate outputs.

**Dependencies**: NFR5, NFR7

**ID**: NFR2

**Title**: Security

**Description**: The game should provide security to the player’s account. Player’s information on the database should be secure so that no other players could make changes in it. The scores achieved should also be secured so that no any other players could delete or change it.

**Rational**: To secure player’s information.

**Dependencies**: NFR5

**ID**: NFR3

**Title**: Usability

**Description**: Player should feel comfortable while playing the game. The selection of colors and graphics should be relevant and consistent which makes the game more entertaining, comfortable and ease to interact with.

**Rational**: To make the system easily usable.

**Dependencies**: NFR5, NFR6

**ID**: NFR4

**Title**: Extendibility

**Description**: The game should be easier to extend its features. Any additional functionalities and features should easily be added into the system during the development period.

**Rational**: To allow system functionalities and features be extended.

**Dependencies**: NFR5, NFR9

**ID**: NFR5

**Title**: Availability

**Description**: The game should be available to all the players. All non-registered players should also be able to play the game after signing up with the game. That means, sign-up should be available to all the users.

**Rational**: To make the game available to all the users.

**Dependencies**: --

**ID:** NFR6

**Title**: Performance

**Description**: Performance speed of the game should be the best. Player should be able to interact with the system as fast as possible. The system should be able to process player’s input with great speed such as controlling cars using keystrokes from keyboard.

**Rational**: To increase performance of the game.

**Dependencies**: NFR5

**ID**: NFR7

**Title**: Reliability

**Description**: Player should be able to rely on the system while saving and calculating high scores. The game should be reliable to play so that player could rely on its outputs.

**Rational**: To make the system/game reliable.

**Dependencies**: NFR1, NFR5

**ID**: NFR8

**Title**: Capacity

**Description**: The game should be able to register as many players as possible. Not only this, it should also be able to store as many scores as achieved by the players and calculate the highest score from it.

**Rational**: To increase capacity of the game to handle users.

**Dependencies**: NFR5

**ID**: NFR9

**Title**: Maintainability

**Description**: The game/system should maintainable for the future. The system should easily be able to handle the bugs or changes in the system with minimum effort.

**Rational**: To make the system maintainable for future.

**Dependencies**: NFR5, NFR4

# **3) Prioritization**

Requirements prioritization is another important part of requirement analysis. It is used in software development to determine which candidate requirements should be included in a certain release. Both functional and non-functional requirements identified act as the candidate requirements for prioritization. The purpose of prioritization is to minimize risk during software/system development so that important requirements are implemented first. This involves prioritizing of the requirements identified in accordance of their importance for the proposed system.

Prioritization can be done using many techniques available such as:

* Ranking
* Grouping (Numerical Assignment)
* MoScoW Technique
* Bubble Sort Technique
* Hundred Dollar Method
* Analytic Hierarchy Process (AHP)
* Five Whys

I will be using MoScoW prioritization technique in order to prioritize my both functional and non-functional requirements.

MoScoW Prioritization technique is used by the analysts and stakeholders to prioritize the requirements in a collaborative fashion. Instead of numbers and rankings, they use priority groups which are discussed below: -

1. **Must (Mandatory):** The requirements which are mandatory and must be included in a system are placed here in this group. The must-have requirements of a project are grouped here.
2. **Should:** This priority group includes those requirements which are of high priority. Highly important requirements that should be included for a system are placed here in this group.
3. **Could:** Those requirements which are preferred but not necessary are included in this group. Requirements with less importance that could be added to a project/system are grouped here.
4. **Would/Won’t:** In this group, the requirements which isn’t necessary and can be postponed are grouped. Those requirements which won’t be included in a system but would be added later in future are categorized to this section.

([Prioritization Techniques, 2016](#prioritization))

Analysts and clients discuss the requirements and prioritize them in the above order in this MoScoW technique.

For my requirements too, I have used the same technique and the results of prioritization are shown below in a table:

|  |  |
| --- | --- |
| **Functional Requirements** | **MoScoW** |
| FR1. Log In | Must |
| FR2. Log Out | Must |
| FR3. Sign Up | Must |
| FR4. Edit Player Profile | Should |
| FR5. View Highest Score | Should |
| FR6. View Personal High Score | Should |
| FR7. Start Game | Must |
| FR8. Control Cars | Must |
| FR9. Collect Circles | Must |
| FR10. Avoid Squares | Must |
| FR11. Pause Game | Should |
| FR12. Resume Game | Should |
| FR13. Exit to Dashboard | Must |
| FR14. Reset Scores | Could |

*Table (1): Functional Requirements prioritization using MoScoW*

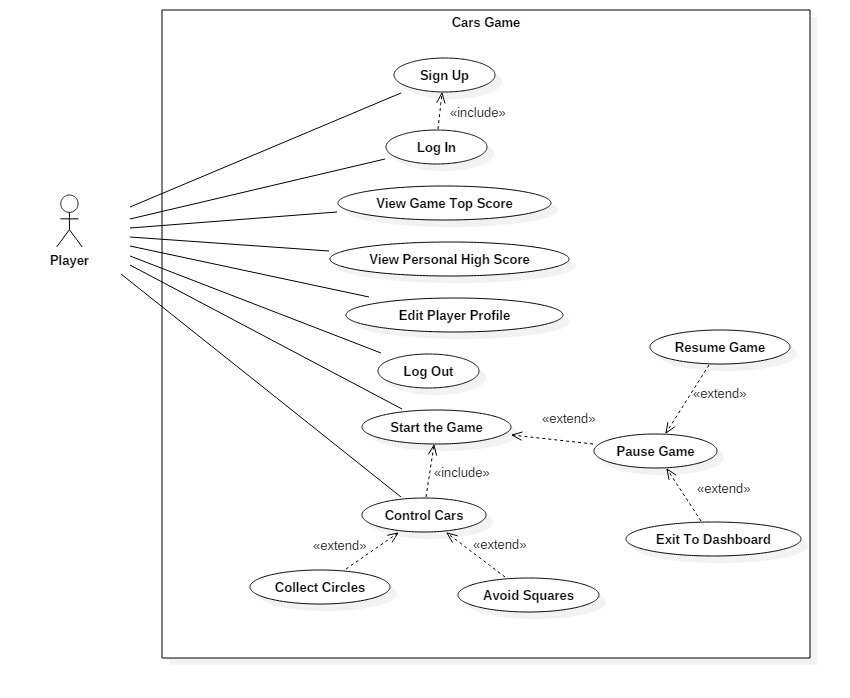
|  |  |
| --- | --- |
| **Non-functional Requirements** | **MoScoW** |
| NFR1. Accuracy | Must |
| NFR2. Security | Must |
| NFR3. Usability | Must |
| NFR4. Extendibility | Should |
| NFR5. Availability | Must |
| NFR6. Performance | Must |
| NFR7. Reliability | Must |
| NFR8. Capacity | Should |
| NFR9. Maintainability | Should |

*Table (2): Non-functional Requirements prioritization using MoScoW*

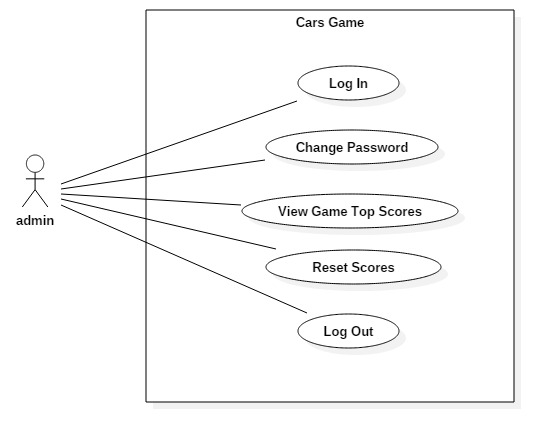
# **4) Use cases**

Use case diagram is an important part in managing your abstractions. It is an UML (Unified Modeling Language) diagram used to represent the broad interactions between different parts of a system. Use case also represents a set of functionalities that must be supported for each part. Those parts are called actors and they may be users or subsystems. Use case is typically used for scenario description. It is designed for the better understanding of the system functionalities. It clearly represents the main activities involved in the system allowing a developer to easily design the system in later stages of development process. They are also really helpful for a developer to present the overview of system functionalities to the clients.

Before diagramming use cases, use case analysis is performed. Use case analysis is an important and valuable requirement analysis technique that has been widely used in modern software engineering. Identification of actors and their use cases on the system is the major activity before preparing use cases. Associations are made between the actors and use cases but not between the actors. It is an UML diagram which is drawn using UML designing tool such as StarUML or Visual Paradigm. For my project, I have decided to use StarUML tool to draw use cases. The use case diagrams were drawn for the ***Cars Game*** in StarUML and they are presented here below: -



*Diagram (2): Use case diagram 1*



*Diagram (3): Use case diagram 2*

**Use case diagram 1:**

Use case scenario description for the ‘Sign up’ from diagram 1 is as follows: -

* Player opens up the sign up/registration form.
* Player fills his/her personal and login credentials such as name, password, address, etc.
* Player clicks the ‘sign up’ button.
* Validation is performed by the system if the form is filled properly.
* The system checks to see if the player already exits in the system.
* The system creates a player with the provided information.
* If registration or sign up is successful, appropriate message is provided by the system.
* If sign up is not successful, the system notifies the player and redirects him/her to the sign-up form.

Use case scenario description for the ‘Log In’ from diagram 1 is as follows: -

* Player opens up the login form.
* Player enters his/her login credentials in the form.
* Player clicks the ‘Log In’ button.
* The system compares the login details provided with the details stored in database.
* If logging in is successful, success message is provided and the player is directed to the dashboard panel.
* If logging in is unsuccessful, appropriate message is provided by the system and the player is redirected to the login form.

Use case scenario description for the ‘View Game Top Score’ from diagram 1 is as follows: -

* Player logins with the system to open up dashboard panel.
* Player clicks the ‘View Top Score’ button to see the highest score in the game so far along with the scorer name.
* The score is retrieved from the database.

Use case scenario description for the ‘View Personal High Score’ from diagram 1 is as follows: -

* After logging in with the game, the player clicks ‘View My High Score’ button from the dashboard to view the highest score he/she has achieved so far in the game.
* The score is retrieved from the database.

Use case scenario description for the ‘Edit Player Profile’ from diagram 1 is as follows: -

* Player clicks ‘Edit Player Profile’ button from dashboard to open up ‘Edit Profile’ form.
* The form is pre-filled with the player information retrieved from database by the system.
* The player edits the details in the form and clicks the ‘Update’ button.
* If update is successful, the system notifies the player.
* If update is unsuccessful, appropriate message is provided to the player and redirected to the form again.

Use case scenario description for the ‘Log Out’ from diagram 1 is as follows: -

* Player clicks ‘Log Out’ button from the dashboard panel to log out from the system.
* Player is directed to login form.

Use case scenario description for the ‘Start the Game’ from diagram 1 is as follows: -

* Player clicks ‘Start the Game’ button from the dashboard after logging in with the system to open the main game panel.
* Player is directed to the game panel and the game time starts
* The game panel consists of two tracks with one car for each track. One car will be colored red and another blue.
* Each track has two lanes.
* Game panel also shows the game running time and score which increases continuously till the game continues.

Use case scenario description for the ‘Control Cars’ from diagram 1 is as follows: -

* Player uses ‘Left-Arrow’ and ‘Right-Arrow’ keys at the same time from keyboard to control two cars at the same time.
* ‘Left-Arrow’ key changes lane of the first car in the first track.
* ‘Right-Arrow’ key changes lane of the second car in the second track.

Use case scenario description for the ‘Collect Circles’ from diagram 1 is as follows: -

* After starting the game, a player controls the cars to change lanes of the tracks to collect circles on the track.
* Circles appear randomly on the tracks which require a player to collect these by colliding cars with the circles.
* If player fails to collect any one of the circles, the game will be over and the player is directed to scoreboard from the game panel.
* Player continuously collects circles till the game continues.
* Score increases continuously until the game is over.
* More longer a player collects circles, more score he/she will achieve.

Use case scenario description for the ‘Avoid Squares’ from diagram 1 is as follows: -

* During the game, player controls the cars using keystrokes to change lanes on the track.
* Player changes lanes to avoid squares on the track.
* Squares appear randomly on the tracks.
* Player needs to protect cars from collision with the squares.
* If any car collides with the squares, the game is over and the player is directed to scoreboard from the game panel.
* More longer a player avoids squares, more longer the game continues and more score he/she will achieve.

Use case scenario description for the ‘Pause Game’ from diagram 1 is as follows: -

* While playing the game, a player can pause the game through ‘P’ keystroke from keyboard or ‘Pause’ button on the game panel.
* All the moving objects including game time and score stops which can later be resumed again.

Use case scenario description for the ‘Resume Game’ from diagram 1 is as follows: -

* After game has been paused, player has option to resume it by just clicking ‘Resume’ option from the game panel.
* Game time, score and objects on the screen continue from where they have been paused.

Use case scenario description for the ‘Exit to Dashboard’ from diagram 1 is as follows: -

* After game has been paused, player has option to exit from the game and return to dashboard.
* Player exits from the game by just clicking ‘Exit to Dashboard’ option on the game panel.

**Use case diagram 2:**

Use case scenario description for the ‘Log In’ from diagram 2 is as follows: -

* Administrator opens up the login form.
* Admin enters his/her login credentials in the form.
* Admin clicks the ‘Log In’ button.
* The system compares the login details provided with the details stored in database.
* If logging in is successful, success message is provided and the admin is directed to the admin dashboard panel.
* If logging in is unsuccessful, appropriate message is provided by the system and the admin is redirected to the login form again.

Use case scenario description for the ‘Change Password’ from diagram 2 is as follows: -

* Administrator opens up a form to change password by clicking ‘Change Password’ option from admin dashboard.
* Admin enters his/her new password for login.
* Admin clicks ‘Change’ button on the form.
* If password is changed successfully, admin is notified by the system.
* If password is not changed successfully, appropriate message is provided by the system and he/she is redirected to the form.

Use case scenario description for the ‘View Top Scores’ from diagram 2 is as follows: -

* Administrator clicks ‘View Top Scores’ option from dashboard to open up score table.
* The table displays top scores of the game with scorer name.
* Scores are listed in descending order with the highest score at the top decreasing continuously as we go down.

Use case scenario description for the ‘Reset Scores’ from diagram 2 is as follows: -

* Administrator clicks ‘Reset Scores’ option from dashboard to reset all the scores scored so far in the game.
* This deletes all score records from database along with the highest score of the game.
* Game is reset to new state with no score records.

Use case scenario description for the ‘Log Out’ from diagram 2 is as follows: -

* Administrator clicks ‘Log Out’ button from the dashboard panel to log out from the system.
* Admin is directed to login form.

# **5) Architecture**

Architecture can simply be defined as the carefully designed structure of something. Here in our case, it is the design of structure of a system/software. In software/system development, Architecture is another important part that comes in analysis phase of the development program. Architecture provides conceptual model of a proposed system representing the structure of system components. It describes how different components of a system or subsystems are composed together to form a complete overall system. An architecture can also provide information on relationships, dependencies and behaviors of system components. They represent how different objects work together to meet the objectives of a system.

An architecture helps in understanding the basic structure and behaviors of system components which is really helpful during the design phase of the development process. It acts as a base for the design and development stage. Thus, architecture has been famous in analysis of a project to provide conceptual, structural, behavioral and brief overview of the proposed system.

## **5.1) System Architecture**

System Architecture is a generic discipline to handle objects called ‘Systems’, in a way that supports reasoning about the structural properties of these objects. It is a response to the conceptual and practical difficulties of the description and design of complex systems. It describes how you are going the mange different aspects of a system such as user interface, entities, models, persistence, etc.

For the ***Cars Game***, I am going to use MVC (Model View Controller) design pattern. The pattern divides the structure of a system into three parts: -

* **Model:** The model represents the data or entity, and does nothing else. They do not depend on the view or controller. They can be player, score, car, etc.
* **View:** The view displays the model data and sends user actions to the controller. They can be independent to the model and controller but, sometimes view can actually be the controller depending on the model. They are user interfaces allowing user to interact with the model through controller.
* **Controller:** The controller provides model data to the view, and interprets user actions. They depend on the view and model. The controller acts as a midway of interaction between users and models. They receive instructions from the user and act on the data.

([Tom Dalling, 2009](#MVC))

The structure of system components is decomposed into three parts which provides better understanding, maintenance and deployment. With the use of MVC pattern, complexity in the structure will be reduced allowing my project to be designed, developed, changed and maintained more easily.

**Receives user action**

View

Model

Controller

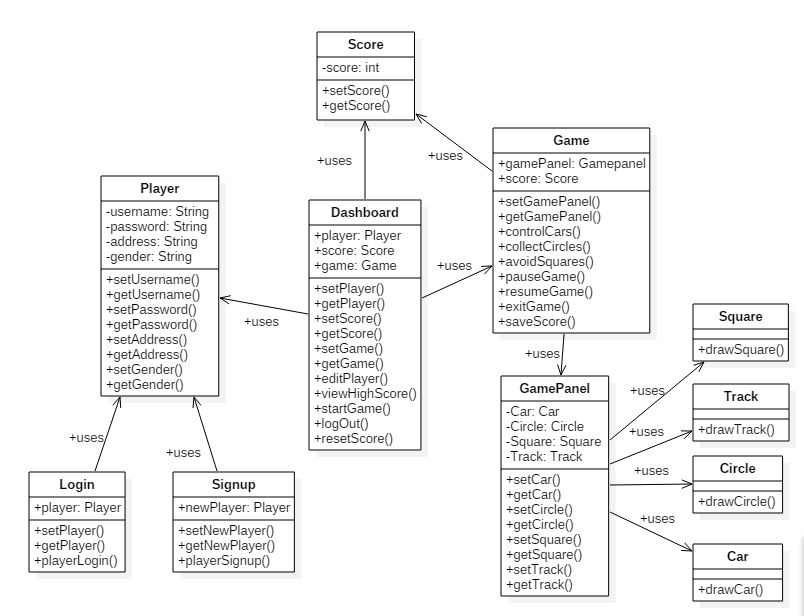
**Acts on**

**Provides model data**

## **5.2) Initial Class Diagram**

Class diagram is an UML (Unified Modelling Language) diagram. It is a modelling technique that runs through nearly all object-oriented methods. The diagram describes different types of objects in a system and various kinds of static relationships that exists between them. They represent the structure of classes, their attributes and operations and also the dependencies and associations between these classes. The diagram is helpful to make the design and development process more easy and systematic.

Here I am going to present the initial class diagram of my ***Cars Game***. The class diagram was designed using NLA (Natural Language Analysis) on the scenario of ***Cars Game***.



*Diagram (4): Initial Class Diagram for* ***Cars game***

This initial class diagram is the first draft class diagram of my project which is prepared just after requirement analysis on the project. The class diagram may change and be updated during the later phases of development process.

# **6) Conclusion**

Requirement gathering and analysis has always been the important and first step in waterfall model of software development. So, the analysis of ***Cars Game*** was the first step of my project. This report provides detailed description of the analysis and requirement capturing performed for the ***Cars Game***.

At first, requirement analysis was performed. Both functional and non-functional requirements were analyzed, discovered, gathered and specified clearly for the game. There were some problems while analyzing the non-functional requirements. There were very less non-functional requirements discovered for my project at first. Later, some more were added after discussion with friends and teachers. Prioritization was the next step after identifying the requirements. MoScoW prioritization technique was used to prioritize both functional and non-functional requirements. Use cases were drawn using StarUML modeling tool based upon the requirements gathered. Separate use cases were drawn for the player and admin producing two use case diagrams. Diagrams are presented clearly here in this report. Then, system architecture was decided which uses MVC (Model View Controller) pattern to divide the system components inti three parts which makes the development and understanding of the system systematic and easier. Finally, initial class diagram was designed using the same modelling tool used for use case called StarUML. Initial class diagram was designed directly from the scenario using NLA (Natural Language Analysis) technique to identify classes and operations. The initial class diagram will be further worked on in the next step of our development process. Now, the project is ready to move into next phase where the designing of various structural and dynamic models will be more focused on.

# **References**

Hershey. (December 22, 2012) *SDLC: Importance of Requirement Analysis* [online] Available at: <http://www.codespread.com/sdlc-importance-of-requirement-analysis.html> [Accessed on: April 20, 2017]

(August 18, 2016) *A List of Requirements Prioritization Techniques You Should Know About* [online] Available at: <https://businessanalystlearnings.com/blog/2016/8/18/a-list-of-requirements-prioritization-techniques-you-should-know-about> [Accessed on: April 21, 2017]

Dalling.T. (May 31, 2009) *Model View Controller Explained* [online] Available at: <http://www.tomdalling.com/blog/software-design/model-view-controller-explained/> [Accessed on: April 22, 2017]