Speed Racers : Furious Adventure

(An Interactive 3D Racing game)

Asad Shoaib (086-BSCS-12)

Hussain Arshad (092-BSCS-12)

Session 2012-2016

Department of Computer Science

GC University Lahore.

Speed Racers : Furious Adventure

(An Interactive 3D Racing game)

Submitted to : GC university

in partial fulfillment of the requirements
For the degree of

BACHELOR OF SCIENCE

IN

COMPUTER SCIENCE

BY

Asad Shoaib (086-BSCS-12)

Hussain Arshad (092-BSCS-12)

Session 2012-2016

Department of Computer Science
GC University Lahore.

Declaration :

We, Asad Shoaib Roll No. 086-BSCS-12 and Hussain Arshad Roll No. 092-BSCS-12 students of BS honors in Computer Sciences session 2012 to 2016, hereby declares that the matter printed in the thesis titled **Speed Racers : Furious Adventure (An interactive 3D Racing game)** is my own work and has not been printed, published and submitted as research work, thesis or publication in any form in any University, Research Institution etc in Pakistan or abroad.

Dated

Asad Shoaib

Hussain Arshad

Project Completion Certificate :

It is certified that the work contained in this Project titled **Speed Racers : Furious Adventure** has been carried out by **Asad Shoaib** Roll. No **086-BSCS-12** and **Hussain Arshad** Roll. No **092-BSCS-12** under my supervision.

Dr. Syed Asad Raza Kazmi

Director

Department of Computer Science

GC University Lahore

Dated	Supervisor

Submitted Through :

Dr. Syed Asad Raza Kazmi Director Department of Computer Science GC University Lahore	Controller of Examination GC University Lahore

Acknowledgment :

First of all We are grateful to the Almighty **Allah**, Who blessed us with the best of everything, We offer our Special thanks to the best and affectionate supervision of **Dr. Asad Raza Kazmi**, Director, Department of Computer Science, GC University Lahore, for his guidance and support throughout our work. Next we want to thanks all our Teachers and Mentors. without their valuable direction this might be impossible for us.

We also would like to say thanks to our my beloved parents who gave us everything we ever asked and helped us in every step of life, and our friends also deserve many thanks for their loving encouragement and prayers.

Abstract :

In this project we are focusing on developing a 3D racing car game, using an incremental development method. This project will cover the implementation of graphics, physics engine, Multiplayer support, sound effects and background music.

Our idea is about to create an adventure and action based video game whose main purpose is to provide really entertaining experience to the people of age group between 8 to 15. For achieving this goal, we have applied an incremental approach to develop an entertaining Multiplayer online racing game. In order to control the mechanics of the video game, we applied Artificial Intelligence, Intelligent Agents, and modern 3D technology approaches such as Unity 3D Game Engine and Photon Cloud.

The game designed and developed in this project is an innovative attempt to develop an engaging multi-player adventure game for kids aged 8 to 15. Our game incorporates the effective use of game theory and also business opportunity, among business opportunity we have integrated google ads for revenue generation. At the completion of this project we will deploy our game in android applications marketplace.

The purpose of writing this documentation is to describe all about the game that is how to analyze game problems, the efficiency of the game, put the elements of interest, structure and implementation process of the game design.

Contents

1 Introduction of the game : 1

1.1 Overview : . . . 1

1.2 Graphics : . . . 2

1.2.1 Everlasting Impression : . . . 2

1.2.2 Graphical effects : . . . 3

1.2.3 Multiplayer game : . . . 3

1.3 Development Facilities : . . . 3

2 Literature Review 5

2.1 Previous Work : . . . 5

2.2 The idea . . . 8

3 Requirement specification : 10

3.1 Functional requirements : . . . 10

3.1.1 Single and Multiplayer modes : . . . 11

3.1.2 Android based : . . . 11

3.1.3 3D platform : . . . 11

3.1.4 Single level : . . . 11

3.1.5 Adventure based : . . . 11

3.1.6 2d GUI and menu system : . . . 12

3.1.7 Testing : . . . 12

3.1.8 Written in C sharp : . . . 12

3.1.9 Photon cloud API's : . . . 12

3.1.10 Models : . . . 12

3.1.11 Textures : . . . 13

3.1.12 Power-ups : . . . 13

3.1.13 Graphical effects : . . . 13

3.1.14 Sounds and music : . . . 13

3.1.15 Time Estimation : . . . 13

3.2 Non-functional requirements : . . . 14

4 Project design and Implementation Constraints: 15

4.1 Development process : . . . 15
4.2 Software process model : . . . 15
4.3 Software development life cycle : . . . 16
4.4 Incremental process methodology : . . . 18
4.5 Our proposed model for developing project: . . . 19
4.6 Architecture overview : . . . 21
4.6.1 Computational Intelligence : . . . 22
Intelligent Agents : . . . 22
Artificial intelligence : . . . 22
4.6.2 Development tools : . . . 23
4.6.3 Photon Cloud : . . . 23
4.7 Design description : . . . 24
4.7.1 Initial concept : . . . 24
4.7.2 Second concept : . . . 24
4.7.3 Final concept : . . . 24
4.7.4 Project scope : . . . 25
4.7.5 Gameplay : . . . 26
4.7.6 Level design : . . . 26
4.7.7 Game modes : . . . 26
4.7.8 Controls of the game : . . . 27
4.7.9 Movements : . . . 27
Move Left: . . . 27
Move Right: . . . 27
Break: . . . 27
Acceleration: . . . 27
Reverse: . . . 27
4.7.10 Winning and Losing : . . . 27
4.7.11 Power-ups : . . . 27
Nitro: . . . 28
Punch: . . . 28

Shield: 28
4.7.12 System requirements : 28

5 Game Design and Implementation : 29
5.1 Screen Shots and Working 29
5.1.1 Enter Name Panel : 29
5.1.2 Main Menu : 30
5.1.3 Store Menu : 31
Purchase new car : 31
Select unlocked cars : 32
5.1.4 Upgrade Menu : 33
Nitro upgrade : 33
Shield upgrade : 33
Engine upgrade : 34
Brakes upgrade : 34
5.1.5 Game Modes : 36
Easy : 36
Normal : 36
Hard : 36
5.1.6 Player Car Controls GUI : 37
Move left : 37
Move right : 37
Move forward : 37
Brakes : 37
Reverse : 37
5.1.7 Game sounds : 38
Background Music : 38
Shield pick up : 38
Nitro pick up : 38
Obstacle collision : 38
Menu Buttons : 38
5.1.8 Enemy car AI : 39

Path points : 39
Rival car nitro and shield mechanism : 39
Enemy car power-up collision detection system : 40
5.1.9 Power-up's : 41
Nitro : 41
Mega Punch : 42
Shield : 43
5.1.10 Obstacles : 44
Fences : 44
Poison box : 44
Rocks : 45
Meteorites : 45
Rotating doors : 46
5.1.11 MultiPlayer : 47
5.2 Few Important Scripts : 49
5.2.1 Player Car Script : 49
5.2.2 AI Car Script : 61

6 UML diagrams **79**
6.1 DataFlow Diagrams : 79
6.2 UseCases Diagrams : 83
6.3 Activity Diagrams : 116

7 Test Cases **142**
7.1 PlayGame 142
7.2 EasyMode 143
7.3 NormalMode 144
7.4 HardMode 145
7.5 QuitGame 146
7.6 UpgradeEngine 147
7.7 UpgradeNitro 148
7.8 UpgradeShield 149

7.9 UpgradeBrakes . . . 150
7.10 ChangeCar . . . 151
7.11 CreateRoom . . . 152
7.12 JoinRoom . . . 153
7.13 MoveAICar . . . 154
7.14 MoveLeft . . . 155
7.15 MoveRight . . . 156
7.16 ApplyBrakes . . . 157
7.17 ReverseCar . . . 158
7.18 PickupShield . . . 159
7.19 PickupNitro . . . 160
7.20 PunchPickup . . . 161
7.21 CollectCoins . . . 162
7.22 CollisionWithRocks . . . 163
7.23 CollisionWithMeteorites . . . 164
7.24 CollisionWithFences . . . 165
7.25 AICarObstacleDetectionMoveLeft . . . 166
7.26 AICarObstacleDetectionMoveRight . . . 167
7.27 AICarObstacleDetection . . . 168
7.28 RaceFinish . . . 169

8 Conclusion and Future Work 170

List of Figures

1 Components . . . 21
2 Enter Name Panel . . . 29
3 Main Menu Image . . . 30
4 Store Menu . . . 31
5 Purchase Car . . . 32
6 Select Car . . . 32
7 Upgrade Menu . . . 33
8 Nitro Upgrade . . . 34
9 Shield Upgrade . . . 34
10 Engine Upgrades . . . 35
11 Brakes Upgrades . . . 35
12 Game Modes . . . 36
13 Car Controls . . . 37
14 Enemy AI . . . 39
15 Nitro . . . 41
16 Punch . . . 42
17 Shield . . . 43
18 Fences . . . 44
19 PoisonBox . . . 44
20 Rocks . . . 45
21 Meteorites . . . 45
22 Rotating doors . . . 46
23 Lobby . . . 47
24 Room Name . . . 47
25 Creating Room . . . 48
26 Waiting For Players to join . . . 48
27 DataFlow Upgrades Menu . . . 79
28 DataFlow Market Menu . . . 80
29 DataFlow Multiplayer . . . 81
30 DataFlow Single Player . . . 82

31 UseCase PlayGame 83
32 UseCase Easy Mode 84
33 UseCase Normal Mode 85
34 UseCase Hard Mode 86
35 UseCase Exit 87
36 UseCase Game 88
37 UseCase Upgrade Engine 89
38 UseCase Upgrade Nitro 90
39 UseCase Upgrade Shield 91
40 UseCase Upgrade Brakes 92
41 UseCase Change Car 93
42 UseCase Create Room 94
43 UseCase Join Room 95
44 UseCase Race Start 96
45 UseCase Start Position 97
46 UseCase Start Car 98
47 UseCase Move Car 99
48 UseCase Left Turn 100
49 UseCase Right turn 101
50 UseCase Apply Brakes 102
51 UseCase Stop Car 103
52 UseCase Reverse Car 104
53 UseCase Pickup Shield 105
54 UseCase Pickup Nitro 106
55 UseCase Pickup Coin 107
56 UseCase Collide with Rock 108
57 UseCase Collide with Meteriote 109
58 UseCase Collide with Fence 110
59 UseCase AI car move left 111
60 UseCase AI car move right 112
61 UseCase AI car turn 113

62 UseCase Finish Race 114
63 UseCase currency and high score save 115
64 Activity Easy mode 116
65 Activity Normal Mode 117
66 Activity Hard Mode 118
67 Activity Upgrade Menu 119
68 Activity Nitro Upgrade 120
69 Activity Brakes Upgrade 121
70 Activity Shield Upgrade 122
71 Activity Engine Upgrade 123
72 Activity Market Menu 124
73 Activity Car Selected 125
74 Activity Purchase Car 126
75 Activity Previous Car 127
76 Activity Next Car 128
77 Activity Market Menu 129
78 Activity Room Joined 130
79 Activity Create Room 131
80 Activity MultiPlayer 132
81 Activity Coin Pickup 133
82 Activity Nitro pickup 134
83 Activity shield pickup 135
84 Activity Punch added 136
85 Activity Collision with rock 137
86 Activity collision with fence 138
87 Activity collision with rotating gate 139
88 Activity collision with meteorites 140
89 Activity Playing Field 141
90 TestCase PlayGame 142
91 TestCase EasyMode 143
92 TestCase NormalMode 144

93 TestCase HardMode 145
94 TestCase QuitGame 146
95 TestCase UpgradeEngine 147
96 TestCase UpgradeNitro 148
97 TestCase UpgradeShield 149
98 TestCase UpgradeBrakes 150
99 TestCase ChangeCar 151
100 TestCase CreateRoom 152
101 TestCase JoinRoom 153
102 TestCase AI car Moving 154
103 TestCase MoveLeft 155
104 TestCase PlayGame 156
105 TestCase ApplyBrakes 157
106 TestCase ReverseCar 158
107 TestCase PickupShield 159
108 TestCase PickupNitro 160
109 TestCase PickupPunch 161
110 TestCase CollectCoins 162
111 TestCase CollisionWithRocks 163
112 TestCase CollisionWithMeteorites 164
113 TestCase CollisionWithFences 165
114 TestCase AICarObstacleDetectionMoveLeft 166
115 TestCase AICarObstacleDetectionMoveRight 167
116 TestCase AICarObstacleDetection 168
117 TestCase RaceFinish 169

بسم الله الرحمن الرحيم

1 Introduction of the game :

Speed Racers: Furious Adventure is developed as bachelors thesis documentation for the department of computer science at Government college university Lahore. The aim of this documentation is to show the whole concept along with development processes, implementation design, and structure as well as its functionalities (functional and non-functional) and others. In this project, Unity3D Game Engine is used to develop this adventurous racing game. Digital games can solve some of the problems where children cannot involve in a physical activities because they so little to do so, space limitations and dangers associated with these activities. These problems can be solved by providing them 3d simulations games like car simulation, aircraft simulation and Ship simulations. These activities are not attainable in real life but they can enjoy it by using a video games.

1.1 Overview :

We are designing and developing an interactive multiplayer racing game for the age group of 8 to 15. This game combines the elements of traditional racing games and elements of current successful games available in a market.

When user starts the game, Name panel will be shown into which user have to enter his name. Then main menu screen will be appear. In main menu user will have several options to choose from which includes single player gameplay and multiplayer gameplay, Store and Upgrades.

When user will select the store button then Market menu will appear in which user will have several options to choose from which includes purchase of differ-

ent cars and their selection.

If user selects upgrades button then upgrades menu will appear and user will have options to upgrade several features which includes engine, nitro, shied and brakes up gradation.

When user will select single player button then game modes menu will appear, there will be three games modes to choose from which are easy, normal and hard modes. Easy mode is for beginners who are new to the racing games, normal mode is for experienced player and hard mode is for die hard gamers. Each mode corresponds to the difficulty of a game, rival cars will get upgrades according to the selected game mode.

After mode selection game will race will begin, in order to win the race the user will have to reach the finish line while dodging all the obstacles which includes rocks, meteorites, rotating gates, fences and poison boxes. During gameplay user can coins spread on a track and the power-ups which includes nitro, shield and mega punch. On the completion of a race in case of win user will get a reward which he can use to purchase upgrades and new cars.

1.2 Graphics :

Graphics creation for this game was a challenging task.

1.2.1 Everlasting Impression :

The game result must impress whoever plays the game. It should last long, and make the players come back and play it over and over again.

1.2.2 Graphical effects :

We decided to use unity as our platform to develop our 3D game. This decision was made with regard to that the platform had many inbuilt tools and provided a good framework for us to get started with the development as fast as possible.

The requirement for the game to contain 3D graphics introduced an interesting challenge for the project group since all had none or little experience in 3D Graphics. The complete game contains graphics made in both Blender and adobe photoshop.

1.2.3 Multiplayer game :

We have incorporated multiplayer features in our game by using a photon cloud and its APIs. Also, it can be played offline in a single player mode. This is the major feature of the game that seems to be handle difficultly as none of them have experienced to make such a game. We have learned a lot, one of the most difficult part of multiplayer was its error handling which includes the handling of network clients behavior. For example if during game Wi-Fi is turned off then the game should be terminated. We have created the lobby system for our multiplayer game in which players can create new rooms and can join existing rooms to play a game.

1.3 Development Facilities :

Software's Constraints :

Softwares required for the project are:

1. Unity3d
2. Visual Studio
3. Adobe Photoshop
4. Blunder

2 Literature Review

Car racing games are getting famous because of ever growing driving simulations in Android. This is all due to the smart capabilities of the Smartphone. The physics between car track and speed gives the player a real virtual experience. This virtual experience gives a feel that we are racing our favorite car on our favorite track and that too on our mobile devices. 2015 and 2016 have been the year of virtual reality and racing games are enjoying the perks of it. The effects and graphics of the racing games today is way beyond our imagination. The features are amazing and keep updating it according to the need of the game.

2.1 Previous Work :

Some of the best games developed in the past that inspire our project, we are elaborating their features and what makes them successful.

- **Forza Motorsports 6 :**

  Forza Motorsports was the highly awaited game of the year 2015 which was released in September for Xbox one. Whats with new with the features? Thats the rainy, wet and sloppy weather racing. To enhance the features and to keep it real the developers personally visited racing tracks to find out where water puddles are developed. Forza Motorsports 6 cast with a lot of numbers of different cars that is around 450. A Series of cars includes FIA, BTCC, WEC, V8 supercars, WTCC. Formula E and F1 are also included in the game.

- **Need for speed Franchise :**

  The most highlighted racing franchise is a need for speed. Need for speed 2015 being the reboot for the franchise has been able to do justice with its features. According to the new feature, the game will need an internet connection to play the game. Need for speed 5 will be released for Microsoft windows and Xbox and as well as play station four.

- **Project cars :**

  Project cars were developed and published by Slightly Mad Studios. The game was released on various platforms like Microsoft Windows, Xbox one, and Linux. The developers later released a date for Nintendo but it seemed like NFS didnt run smoothly on Nintendo. Project Cars is now popular among many consoles. Being a limited budget game, the cars you will find will be models of Mitsubishi only. The Japanese cars are said to be very expensive, so they are out of the question. While Xbox one has been suffering from low audio quality and bugs. But, on a lighter note when it comes down to its graphics Project Cars has beaten their rivals. This game portrays the real life racing, shakedowns, and random race on main tracks.

- **Asphalt 8: Airborne :**

  This game is an out and out arcade racing game. And as the title says it all, the game is a Track Mania kind of game. The jumpy track allows you to do stunts like 360-degree flips and roll barrels. There many sports cars in the game. Almost around 47 cars are there to choose from. Cars like Ferrari FXX, Pagani Zonda R, and Bugatti Veyron. And these cars

have some freshly recorded new sounds now. The game was introduced for smartphones and tab users. But now it is available on Microsoft windows 8 and you can find this game on windows store.

- **The crew-wild run :**

  The crew wild run game is a game with the worldwide open destination. The new expansion in games brings lots of challenges and new cars. The game was released two weeks after a need for speed. The crew has been praised for its amazing graphics, but they didnt find the physics up to the expectation. The game will take you to places in the United States name the summit, where you have to compete in different challenges in order to find the king of the road. The vehicles you will see in the game are cars, trucks and motorbikes.

- **Drive club :**

  Drive club is a play station 4 based game, it was released in 2014. The game believes in working as a team and faces the challenges. Its graphics have been improved yet, the environment looks dull. Through multiplayer option, now you can connect and team up with real world players and play with them. The game was published by Sony computer entertainment only and works with a play station 4 only.

- **Temple run :**

  Temple run is one of the most successful games in the history of mobile devices. When this game was released, it introduces some of the most

revolutionary features that include in-game currency collection, hurdles, and power-ups. Its an infinite runner game, which means the game never stops, the game will continue until the player will die. Temple run was released only for mobile platforms.

- **Death Race Movie :**

  Everyone knows about death race, it is a really famous Hollywood movie in which several racers participate in a race, in order to earn their freedom, in order to survive they have to use their special race cars which are mounted with special weapons like machine guns, rocket launchers, and bombs. By using their firepower the racers have to defend themselves and eliminate other racers on a track. Its a really fun idea, which we think that should be adopted in a game.

2.2 The idea

Our idea is simple to some extent, we have extracted several features and concepts from the above-mentioned games and movies. We want to convert them into fun Multi-Player racing game [**MultiPlayerRacing**] in which we can implement all the features that have made the above-mentioned games successful. The innovative idea [**anderson2016innovative**] of multiplayer racing comes from the different successful games, We have extracted the idea of coins placement in a game which user can collect and use them to buy a new cars or purchase upgrades for their current cars. The idea of the shield is collected from Need for speed Blur, We have extracted the idea of placing nitro on a game track from Asphalt 8, and this is a really cool feature that allows the

user to catch nitro on a track instead of recovering nitro with time.

The unity3d network functionality [**NetworkFunctionality**] will be achieve by photon cloud and we have extracted the idea of using a Mega Punch from the death race movie, instead of adding firepower which shows destruction type effects, because we are developing this game for children's that's why we have added mega punch to destroy enemy cars. Which is really fun playing a factor in this game, we have combined these factors and added few new ideas that involve the obstacle systems, and there are three types of obstacles in this game which are rocks, Meteorites, and Rotating Doors. This game will be ported in android platform [**GameForAndroidDevice**] by using unity3d.

In order to reach the destination the cars have to dodge obstacles, one of the special obstacles that we have included is the poison box, when the car hit the poison box it becomes poisonous for 30 seconds, during its poisonous period the car speed and capabilities drops drastically. Multi user participation [**RacingGameMultiUser**] in this game will make it even more interesting. At the end when a player wins the race it will get coins as a reward according to its position, 1st, 2nd, and 3rd are winning positions and greater than 3 are losing positions. We have included the coins collection mechanism on a track because we wanted that if the player loses the race still it gets something for playing it, all above features makes interested this racing simulator [**CarRacingSimulator**] game.

3 Requirement specification :

3.1 Functional requirements :

Functional requirements are those that we have to implement in our game project.

- Single and multiplayer Player.
- Android based.
- 3D platform.
- Single level.
- Adventure based.
- 2d GUI and menu system.
- Testing for removing all the bugs in game.
- Written in C sharp for unity.
- Photon cloud.
- Models.
- Power-ups.
- Graphical effects.
- Sounds and music.
- Time integration.

3.1.1 Single and Multiplayer modes :

The game is based on single as well as multiplayer. Single player is for offline and the multi-player feature is added when the user wants to play online with others throughout the world.

3.1.2 Android based :

The game is for android cell phone. We decided to make a game for android due to reason that now a days android phones are going to very popular.

3.1.3 3D platform :

Our game is based on the 3d platform rather than 2d. Initially, we want to integrate both 2d and 3d platform but due to the short time we just implement 3d. But we implement buttons and game cover in 2d.

3.1.4 Single level :

Due to time shortage, we design only one level. If we found enough time then we will implement more than one level.

3.1.5 Adventure based :

Along with action we also made it adventure based. The adventure part of the game is to avoid from hurdles in the track and pick the powers to take

advantage on other cars.

3.1.6 2d GUI and menu system :

The overall game is 3d but the GUI menu and button are in the 2D platform. 3d buttons are non-functional requirements.

3.1.7 Testing :

The game is tested in the proper way until every bug removed. Bugs in the game should be zero.

3.1.8 Written in C sharp :

The game is written in c-sharp code and is integrate into unity 3d platform. Initially, we decided to use unity own cloud but later we decided to use photon cloud to connecting the server.

3.1.9 Photon cloud API's :

We have used Photon cloud API's to integrate Multiplayer features in our game.

3.1.10 Models :

3d models are made in a blender. As blender is easily integrated into unity 3d game engine.

3.1.11 Textures :

The models textures were satisfactory, although they were not always particularly uniform.

3.1.12 Power-ups :

All desired power-ups, such as punch, nitro, coins, and shield were implemented.

3.1.13 Graphical effects :

The important graphical effects that we wanted to use in the game will be added with an appealing visual result.

3.1.14 Sounds and music :

The sound effects that was added satisfactory, was car engine sound, power-up sounds, and coin sounds.

3.1.15 Time Estimation :

In a software development project, one thing that is considered to be very difficult is time estimations. The time estimations in our game project were very good, and we managed to produce a playable game within the given time limit.

3.2 Non-functional requirements :

Non-functional requirements are those that are not necessary to be implemented in the project if time found then it may be added to the project later. Following are the non-functional requirements of our project.

- Ios build.
- multi-level
- Creation of 3D models
- Using unity cloud
- Drifting mechanism.
- Facebook Leaderboard.
- In app purchases.
- Google play services.

4 Project design and Implementation Constraints:

Game development methodology refers to the series of techniques and processes that we used to develop a game. A selection of a right process model is very crucial for the successfully development of the game, we have to keep few factors in mind before choosing the software development life cycle.

- How much time and resources we have.
- Does all the requirements are clear and Can we expect more requirements to be added in future?

4.1 Development process :

In a software development process, the resulting product is required to fulfill many different qualities. Examples of such quality requirements are Robustness, availability, maintainability, dependability and usability. To meet such varying demands, it is important to base the work on a well-prepared strategy. This strategy is called software process, which is built on one or several software process models.

4.2 Software process model :

Software process model is also known as software development life cycle is a process used for designing, developing and testing high quality in the software industry. It targets to produce high quality software to meet the expectations of their valued customers, to complete the software process within the given time span and estimates the cost.

4.3 Software development life cycle :

A game project goes through a process in an organization that is called SDLC. The process is consist of detailed plan defining, how to maintain, alter, develop, replace and enhance particular software. The lifecycle follows certain methods in order to assure and achieve quality as well as a complete development process. There are altogether 6 stages or phases of the software development life cycle.

- **Communication and analysis for requirements :**

  Analysis of requirements is the most important phase. Many people are involved in this process and only highly professional and experienced seniors can perform it. There are a number of things that are kept in mind before proceeding that is market surveys, experts of the domain in the industry. The process further undergoes further into conducting product availability and that too in best economical, technical and operational way. The quality assurance, risk is also the part of planning.

- **Planning and detailing the requirements :**

  Once the analysis is done, the next step comes to give details of the product and document all the requirements and get them authorized. These documentations are done by the help of SRS. Software requirement specification consists of all the details and requirement of the product.

- **Modeling and Architecture the Product design :**

  According to the requirements listed in SRS, more than one design is being proposed and approached for product architect. And is later doc-

umented in the software DDS. The Design document specification is examined by all the stakeholders. Considering product robustness, budget and time restraint, risk assessment and design balance the best design is being selected. The approached design clearly shows the architecture and flow of communication and flow of data representation with the third party.

- **Construction and Implementation :**

  The actual product is built in this stage. The code generation is performed as executed by DDS. Developers have to follow the code guidelines. The programming language is chosen according to the type of software being developed.

- **Product testing :**

  This is the subset of all other phases. Testing activities are done in almost all the phases of SDLC. In this stage defected products are reported, retested and fixed until the required quality is satisfied as mentioned by SRS.

- **Maintenance and distribution of product in the market :**

  Then here comes the process of distribution. Once the quality is tested and assured, the product is deployed. Sometimes the distribution is done under the instruction of the organization itself. Sometimes the company only releases the product in limit and wait for the test result from real business environment.

Then on the demand the product is further released, according to the market value and feedback. Once the product is out in the market its maintenance is done on individual customer base.

These were the phases and life cycle of software development. A software company undergoes many stages in order to bring the best for their customers or organization.

4.4 Incremental process methodology :

In the incremental model, the incremental process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

An incremental life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

The good thing about the incremental module is that you don't have to wait to develop and integrate all the features at once in order to publish the game. As each module is self-built and independent and is not dependent on other modules, therefore, we can publish the game with the most basic features and later keep adding the enhanced features. Incremental process keeps going until all the modules and features are integrated and all the desired requirements are implemented.

All iteration goes through a phase of requirement, coding, design and testing.

Each new version release adds up to the new features in a game which was previously released.

The system is put under production where the first increment is produced. Core products and basic requirements can be found in the first iteration. Later supplementary features are added to it. Once the client is done by analyzing the core product, the plan for another increment is developed.

Characteristics of incremental model:

- The final system is built once the partial systems are built successfully.
- First thing first, the top most priority is handled first, and then moves to the other one.

4.5 Our proposed model for developing project:

In this gaming project, we had choose incremental development model for the development of our project. The decision was mainly based on the size and the duration of the project. We had a limited amount of time, we have to give a round about 4 months to complete this project, and during this time we wanted a rapid development.

We planned our project over a period of 4 months and divided it into four phases. We planned the first phase of game project planning, the second phase of game design, a third phase for coding and the final phase of testing and finalizing the product.

- **First Phase (Project Planning) :**

  The plan is the first step that has to be done with more carefully because all the other work depends upon your plan if the plan is not good enough

than the game will not do well and as a result it will get flopped. Successful completion of a project is heavily dependent on effective planning. In the first iteration, we focused on Project Planning because Successful completion of a project is heavily dependent on effective planning.

- **Second Phase (Designing Game) :**

  One of the most important task is to design the game model. Me and my group member gathered and discussed how to design game. Each of us providing their own opinions and at some requirements one of us was not agreed, it was a difficult situation but we planned and decided the most basic requirements for our gaming project. Each one of us denoted the attributes or properties of the game that we are dreamed of implementing.

  We gathered suggestions together and chose the ones that were possible to be implemented within a 4 months time period. As soon as the game concept became clear, we made some early decisions on basic requirements of the project in order to more easily reach the development goals.

- **Third Phase (game coding) :**

  The third phase contains coding session. In this phase, we decided which one tool should be used. We decided to use Microsoft visual studio for coding purpose. Because we had expertise in c sharp.

- **Fourth Phase (test phase) :**

  Before publishing a game on the Android app store, there was a need

for testing the game to removes errors and bugs. The product must be tested before publishing. The successful testing process of software requires a good plan. Therefore, after the requirements of the project are confirmed, the future testing of the system and the code were planned. Test cases were designed for the planned tests.

4.6 Architecture overview :

An illustration of few components of our gaming project could be seen in a below diagram.

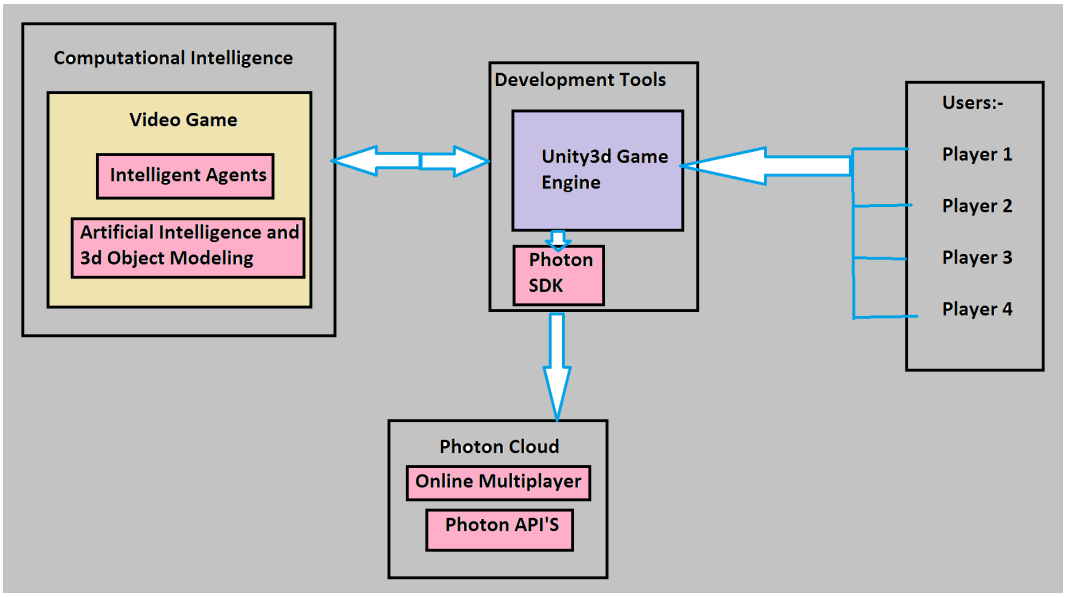

Figure 1: Components

An overview of the conceptual framework of this project, in which the four main parts of the Racing video game are demonstrated:

- Computational Intelligence :
- Development Tools :
- Photon Cloud :

4.6.1 Computational Intelligence :

Artificial intelligence is used make intelligent behavior possible for rival cars.

Intelligent Agents : An Intelligent Agent is one that acts intelligently, its actions are appropriate for its goals and circumstances. It makes appropriate choices given limitations. In this project, In case of online challenge both players are the client users that compete with each other's and follow the rules of the game. In the case of offline, there will be few rival cars which are controlled by the computer, who follows its own rules and analyzes nodes spread all around the track in places, and acts as a hard opponent to defeat.

Artificial intelligence : The AI as a part of Computational Intelligence focuses on providing the capacity to computers to perform tasks that require human intelligence. Therefore, we combined few AI techniques for the development of complex algorithms to solve practical problems in the game.

- **Path Planning :**

  For each computer player we have defined three path patterns on which it can move. Each pattern is consisted of several nodes, whenever computer controlled rival car reaches to the specific particular node then Unity3d

random function will be called and it will switch path and select new path depending upon the random value returned by the random function.

- **Obstacle Detection System :**

  We have used unity3d raycasting system to implement obstacle avoiding system, each computer controlled rival car instantiates 5 raycasts, the main rays are Front Left Ray, Front Center Ray and Front Right Ray. When Front Left Ray collides with the upcoming obstacle the rival cars move right. If the front Right Ray collides with an obstacle the rival car will moves left and if the Front Center Ray will collide with an obstacle the car will move randomly left or right in order to avoid obstacles.

4.6.2 Development tools :

Following developing tools are to be used in our project.

- Unity 3d.
- Visual studio.
- Adobe photoshop.
- Blender.

4.6.3 Photon Cloud :

We are using unity3d photon cloud APIs to implement Multiplayer functionality in our game. We are using a free version of photon cloud which allows maximum of 20 concurrent players to play game at same time. We will buy premium if our project will become successful.

4.7 Design description :

We were planning to develop a multiplayer racing game, with time the game concept evolved, as more and more features were added. To further explain how the game concept evolved, the development has been divided into three parts.

4.7.1 Initial concept :

Everyone plays game only for fun and entertainment, our gaming project was inspired by the concept of the famous movie death race, where each player had a car equipped with a weapon, and the goal was to hunt down the opponents cars. The other idea was to create a simple race track where players could collect coins in order to win. Since the first idea involved much greater work than the latter one, it was decided that the first idea were to be given a low priority, whereas the second were to be given a high priority.

4.7.2 Second concept :

We decided that a joint solution would be best, where players are able to collect coins and then use the money to make upgrades. An example of upgrades was enhancement of engine, nitro, shield and brakes. It was also decided that the racing track should be in a forest, filled with boxes, rocks, meteorites and rotating gates. And miscellaneous objects suitable for that environment.

4.7.3 Final concept :

We kept most of the ideas from the second iteration, however, a racing game where players only may collect coins and nothing more sounded a bit boring.

Influenced by games like Super Mario Kart, it was decided that power-ups should be added. There are a number of different power-ups, likes nitro, shield and mega punch. These power ups makes the game interesting and fun to play.

4.7.4 Project scope :

Because this project contains the features of the best racing games ever made, this project have a good scope of success, gaming industry is a really profitable industry, this game have a huge potential of earning revenue through google ads and in-app purchases. This Project scope covers all functional requirements that we are willing to add on our game project. The project will be based on creating a multiplayer racing video game and our goal is to make an interactive, fun and entertaining game.

- Single and multiplayer Player
- Android based
- 3D platform
- Single level
- Adventure based
- 2d GUI and menu systems
- Photon cloud.
- Power-ups.
- Hurdles.
- Proper Testing.

4.7.5 Gameplay :

The racing game provides a mixture of power-ups and hurdles like obstacles which includes boxes, meteorites and rotating gates and power-ups includes nitro, shield, coins and mega punch. The only goal is to cross the finish line first to win the game against your opponent.

The game has a design with only single-player as well as multiplayer mode. The single player mode is only available in offline practice where the multi-player mode is for when connecting to the server to compete with other throughout the world. In the single-player mode one can find three sub-modes: easy, normal and hard. Depends on sub-modes, the number of hurdles and pick-ups will vary. When the game becomes harder, the computer players capabilities enhances.

4.7.6 Level design :

We designed track in a forest. Currently, we made only one level but later we will develop other levels. In each level powers and pickups vary. Players can enjoy different powers on each level. We will add levels in which there exist additional powers

4.7.7 Game modes :

Game modes are only for single player for offline practice not for multiplayer. There exist three types of modes:

- Easy
- Normal
- Hard

Depends on sub-modes, the computer controlled player capabilities will vary.

4.7.8 Controls of the game :

The game utilizes the GUI buttons shown on your android cell phone in your hand. The attacks are done by touching the screen of your android phone. Here is the list how to control the game.

4.7.9 Movements :

Move Left: Touch the GUI texture of Left Arrow.

Move Right: Touch the GUI texture of Right Arrow.

Break: Touch the GUI texture of a Brake.

Acceleration: Touch the GUI texture of Accelerator.

Reverse: Touch and hold on the GUI texture of a Brake.

4.7.10 Winning and Losing :

The objective of the game is to cross the finish line of the track before the opponent player.

4.7.11 Power-ups :

Power-ups will be collectible objects that are placed on different points on the terrain.

Nitro: The players car moves faster. Nitro power provides the ability to move faster than the normal speed up to 3 times faster.

Punch: Punch the other car and reduces the Speed of the opponent car.

Shield: Save player car from obstacles..

4.7.12 System requirements :

System must have following minimum requirements:

- Android phone OS version ¿= 4.2
- Ram 1gb at least

5 Game Design and Implementation :

5.1 Screen Shots and Working

In this section we have added few screenshots of our gaming project and we will explain working and some functionality of a game in this section.

5.1.1 Enter Name Panel :

When game initializes, Name panel shown to the user in which he/she enters his name and then continue towards main menu.

Figure 2: Enter Name Panel

User enter his/her name in this panel.

- If user enter something in the field, then it will be assigned to the name string otherwise If user didn't enter anything in the field, then "Guest" will be assigned to the name string.

5.1.2 Main Menu :

The main menu of the game contains Five Buttons.

- Play
- Multiplayer
- Upgrade
- Store
- Quit

Figure 3: Main Menu Image

- If user clicks Play button then the game modes menu will be shown.
- If user clicks Multiplayer button then the user will connect to the multiplayer lobby.

- If user clicks the Upgrade buttons then upgrades panel will be shown.
- If user clicks the Store buttons then Market panel will be shown.
- If user clicks Quit button then game will quit.

5.1.3 Store Menu :

In a Store menu user can perform three main actions which includes changing cars by pressing next button and previous button, user can purchase car to unlock it, user can select unlocked cars.

Figure 4: Store Menu

Purchase new car : In this menu, we provide the facility to user to purchase new cars with more features and functions. By clicking left and right arrow we can change existing car and then by clicking on buy button we can purchase the new car in the market menu.

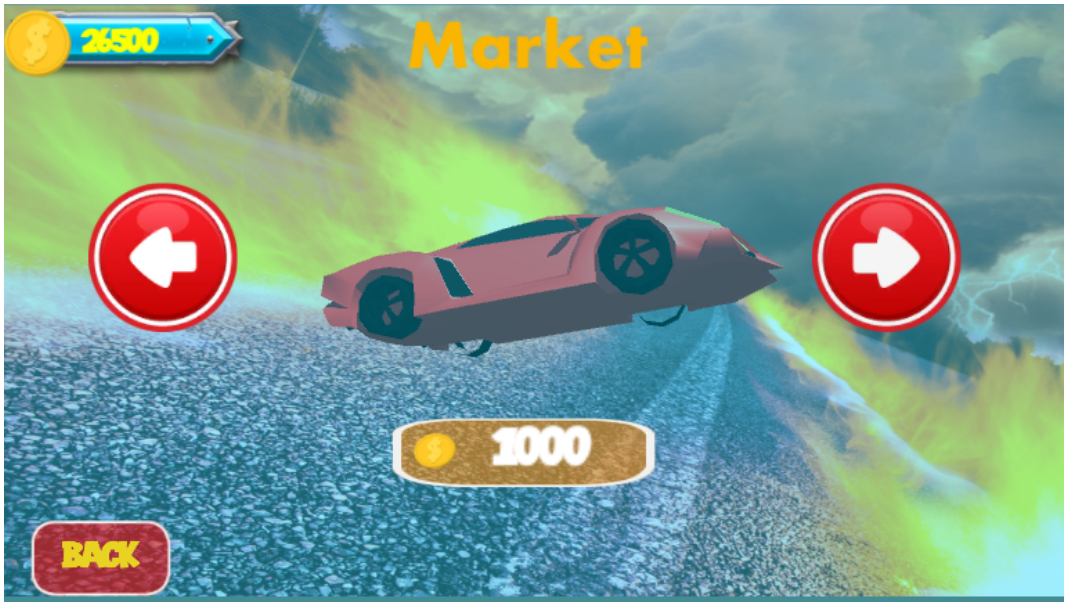

Figure 5: Purchase Car

Select unlocked cars : In this menu, we provide the facility to user to select already unlocked cars with more features and functions. By clicking left and right arrow we can change existing car and then by clicking on select car button we can select the new car in the market menu.

Figure 6: Select Car

5.1.4 Upgrade Menu :

Upgrade menu of the game includes following upgrades:

Figure 7: Upgrade Menu

We can upgrade any given items by using coins for each upgrade level.

Nitro upgrade : Nitro can be upgraded by spending coins. Up gradation level of nitro is four. By spending 25 coins for first level updating we can upgrade nitro by 25 percent. In each up gradation level, we enhance the time for nitro release.

Shield upgrade : Like nitro upgrade mechanism, shield mechanism works in the same way. It can also upgrade by spending 25 coins for first level updating we can upgrade shield by 25 percent. Shield can be upgraded at the cost of 25 coins. Up gradation level of the shield is four. In each up gradation level, we enhance the time for shield protection.

Figure 8: Nitro Upgrade

Figure 9: Shield Upgrade

Engine upgrade : Engine up gradation also has four levels, in each update level engine optimization increased by 25 times. Each level can be upgraded by spending coins. Coins can be picked up on the road.

Brakes upgrade : Brakes mechanism can also be upgraded in the upgrade menu. It can also upgrade by spending coins in each level.

Figure 10: Engine Upgrades

Figure 11: Brakes Upgrades

5.1.5 Game Modes :

We introduced three modes of the game. Difficulty level enhances in the mode to mode.

Figure 12: Game Modes

Easy : It is the easiest mode of the game. In this mode, the player learns how to play the game. Before going to difficult mode the beginner plays the game in the easiest mode to understand how the game works and how to play the game.

Normal : After easy mode next level is designed for a professional player. This mode designed some difficult than easy mode. Enemy upgrades are enhanced in this mode.

Hard : In hard mode, Enemy upgrades are even more enhanced in this mode. The player that play the game in this mode, we ranked him as an expert player.

5.1.6 Player Car Controls GUI :

Car mechanics consists of five actions.

Figure 13: Car Controls

Move left : User will tap on a left arrow to move left.

Move right : User will tap on a right arrow to move right.

Move forward : User will tap on an accelerator button located at bottom right to move forward.

Brakes : User will tap on a brake button located at bottom center to apply brakes

Reverse : User will tap and hold on a brake button for few seconds to reverse car.

5.1.7 Game sounds :

We have integrated awesome background musics, Different power ups have different sounds, when we catch coins we can listen the treasure caching sound when we caught nitro we will hear rocket like sound until the nitro has timed out, when we catch shield we hear collection sound, during shield time when we hit the obstacles we can hear blast sound.

Background Music : we have integrated different background musics for menu screens and game playing field.

Shield pick up : When we catch shield we hear collection sound, during shield time when we hit the obstacles we can hear blast sound.

Nitro pick up : When we caught nitro we will hear rocket like sound until the nitro has timed out.

Obstacle collision : When we collide with obstacles, the car stops and the collision sound produce that indicate the car collide with an obstacle.

Menu Buttons : Briefcase open like sound could be heard when we touch buttons.

5.1.8 Enemy car AI :

We applied Artificial intelligence for rival cars. The rival cars find the path. For path finding algorithm we defined three or four paths on which rival cars randomly switch from one to other paths.

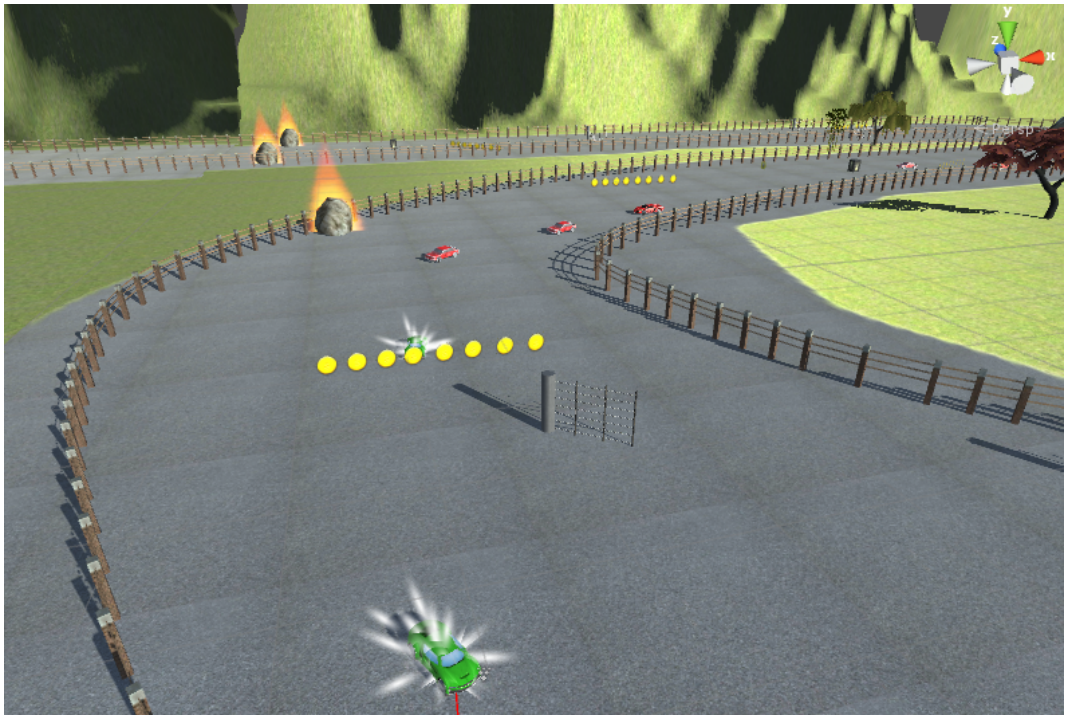

Figure 14: Enemy AI

Path points : We defined multiple points on the track. Path finding by rival cars depends on upon these path points that are spread on the track at different locations.

Rival car nitro and shield mechanism : Like a player car, the rival car can also pick nitro and shield and these powers work for the specific given time.

Enemy car power-up collision detection system : Enemy cars can detect the powers and picks up all the provided powers except coins. These powers work the same way as work for the player car.

5.1.9 Power-up's :

Nitro : Nitro increases the speed of the car for some time when it released. After finish the given time the cars runs with the normal speed as before.

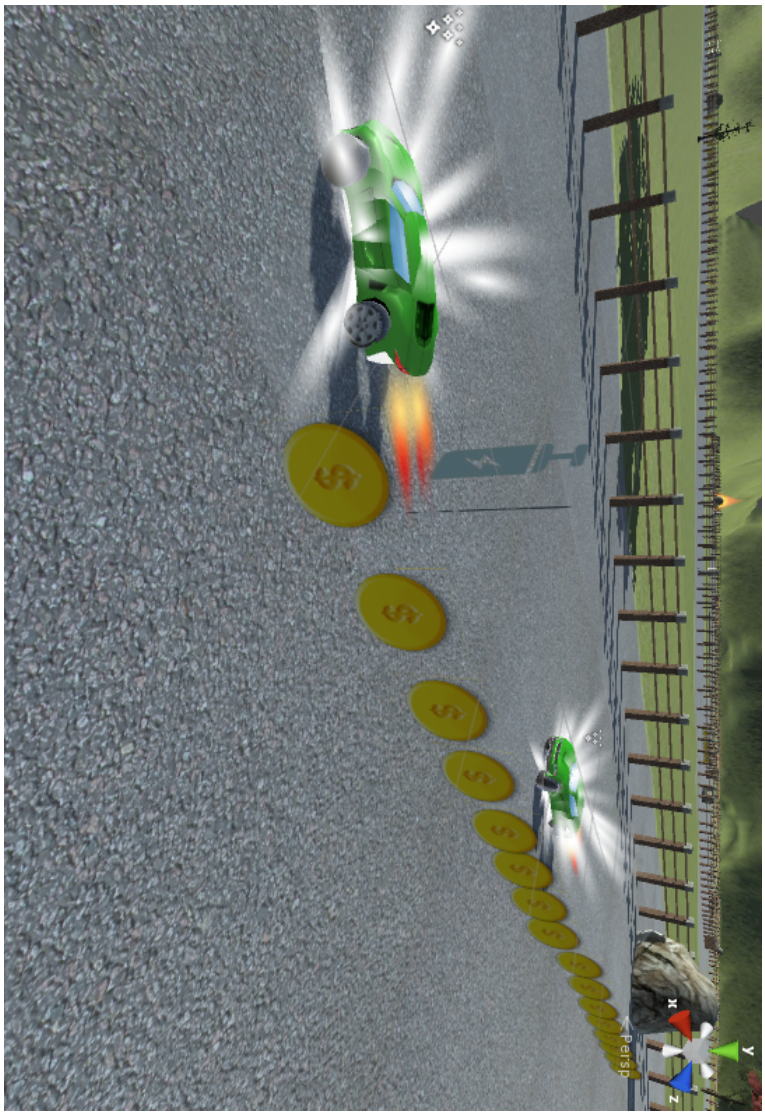

Figure 15: Nitro

Mega Punch : Punch hits the other cars and make them poisoned. It forces the other car to move slowly and and after few seconds it effect vanishes.

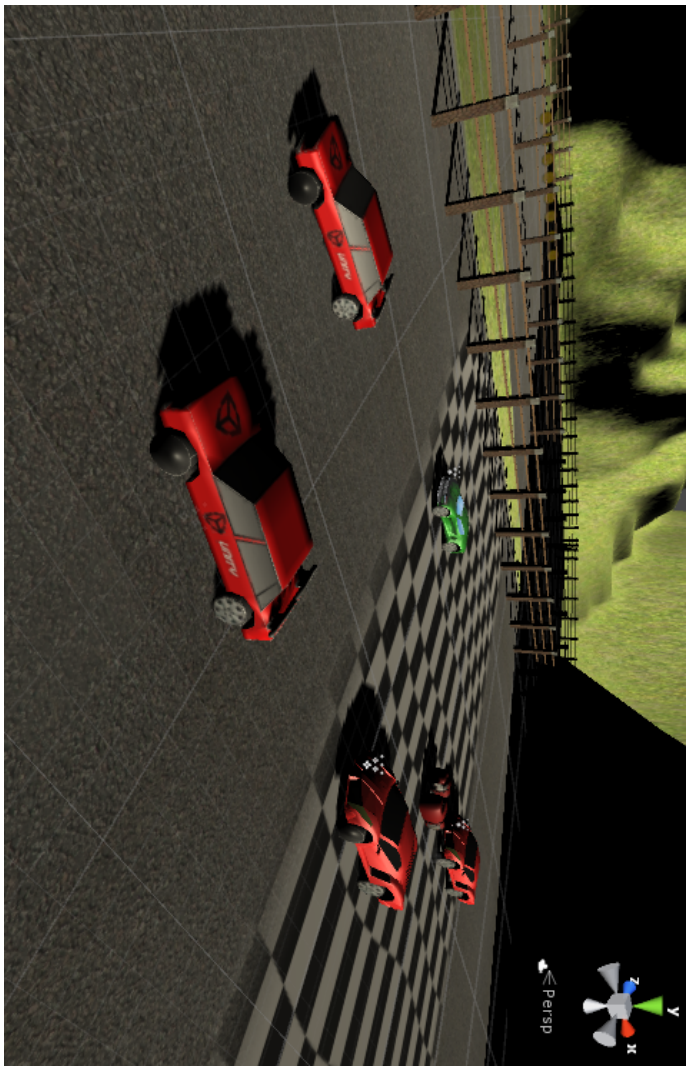

Figure 16: Punch

Shield : Shield protects the car from any kind of obstacles that is rocks, fences, rotating doors. It the protection mirror that saves the car from the crash.

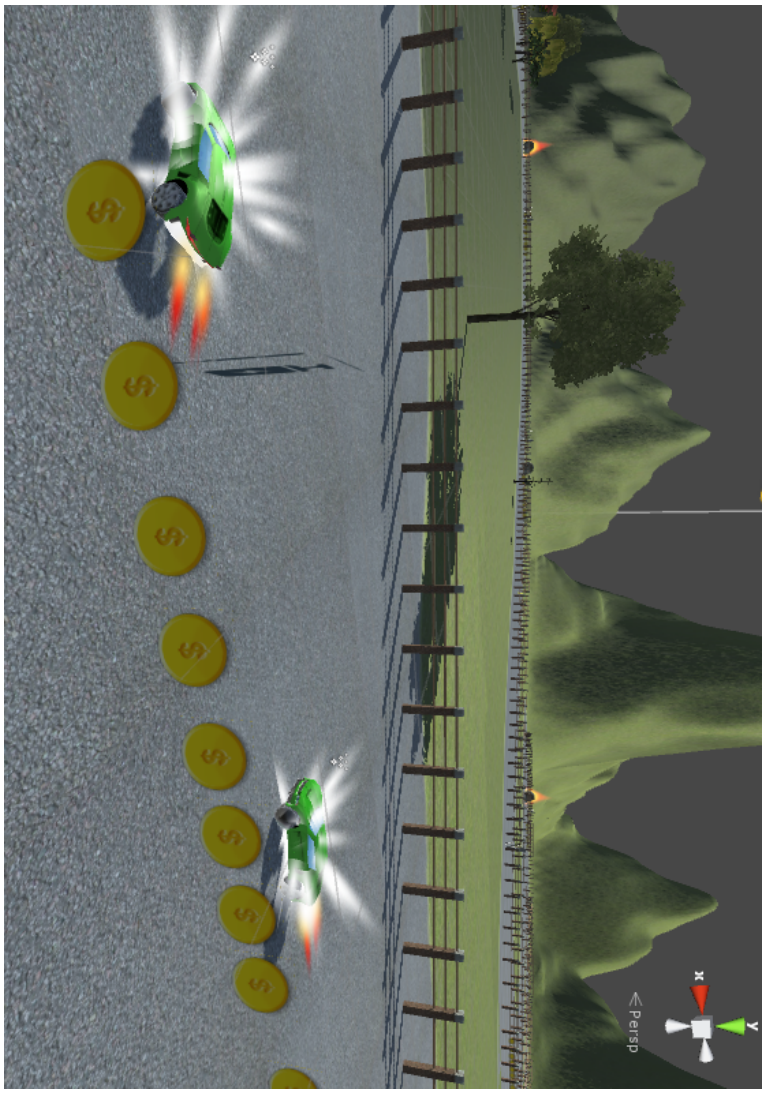

Figure 17: Shield

5.1.10 Obstacles :

Fences : Fence covers the road track on both sides. Fences are the boundaries of the track so that car remains within the track.

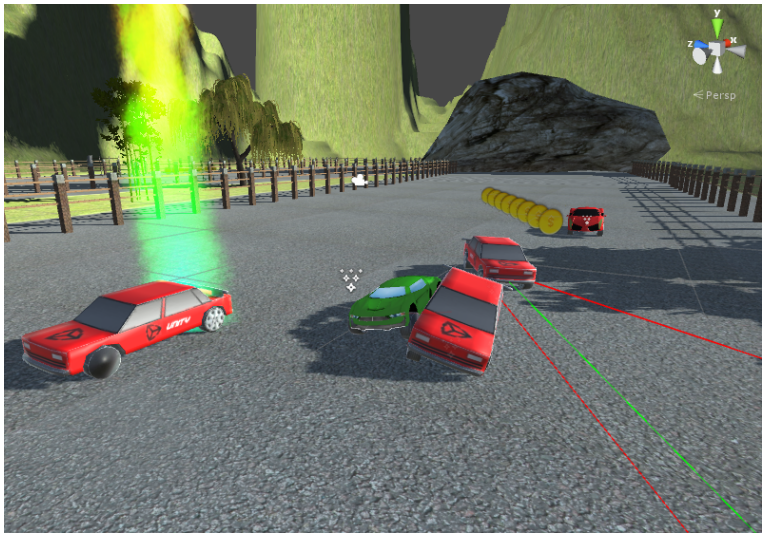

Figure 18: Fences

Poison box : Poison box is kept in track when car hits that poison box then the poison powers added to the car. At any time player can release the power that poisons others cars.

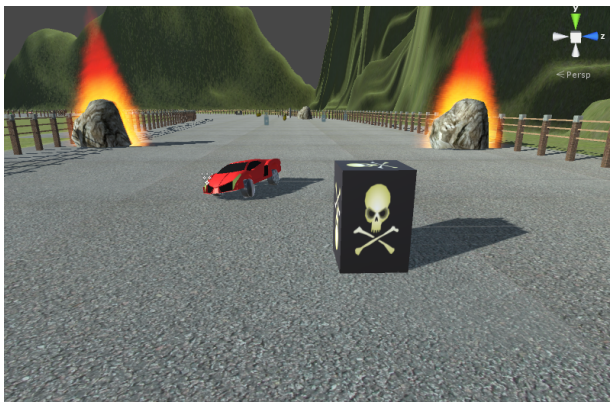

Figure 19: PoisonBox

Rocks : Rocks are obstacles placed on the track. When a car collides with rocks it stops running and increases the chances for the player to lose the game.

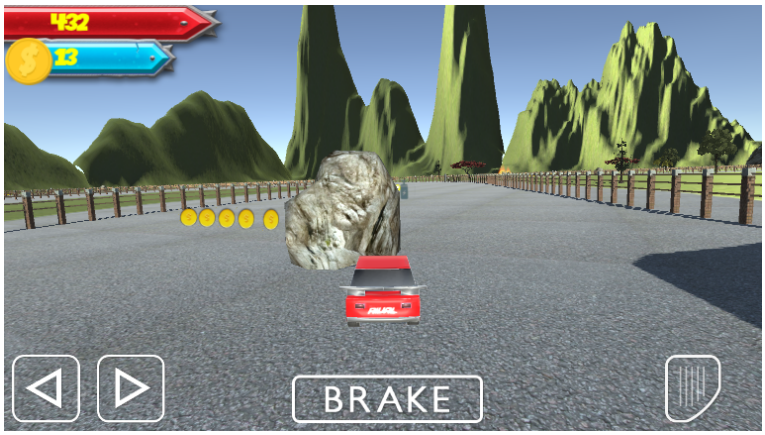

Figure 20: Rocks

Meteorites : Meteorites are similar to rocks but they are larger and show flame.

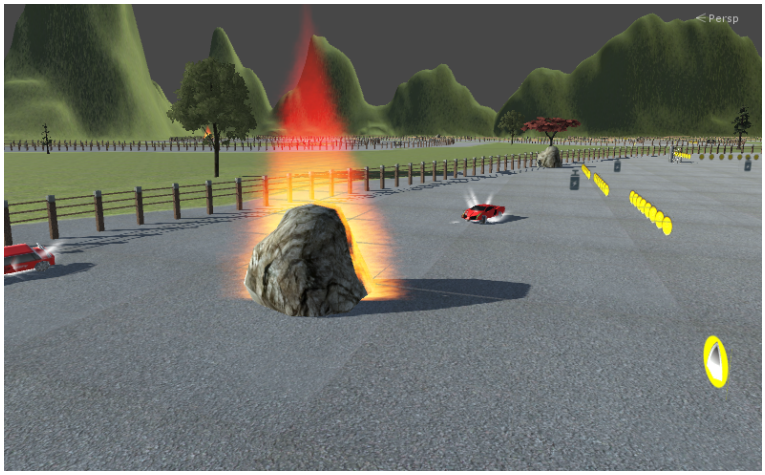

Figure 21: Meteorites

Rotating doors : Rotating doors the hurdles placed randomly in track. These doors continuously rotate. If a car collides them the car stops and these doors push back the car.

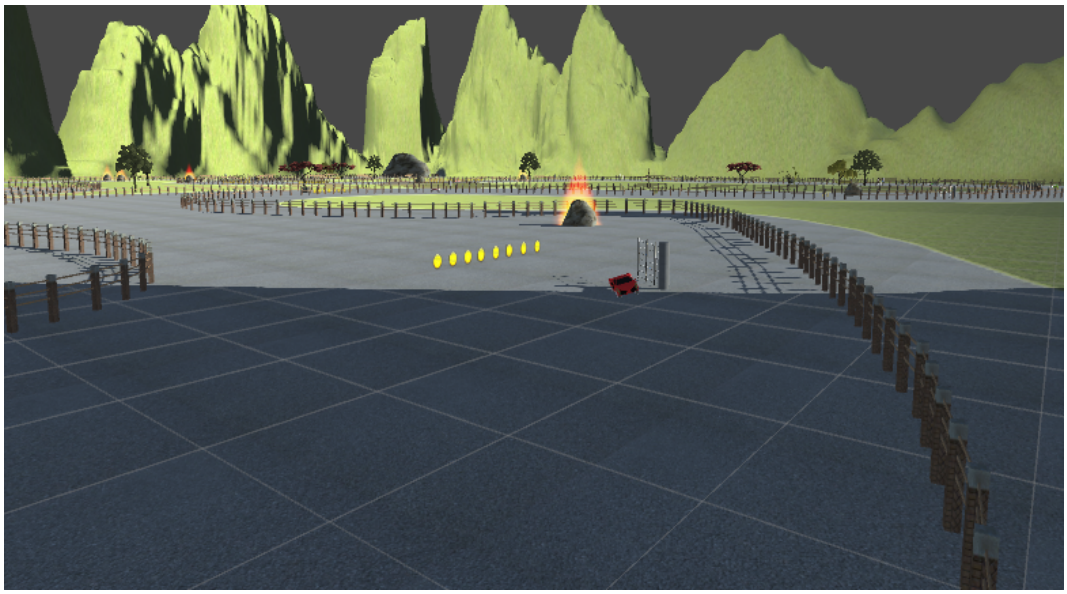

Figure 22: Rotating doors

5.1.11 MultiPlayer :

After connecting to the photon server you will enter in a multiplayer lobby where you can create a new room or can join existing room to play agains the players all around the world.

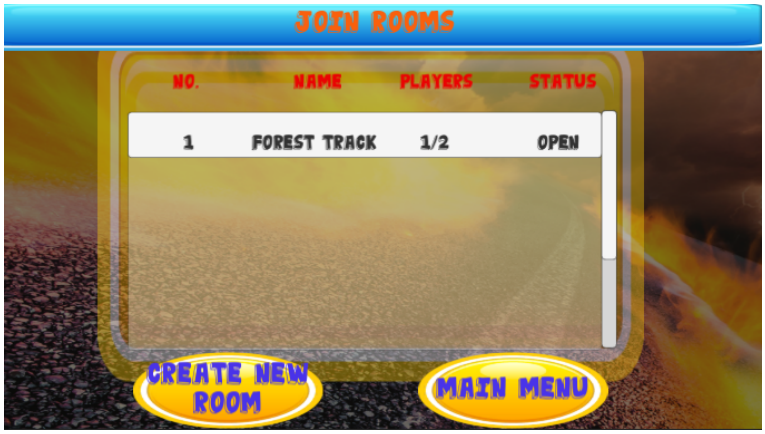

Figure 23: Lobby

Figure 24: Room Name

Figure 25: Creating Room

Figure 26: Waiting For Players to join

5.2 Few Important Scripts :

Here are some of the few important scripts related to the working of this game.

5.2.1 Player Car Script :

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
public class PlayerCarScript : MonoBehaviour {
public bool EngineSoundBoostFlag;
public float EngineSoundAdjustment;
public bool BrakeSoundBoostFlag;
public float BrakeSoundAdjustment;
public float axleSpeed;
public float Adjustment;
public float maxSteer;
public float tempSteer;
public float currentSteerValue;
public Vector3 centerOfMass;
public WheelCollider WheelFL;
public WheelCollider WheelFR;
public WheelCollider WheelRL;
public WheelCollider WheelRR;
public float maxTorque;
public float currentSpeed;
public float topSpeed;
public float decellarationSpeed;
```

```
public Renderer breakingMesh;
public Material idleBreakLight;
public Material activeBreakLight;
public bool isBreaking;
public bool inSector;
public bool reversing;
public float reversingCounter;
public float waitToReverse = 2.0f;
public float reverseFor = 1.5f;
public bool movementFlag;
public bool rightSteerFlag;
public bool leftSteerFlag;
public bool breakingFlag;
public float steerSpeedController;
public int CarPosition;
public AudioSource audioEngine;
public AudioSource audioSkid;
void ActivateControls()
{
if (MultiPlayerSceneFlag.multiPlayerFlag == false) {
PointerTrackerAccelarator.obj =
    gameObject.GetComponent<PlayerCarScript> ();
PointerTrackerBrake.obj = gameObject.GetComponent<PlayerCarScript>
    ();
PointerTrackerLeftArrow.obj =
    gameObject.GetComponent<PlayerCarScript> ();
PointerTrackerRightArrow.obj =
    gameObject.GetComponent<PlayerCarScript> ();
}
```

```
} // End Activate Control
// Use this for initialization
void Start () {
EngineSoundBoostFlag = false;
EngineSoundAdjustment = 0f;
audioEngine.volume = 0f;
BrakeSoundBoostFlag = false;
BrakeSoundAdjustment = 0f;
audioSkid.volume = 1f;
axleSpeed = 0f;
Adjustment = 0f;
tempSteer = maxSteer;
audioEngine.Play ();
gameObject.name = "PlayerCarObject";
CarPosition = 0;
// For Engine Upgrades
if (PlayerPrefs.GetInt ("EngineFilledValue", 0) == 1)
{
maxTorque = maxTorque + 100f;
topSpeed = topSpeed + 50;
}
if (PlayerPrefs.GetInt ("EngineFilledValue", 0) == 2)
{
maxTorque = maxTorque + 150f;
topSpeed = topSpeed + 75;
}
if (PlayerPrefs.GetInt ("EngineFilledValue", 0) == 3)
{
maxTorque = maxTorque + 200f;
```

```
topSpeed = topSpeed + 100;
}
if (PlayerPrefs.GetInt ("EngineFilledValue", 0) == 4)
{
maxTorque = maxTorque + 250f;
topSpeed = topSpeed + 125;
}
// For Brakes Upgrades
if (PlayerPrefs.GetInt ("BrakesFilledValue", 0) == 1)
{
decellarationSpeed = decellarationSpeed + 300f;
}
if (PlayerPrefs.GetInt ("BrakesFilledValue", 0) == 2)
{
decellarationSpeed = decellarationSpeed + 500f;
}
if (PlayerPrefs.GetInt ("BrakesFilledValue", 0) == 3)
{
decellarationSpeed = decellarationSpeed + 700f;
}
if (PlayerPrefs.GetInt ("BrakesFilledValue", 0) == 4)
{
decellarationSpeed = decellarationSpeed + 1000f;
}
// End Upgrades
ActivateControls ();
movementFlag = false;
rightSteerFlag = false;
leftSteerFlag = false;
```

```
breakingFlag = false;
steerSpeedController = 0;
currentSteerValue = 0;
gameObject.GetComponent<Rigidbody> ().centerOfMass = centerOfMass;
}
// Update is called once per frame
void Update () {
axleSpeed = WheelRR.rpm;
Adjustment = 0f;
maxSteer = tempSteer;
currentSpeed = 2 * (22 / 7) * WheelRL.radius * WheelRL.rpm * 60 /
    1000;
currentSpeed = Mathf.Round (currentSpeed);
if (currentSpeed > 1) {
Adjustment = (0.007f * currentSpeed);
maxSteer = maxSteer / Adjustment;
}
// Engine Sound

if (EngineSoundBoostFlag == true)
{
audioEngine.volume = 1f;
}
if (EngineSoundBoostFlag == false)
{
audioEngine.volume = 0.4f;
}

//Brakes Sound
```

```
BrakeSoundAdjustment = 0.009f * currentSpeed;
if (BrakeSoundBoostFlag == true) {
audioSkid.volume = BrakeSoundAdjustment;
}
if (BrakeSoundBoostFlag == false)
{
audioSkid.volume = 0f;
}
steerSpeedController = 1000 - currentSpeed;
}
void FixedUpdate()
{
// For Movement
if (movementFlag == true) {
currentSpeed = 2 * (22 / 7) * WheelRL.radius * WheelRL.rpm * 60 /
    1000;
currentSpeed = Mathf.Round (currentSpeed);
if (currentSpeed <= topSpeed) {
WheelRL.motorTorque = maxTorque;
WheelRR.motorTorque = maxTorque;
WheelRL.brakeTorque = 0;
WheelRR.brakeTorque = 0;
}
if (currentSpeed > topSpeed) {
WheelRL.motorTorque = 0;
WheelRR.motorTorque = 0;
WheelRL.brakeTorque = decellarationSpeed;
WheelRR.brakeTorque = decellarationSpeed;
}
```

```
}
// For Breaking and Reversing
if (breakingFlag) {
breakingMesh.material = activeBreakLight;
WheelRL.motorTorque = 0;
WheelRR.motorTorque = 0;
WheelRL.brakeTorque = decellarationSpeed + 1000f;
WheelRR.brakeTorque = decellarationSpeed + 1000f;
if (gameObject.GetComponent<Rigidbody> ().velocity.magnitude <= 0.5)
    {
reversingCounter += Time.deltaTime;
if (reversingCounter >= waitToReverse) {
reversingCounter = 0;
reversing = true;

}
}
if(reversing == true)
{
WheelRL.motorTorque = -(maxTorque * 0.2f) ;
WheelRR.motorTorque = -(maxTorque * 0.2f);
WheelRL.brakeTorque = 0f;
WheelRR.brakeTorque = 0f;

}
}
if (Input.GetKeyDown (KeyCode.W))
{
EngineSoundBoostFlag = true;
```

```
audioEngine.Play ();
movementFlag = true;
}
if (Input.GetKeyUp (KeyCode.W))
{
EngineSoundBoostFlag = false;
movementFlag = false;
WheelRL.motorTorque = 0f;
WheelRR.motorTorque = 0f;
WheelRL.brakeTorque = 0f;
WheelRR.brakeTorque = 0f;
}
if (Input.GetKeyDown (KeyCode.S))
{
BrakeSoundBoostFlag = true;
audioSkid.Play();
breakingFlag = true;
}
if (Input.GetKeyUp (KeyCode.S))
{
BrakeSoundBoostFlag = false;
audioSkid.Stop ();
breakingFlag = false;
breakingMesh.material = idleBreakLight;
WheelRL.motorTorque = 0f;
WheelRR.motorTorque = 0f;
WheelRL.brakeTorque = 0;
WheelRR.brakeTorque = 0;
reversing = false;
```

```
}
if (Input.GetKeyDown (KeyCode.A))
{
currentSteerValue = -maxSteer;
WheelFL.steerAngle = currentSteerValue;
WheelFR.steerAngle = currentSteerValue;
}
if (Input.GetKeyUp (KeyCode.A))
{
WheelFL.steerAngle = 0f;
WheelFR.steerAngle = 0f;
}
if (Input.GetKeyDown (KeyCode.D))
{
currentSteerValue = maxSteer;
WheelFL.steerAngle = currentSteerValue;
WheelFR.steerAngle = currentSteerValue;
}
if (Input.GetKeyUp (KeyCode.D))
{
WheelFL.steerAngle = 0f;
WheelFR.steerAngle = 0f;
}
}
public void OnAccelaratorDown()
{
EngineSoundBoostFlag = true;
audioEngine.Play ();
movementFlag = true;
```

```
}
public void OnAccelaratorUp()
{
EngineSoundBoostFlag = false;
movementFlag = false;
WheelRL.motorTorque = 0f;
WheelRR.motorTorque = 0f;
WheelRL.brakeTorque = 0f;
WheelRR.brakeTorque = 0f;
}
public void OnLeftArrowDown()
{
currentSteerValue = -maxSteer;
WheelFL.steerAngle = currentSteerValue;
WheelFR.steerAngle = currentSteerValue;
}
public void OnLeftUp()
{
WheelFL.steerAngle = 0f;
WheelFR.steerAngle = 0f;
}
public void OnRightArrowDown()
{
currentSteerValue = maxSteer;
WheelFL.steerAngle = currentSteerValue;
WheelFR.steerAngle = currentSteerValue;
}
public void OnRightUp()
{
```

```
WheelFL.steerAngle = 0f;
WheelFR.steerAngle = 0f;
}
public void OnBrakeDown()
{
BrakeSoundBoostFlag = true;
audioSkid.Play();
breakingFlag = true;
}
public void OnBrakeUp()
{
BrakeSoundBoostFlag = false;
audioSkid.Stop ();
breakingFlag = false;
breakingMesh.material = idleBreakLight;
WheelRL.motorTorque = 0f;
WheelRR.motorTorque = 0f;
WheelRL.brakeTorque = 0;
WheelRR.brakeTorque = 0;
reversing = false;
}
void Move()
{
currentSpeed = 2 * (22 / 7) * WheelRL.radius * WheelRL.rpm * 60 /
    1000;
currentSpeed = Mathf.Round (currentSpeed);
if (currentSpeed <= topSpeed) {
if (reversing == false) {
WheelRL.motorTorque = maxTorque;
```

```
WheelRR.motorTorque = maxTorque;
}
if(reversing == true)
{
WheelRL.motorTorque = -maxTorque;
WheelRR.motorTorque = -maxTorque;
}
WheelRL.brakeTorque = 0;
WheelRR.brakeTorque = 0;
}
if (currentSpeed > topSpeed) {
WheelRL.motorTorque = 0;
WheelRR.motorTorque = 0;
WheelRL.brakeTorque = decellarationSpeed;
WheelRR.brakeTorque = decellarationSpeed;
}
}
void BreakingEffect()
{
if (breakingFlag) {
breakingMesh.material = activeBreakLight;
WheelRL.motorTorque = 0;
WheelRR.motorTorque = 0;
WheelRL.brakeTorque = decellarationSpeed;
WheelRR.brakeTorque = decellarationSpeed;
if (gameObject.GetComponent<Rigidbody> ().velocity.magnitude <= 0.5)
    {
reversingCounter += Time.deltaTime;
if (reversingCounter >= waitToReverse) {
```

```
reversingCounter = 0;
reversing = true;

}
}
if(reversing == true)
{
WheelRL.motorTorque = -maxTorque;
WheelRR.motorTorque = -maxTorque;
}
}
else
{
reversing = false;
breakingMesh.material = idleBreakLight;
WheelRL.motorTorque = maxTorque;
WheelRR.motorTorque = maxTorque;
WheelRL.brakeTorque = 0;
WheelRR.brakeTorque = 0;
}
}
}
```

5.2.2 AI Car Script :

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
```

```
public class AICarScript : MonoBehaviour {
public List<Transform> path;
public List<Transform> path2;
public List<Transform> path3;
public Transform pathGroup;
public int CurrentPathDecidingCounter;
public float maxSteer;
public Vector3 centerOfMass;
public WheelCollider WheelFL;
public WheelCollider WheelFR;
public WheelCollider WheelRL;
public WheelCollider WheelRR;
public int currentPathObj;
public float distFromPath = 20f;
public float maxTorque;
public float currentSpeed;
public float topSpeed;
public float decellarationSpeed;
public Renderer breakingMesh;
public Material idleBreakLight;
public Material activeBreakLight;
public bool isBreaking;
public bool inSector;
public float sensorLength;
public float sidewaySensorsLength;
public float frontSensorStartPoint;
public float frontSensorSideDistance;
public float frontSensorsAngle = 30f;
private int flag = 0;
```

```
public float avoidSpeed = 10;
public bool reversing;
public float reversingCounter;
public float waitToReverse = 2.0f;
public float reverseFor = 1.5f;
public float respawnWait = 5f;
public float respawnCounter = 0.0f;
public bool DebugSensorsFlag;
public float sensorLengthOriginal;
public float sidewaySensorLengthOriginal;
public float RayOriginateHeight;
public int upgradeValue;
public int difficultyValue;
public float tempSteer;
public int CarPosition;
//public float dir;
// Use this for initialization
void Start () {
CarPosition = 0;
CurrentPathDecidingCounter = 1;
difficultyValue = PlayerPrefs.GetInt ("DifficultyLevel", 1);
if (difficultyValue == 1) {
upgradeValue = Random.Range (0, 3);
}
if (difficultyValue == 2) {
upgradeValue = Random.Range (2, 4);
}
if (difficultyValue == 3) {
upgradeValue = Random.Range (3, 5);
```

```
}
// For Engine Upgrades
if (upgradeValue == 1)
{
maxTorque = maxTorque + 100f;
topSpeed = topSpeed + 50;
}
if (upgradeValue == 2)
{
maxTorque = maxTorque + 150f;
topSpeed = topSpeed + 75;
}
if (upgradeValue == 3)
{
maxTorque = maxTorque + 200f;
topSpeed = topSpeed + 100;
}
if (upgradeValue == 4)
{
maxTorque = maxTorque + 250f;
topSpeed = topSpeed + 125;
}
// For Brakes Upgrades
if (upgradeValue == 1)
{
decellarationSpeed = decellarationSpeed + 300f;
}
if (upgradeValue == 2)
{
```

```
decellarationSpeed = decellarationSpeed + 500f;
}
if (upgradeValue == 3)
{
decellarationSpeed = decellarationSpeed + 700f;
}
if (upgradeValue == 4)
{
decellarationSpeed = decellarationSpeed + 1000f;
}
// End Upgrades
sensorLengthOriginal = sensorLength;
sidewaySensorLengthOriginal = sidewaySensorsLength;
gameObject.GetComponent<Rigidbody> ().centerOfMass = centerOfMass;
GetPath ();
tempSteer = maxSteer;
}
// Update is called once per frame
void Update () {
currentSpeed = 2 * (22 / 7) * WheelRL.radius * WheelRL.rpm * 60 /
    1000;
currentSpeed = Mathf.Round (currentSpeed);
if (currentSpeed > 70) {
maxSteer = tempSteer / 2;
} else if (currentSpeed < 40) {
maxSteer = tempSteer * 3;
}
else if(currentSpeed >= 40 && currentSpeed <= 70)
{
```

```
maxSteer = tempSteer;
}
if (flag == 0) {
GetSteer ();
}
Move ();
BreakingEffect ();
Sensors ();
Respawn ();
Debug.Log ("Torque = " + WheelRL.motorTorque);
Debug.Log ("BrakeTorque = " + WheelRL.brakeTorque);
}
void GetPath()
{
pathGroup = GameObject.Find ("Path_Track").transform;
Transform[] path_objects =
    pathGroup.GetComponentsInChildren<Transform> ();
path = new List<Transform> ();
foreach(Transform path_obj in path_objects)
{
if (path_obj != pathGroup) {
path.Insert (path.Count, path_obj);
}
}
pathGroup = GameObject.Find ("Path_Track2").transform;
path_objects = pathGroup.GetComponentsInChildren<Transform> ();
path2 = new List<Transform> ();
foreach(Transform path_obj in path_objects)
{
```

```
if (path_obj != pathGroup) {
path2.Insert (path2.Count, path_obj);
}
}
pathGroup = GameObject.Find ("Path_Track3").transform;
path_objects = pathGroup.GetComponentsInChildren<Transform> ();
path3 = new List<Transform> ();
foreach(Transform path_obj in path_objects)
{
if (path_obj != pathGroup) {
path3.Insert (path3.Count, path_obj);
}
}
}
void GetSteer()
{
Transform temp1 = path [currentPathObj];
if (CurrentPathDecidingCounter == 1) {
temp1 = path [currentPathObj];
}
if (CurrentPathDecidingCounter == 2) {
temp1 = path2 [currentPathObj];
}
if (CurrentPathDecidingCounter == 3) {
temp1 = path3 [currentPathObj];
}
Vector3 steerVector = transform.InverseTransformPoint (new Vector3(
    temp1.position.x, transform.position.y, temp1.position.z));
float newSteer = maxSteer * (steerVector.x / steerVector.magnitude);
```

```
if (reversing == false) {
WheelFL.steerAngle = newSteer;
WheelFR.steerAngle = newSteer;
}
if (reversing == true) {
WheelFL.steerAngle = 0f;
WheelFR.steerAngle = 0f;
}
if (steerVector.magnitude <= distFromPath) {
currentPathObj++;
if (currentPathObj >= path.Count) {
currentPathObj = 0;
}
CurrentPathDecidingCounter = Random.Range (1, 4);
}
}
// GetSteer Function Ends
void Move()
{
currentSpeed = 2 * (22 / 7) * WheelRL.radius * WheelRL.rpm * 60 /
    1000;
currentSpeed = Mathf.Round (currentSpeed);
if (currentSpeed <= topSpeed && !inSector) {
if (reversing == false) {
WheelRL.motorTorque = maxTorque;
WheelRR.motorTorque = maxTorque;
}
if(reversing == true)
{
```

```
WheelRL.motorTorque = -maxTorque;
WheelRR.motorTorque = -maxTorque;
Debug.Log ("Reversing is true :" + WheelRL.motorTorque );
}
WheelRL.brakeTorque = 0;
WheelRR.brakeTorque = 0;
}
else if(!inSector)
{
WheelRL.motorTorque = 0;
WheelRR.motorTorque = 0;
WheelRL.brakeTorque = decellarationSpeed;
WheelRR.brakeTorque = decellarationSpeed;
}
}
void BreakingEffect()
{
if (isBreaking) {
breakingMesh.material = activeBreakLight;
} else {
breakingMesh.material = idleBreakLight;
}
}
void Sensors()
{
if (reversing == false) {
sensorLength = sensorLengthOriginal;
sidewaySensorsLength = sidewaySensorLengthOriginal;
}
```

```
if (reversing == true) {
sensorLength = 0.0f;
sidewaySensorsLength = 0.0f;
}
flag = 0;
float avoidSensitivity = 0;
Vector3 pos;
RaycastHit hit;
Vector3 rightAngle = Quaternion.AngleAxis (frontSensorsAngle,
    transform.up) * transform.forward;
Vector3 leftAngle = Quaternion.AngleAxis (-frontSensorsAngle,
    transform.up) * transform.forward;
pos = new Vector3 (transform.position.x, transform.position.y +
    RayOriginateHeight, transform.position.z);
pos += transform.forward * frontSensorStartPoint;
//Breaking Sensor
if (DebugSensorsFlag == true) {
Debug.DrawLine (pos, pos + new Vector3 (0, 0, sensorLength * 1.5f),
    Color.blue);
}
if (currentSpeed >= 15) {
if (Physics.Raycast (pos, transform.forward, out hit, sensorLength *
    1.5f)) {
if (hit.transform.tag != "Terrain" && hit.transform.tag !="Coin" &&
    hit.transform.tag != "Shield" && hit.transform.tag !="Nitro" ) {
flag++;
if (reversing == false) {
WheelRL.brakeTorque = decellarationSpeed;
WheelRR.brakeTorque = decellarationSpeed;
```

```
}
Debug.DrawLine (pos, hit.point, Color.green);
Debug.Log ("" + "Red");
}
} else {
WheelRL.brakeTorque = 0;
WheelRR.brakeTorque = 0;
}
}
//Front Right Sensor
pos += transform.right * frontSensorSideDistance;
if (DebugSensorsFlag == true) {
Debug.DrawLine (pos, pos + new Vector3 (0, 0, sensorLength),
    Color.blue);
}
if(Physics.Raycast(pos, transform.forward, out hit, sensorLength))
{
if (hit.transform.tag != "Terrain" && hit.transform.tag !="Coin" &&
    hit.transform.tag != "Shield" && hit.transform.tag !="Nitro")
{
flag++;
avoidSensitivity -= 1f;
Debug.Log ("Avoiding + Right Sensor");
Debug.DrawLine (pos, hit.point, Color.red);
}
}
else if(Physics.Raycast(pos, rightAngle, out hit, sensorLength))
{
```

```
if (hit.transform.tag != "Terrain" && hit.transform.tag !="Coin" &&
    hit.transform.tag != "Shield" && hit.transform.tag !="Nitro") {
avoidSensitivity -= 0.5f;
flag++;
Debug.DrawLine (pos, hit.point, Color.red);
Debug.Log ("Avoiding + Angle Sensor");
}
}
if (DebugSensorsFlag == true) {
Debug.DrawLine (pos, pos + (rightAngle * sensorLength), Color.blue);
}
// Front Angled Right Sensor
/*
Debug.DrawLine (pos, pos + (rightAngle * sensorLength), Color.blue);
if(Physics.Raycast(pos, rightAngle, out hit, sensorLength))
{
Debug.DrawLine (pos, hit.point, Color.red);
Debug.Log ("" + "Red");
}
*/
//Front Left Sensor
pos = new Vector3 (transform.position.x, transform.position.y +
    RayOriginateHeight, transform.position.z);
pos += transform.forward * frontSensorStartPoint;
pos -= transform.right * frontSensorSideDistance;
if (DebugSensorsFlag == true) {
Debug.DrawLine (pos, pos + new Vector3 (0, 0, sensorLength),
    Color.blue);
}
```

```
if(Physics.Raycast(pos, transform.forward, out hit, sensorLength))
{
if (hit.transform.tag != "Terrain" && hit.transform.tag !="Coin" &&
    hit.transform.tag != "Shield" && hit.transform.tag !="Nitro")
{
flag++;
avoidSensitivity += 1f;
//Debug.Log ("Avoiding");
Debug.DrawLine (pos, hit.point, Color.red);
Debug.Log ("Avoiding" + "Left Angle Sensor -> " + hit.transform.tag);
}
}
else if(Physics.Raycast(pos, leftAngle, out hit, sensorLength))
{
if (hit.transform.tag != "Terrain" && hit.transform.tag !="Coin" &&
    hit.transform.tag != "Shield" && hit.transform.tag !="Nitro") {
flag++;
avoidSensitivity += 0.5f;
Debug.DrawLine (pos, hit.point, Color.red);
Debug.Log ("Avoiding + " + "Left Angle Sensor -> " +
    hit.transform.tag);
}
}
if (DebugSensorsFlag == true) {
Debug.DrawLine (pos, pos + (leftAngle * sensorLength), Color.blue);
}
// Front Angled Left Sensor
/*
Debug.DrawLine (pos, pos + (leftAngle * sensorLength), Color.blue);
```

```
if(Physics.Raycast(pos, leftAngle, out hit, sensorLength))
{
Debug.DrawLine (pos, hit.point, Color.red);
Debug.Log ("" + "Red");
}
*/
// Sideway Right Sensor
pos = new Vector3 (transform.position.x, transform.position.y +
    RayOriginateHeight, transform.position.z);
if (DebugSensorsFlag == true) {
Debug.DrawLine (pos, transform.position + new Vector3
    (sidewaySensorsLength, 0, 0), Color.blue);
}
if(Physics.Raycast(transform.position, transform.right, out hit,
    sidewaySensorsLength))
{
if (hit.transform.tag != "Terrain" && hit.transform.tag !="Coin" &&
    hit.transform.tag != "Shield" && hit.transform.tag !="Nitro") {
flag++;
avoidSensitivity -= 0.5f;
Debug.DrawLine (transform.position, hit.point, Color.red);
Debug.Log ("Avoiding + Right SideWay Sensor");
}
}
// Sideway Left Sensor
pos = new Vector3 (transform.position.x, transform.position.y +
    RayOriginateHeight, transform.position.z);
if (DebugSensorsFlag == true) {
```

```
Debug.DrawLine (pos, transform.position + new Vector3
    (-sidewaySensorsLength, 0, 0), Color.blue);
}
if(Physics.Raycast(transform.position, -transform.right, out hit,
    sidewaySensorsLength))
{
if (hit.transform.tag != "Terrain" && hit.transform.tag !="Coin" &&
    hit.transform.tag != "Shield" && hit.transform.tag !="Nitro") {
flag++;
avoidSensitivity += 0.5f;
Debug.DrawLine (transform.position, hit.point, Color.red);
Debug.Log ("Avoiding + Left SideWay Sensor");
}
}
//Front Mid Sensor
pos = new Vector3 (transform.position.x, transform.position.y +
    RayOriginateHeight, transform.position.z);
pos += transform.forward * frontSensorStartPoint;
if (DebugSensorsFlag == true) {
Debug.DrawLine (pos, pos + new Vector3 (0, 0, sensorLength),
    Color.blue);
}
if (avoidSensitivity == 0) {
if (Physics.Raycast (pos, transform.forward, out hit, sensorLength *
    1.5f)) {
if (hit.transform.tag != "Terrain" && hit.transform.tag !="Coin" &&
    hit.transform.tag != "Shield" && hit.transform.tag !="Nitro") {
if (hit.normal.x > 0) {
avoidSensitivity = -1;
```

```
} else {
avoidSensitivity = 1;
}
Debug.DrawLine (pos, hit.point, Color.red);
Debug.Log ("" + "Red");
}
}
}
if (gameObject.GetComponent<Rigidbody> ().velocity.magnitude < 2 &&
    gameObject.GetComponent<Rigidbody> ().velocity.magnitude >= 0 &&
    !reversing) {
reversingCounter += Time.deltaTime;
if (reversingCounter >= waitToReverse) {
reversingCounter = 0;
reversing = true;
Debug.Log ("Reversing= " + reversing);
}
}
else if (!reversing)
{
reversingCounter = 0;
}
if (reversing)
{
avoidSensitivity *= -1;
//avoidSensitivity *= 1;
reversingCounter += Time.deltaTime;
if (reversingCounter >= reverseFor)
{
```

```
reversingCounter = 0;
reversing = false;
}
}
if (flag != 0) {
AvoidSteer (avoidSensitivity);
}
} // End Sensor Function
void AvoidSteer(float sensitivity)
{
WheelFL.steerAngle = avoidSpeed * sensitivity;
WheelFR.steerAngle = avoidSpeed * sensitivity;
}
void Respawn()
{
if (gameObject.GetComponent<Rigidbody> ().velocity.magnitude < 2) {
respawnCounter += Time.deltaTime;
if (respawnCounter >= respawnWait) {
if (currentPathObj == 0) {
transform.position = path [path.Count - 1].position;
} else {
transform.position = path [currentPathObj - 1].position;
}
respawnCounter = 0;
transform.localEulerAngles = new Vector3
    (transform.localEulerAngles.x, transform.localEulerAngles.y, 0f);
}
}
if (gameObject.GetComponent<Rigidbody> ().velocity.magnitude >= 2) {
```

```
respawnCounter = 0;
}
}
}
```

6 UML diagrams

This section contains dataflow, usecases and activity diagrams of this gaming project.

6.1 DataFlow Diagrams :

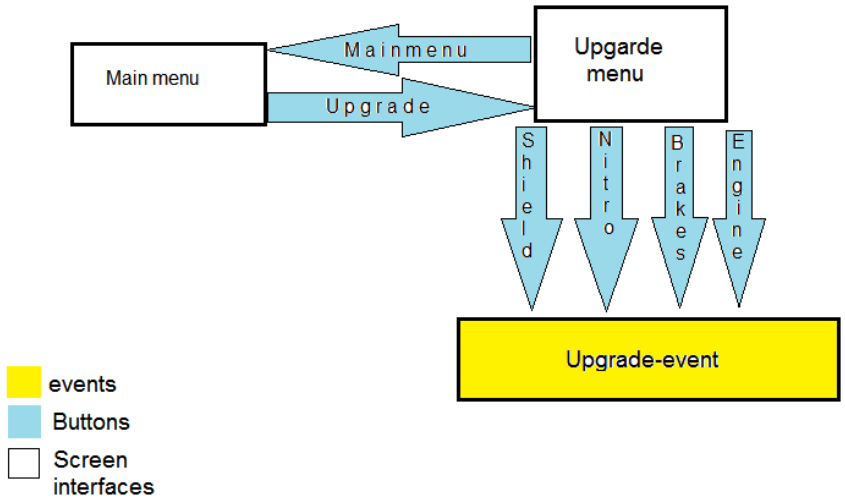

Figure 27: DataFlow Upgrades Menu

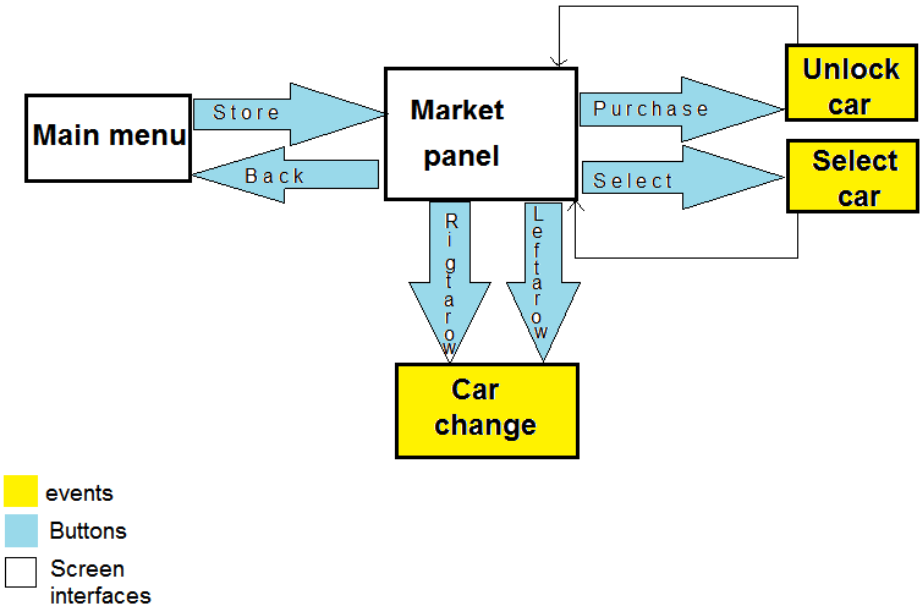

Figure 28: DataFlow Market Menu

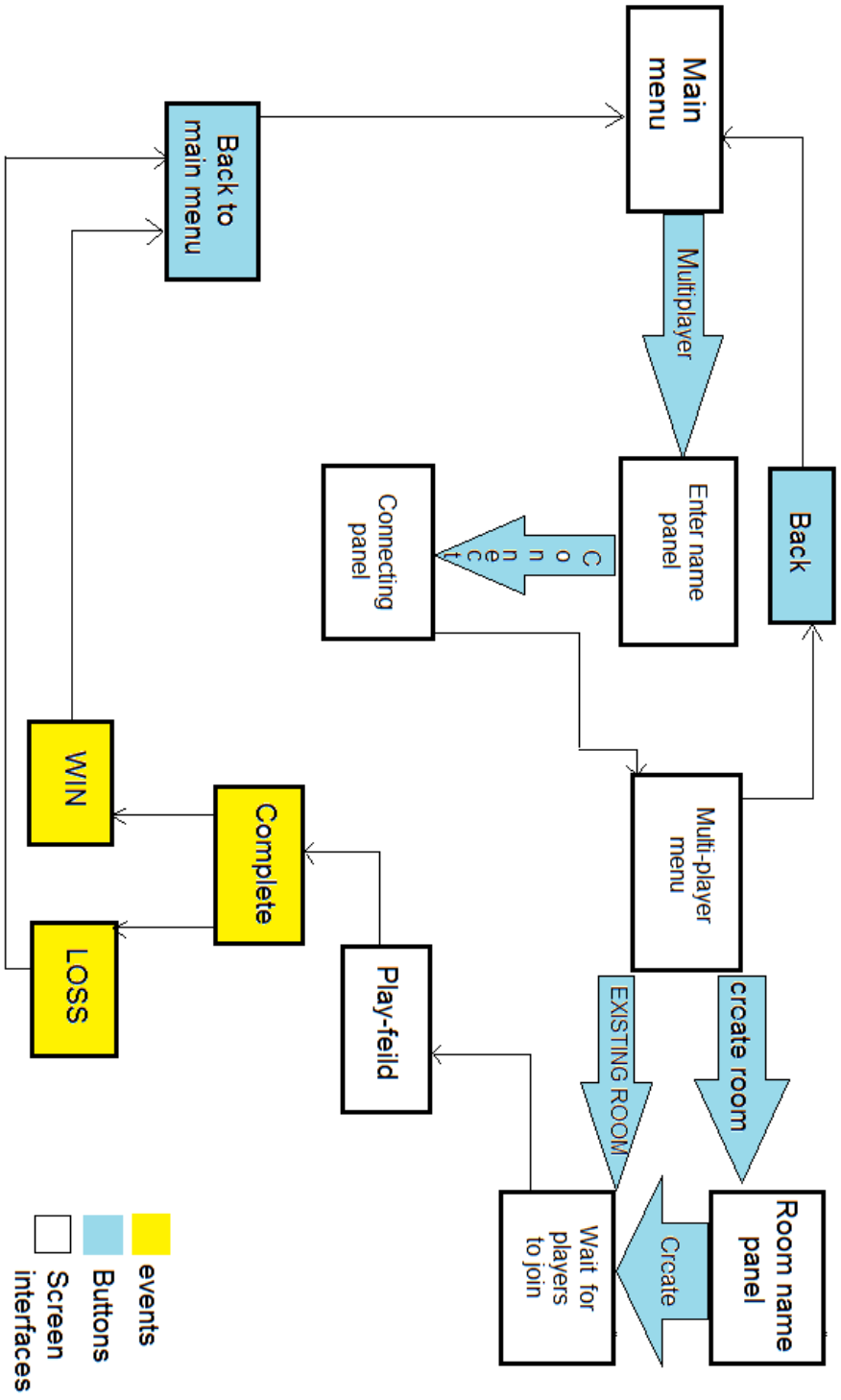

Figure 29: DataFlow Multiplayer

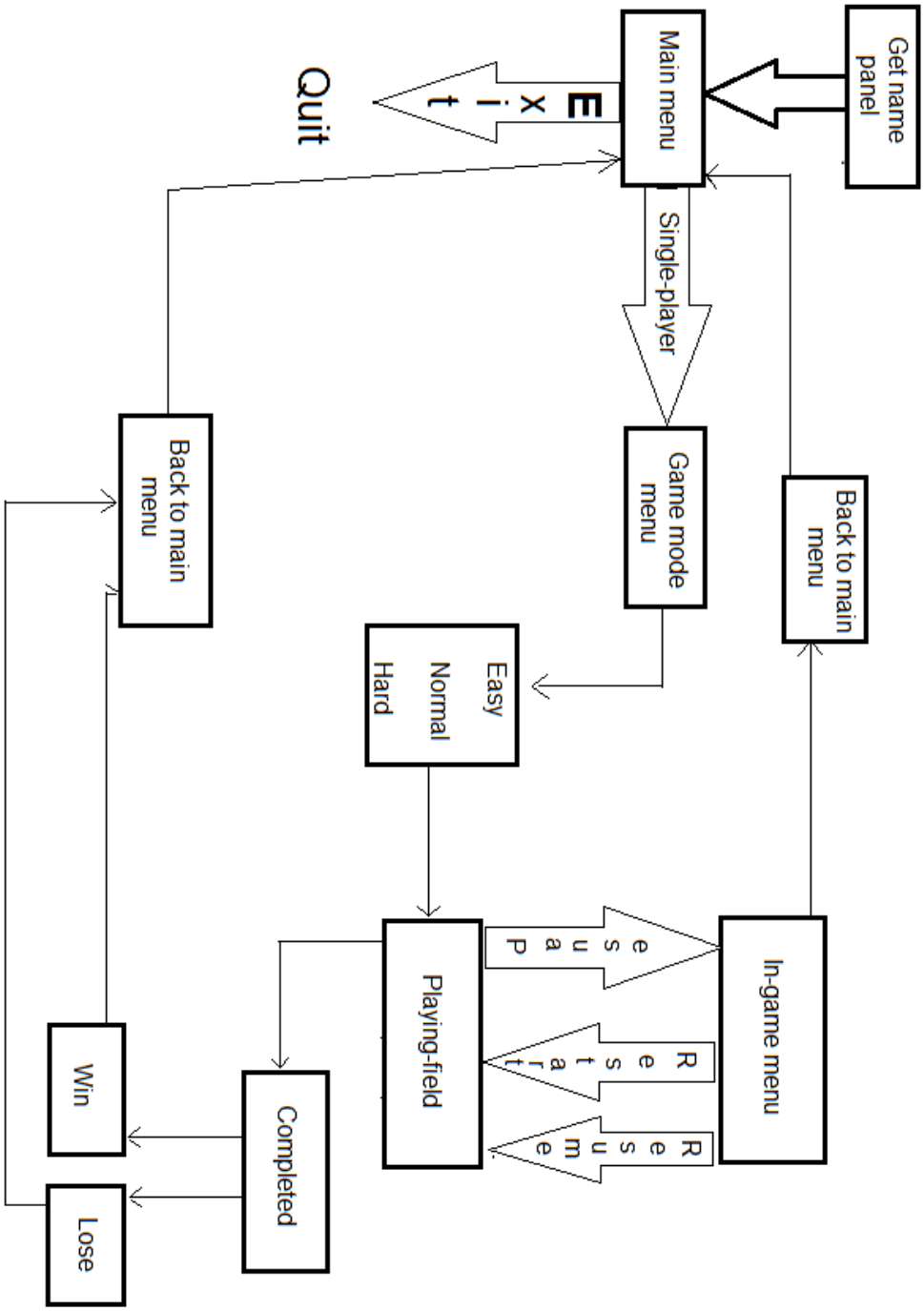

Figure 30: DataFlow Single Player

6.2 UseCases Diagrams :

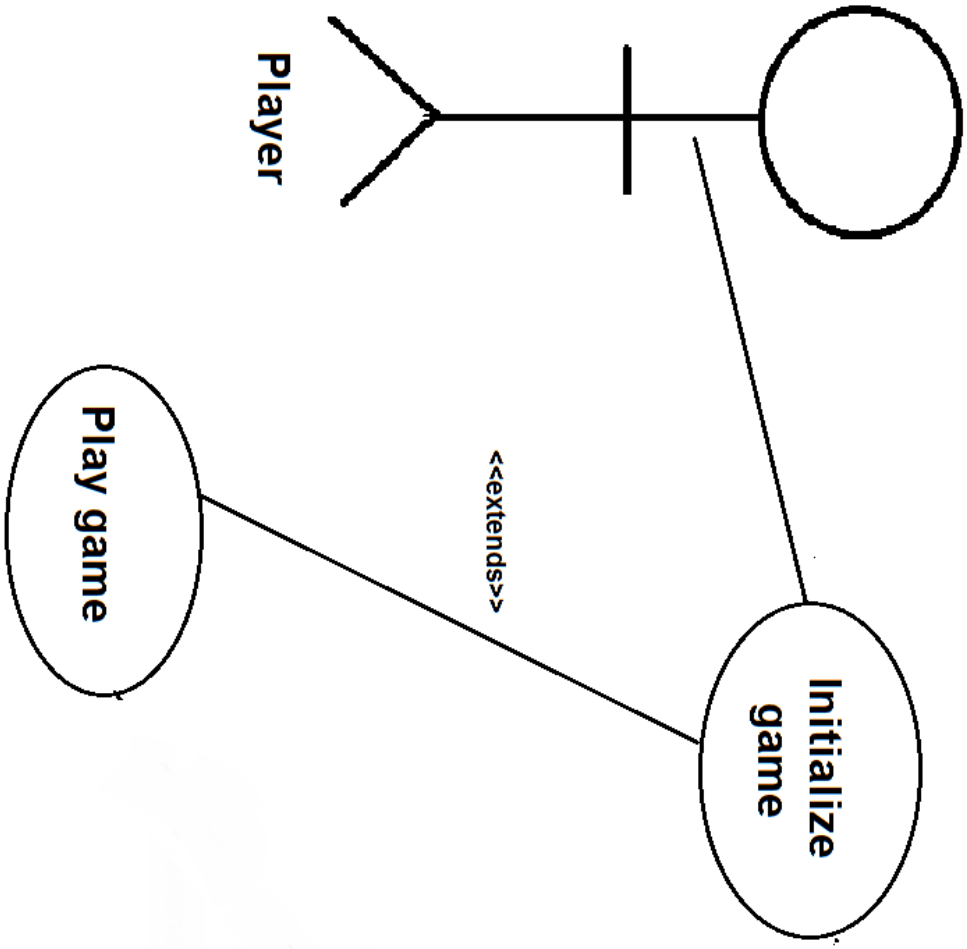

Figure 31: UseCase PlayGame

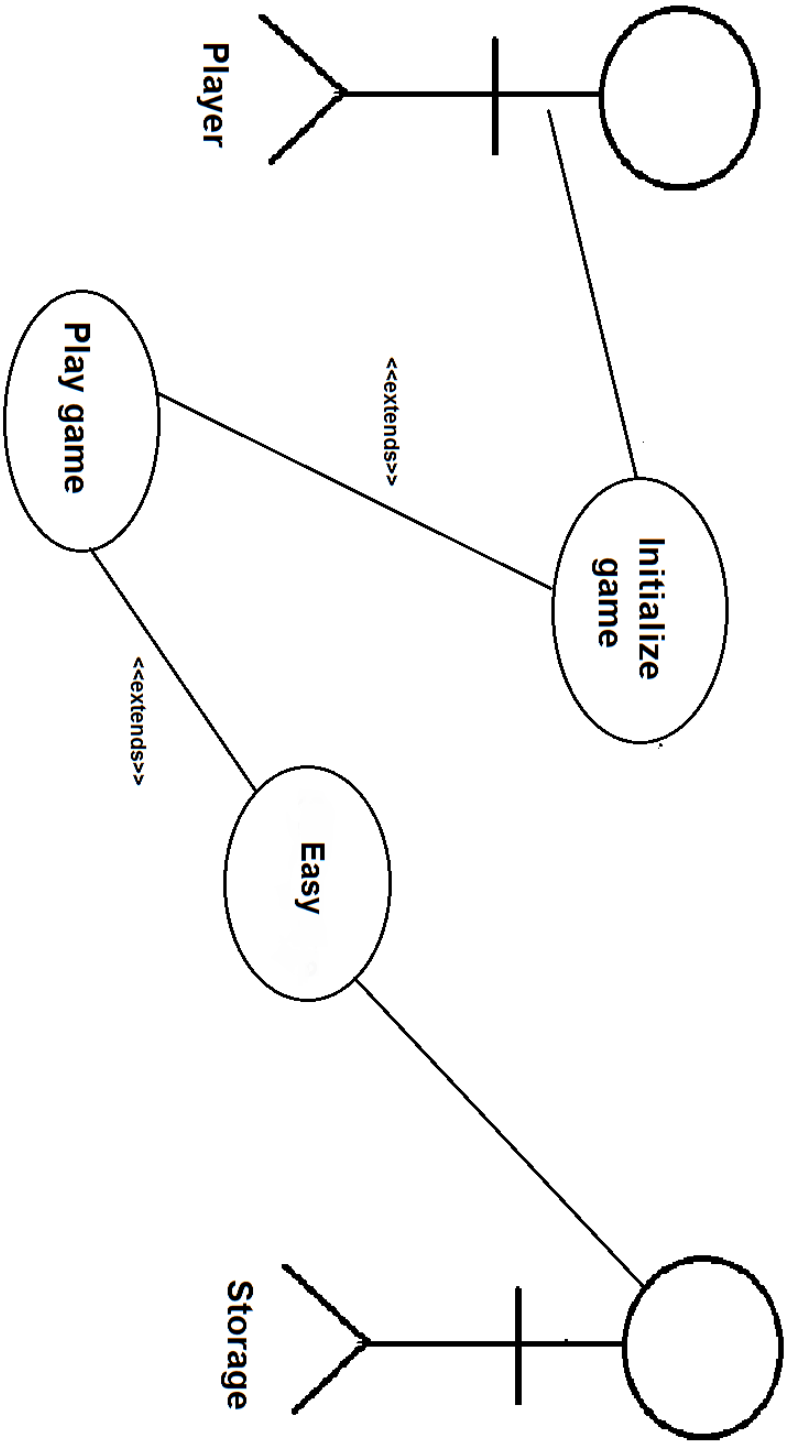

Figure 32: UseCase Easy Mode

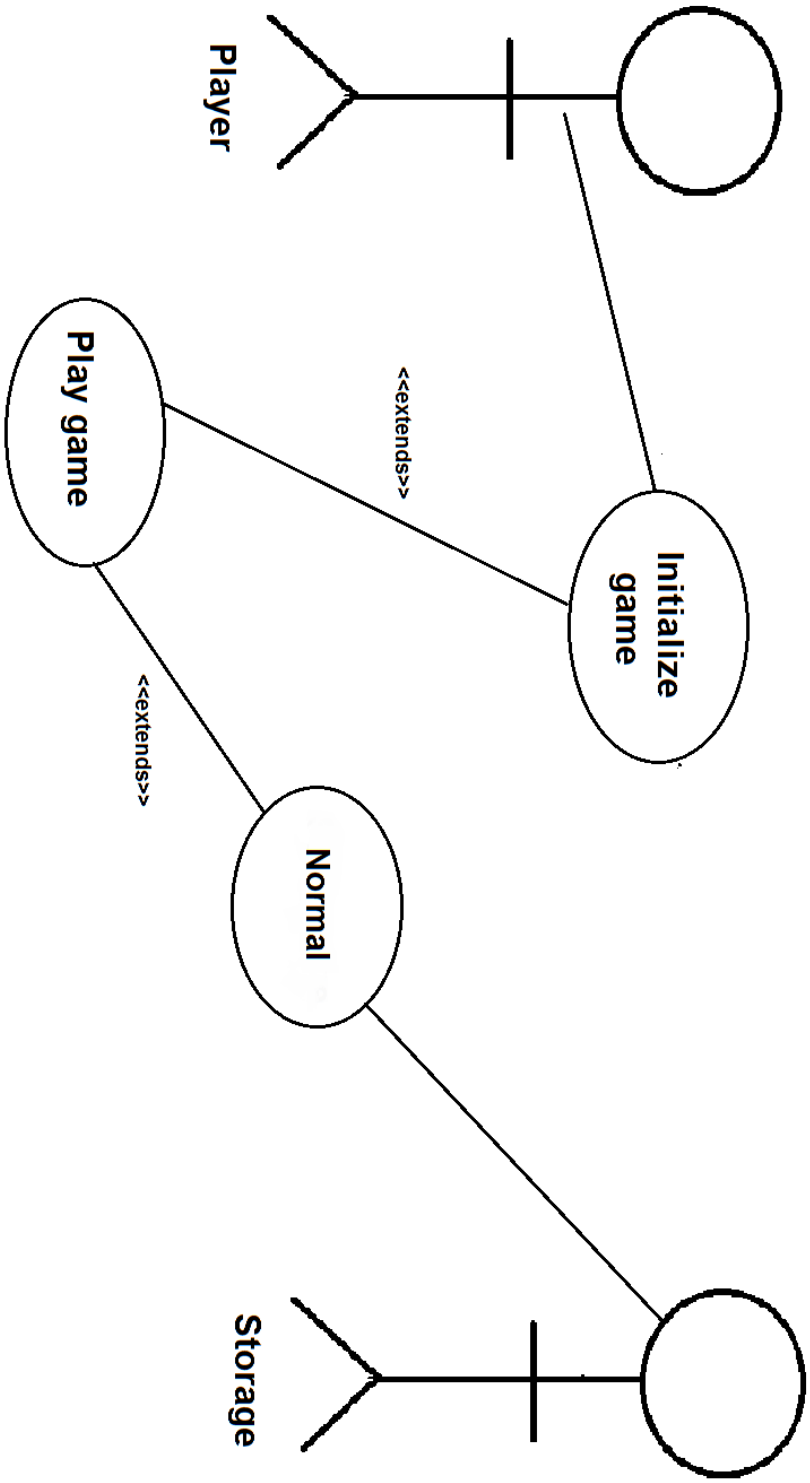

Figure 33: UseCase Normal Mode

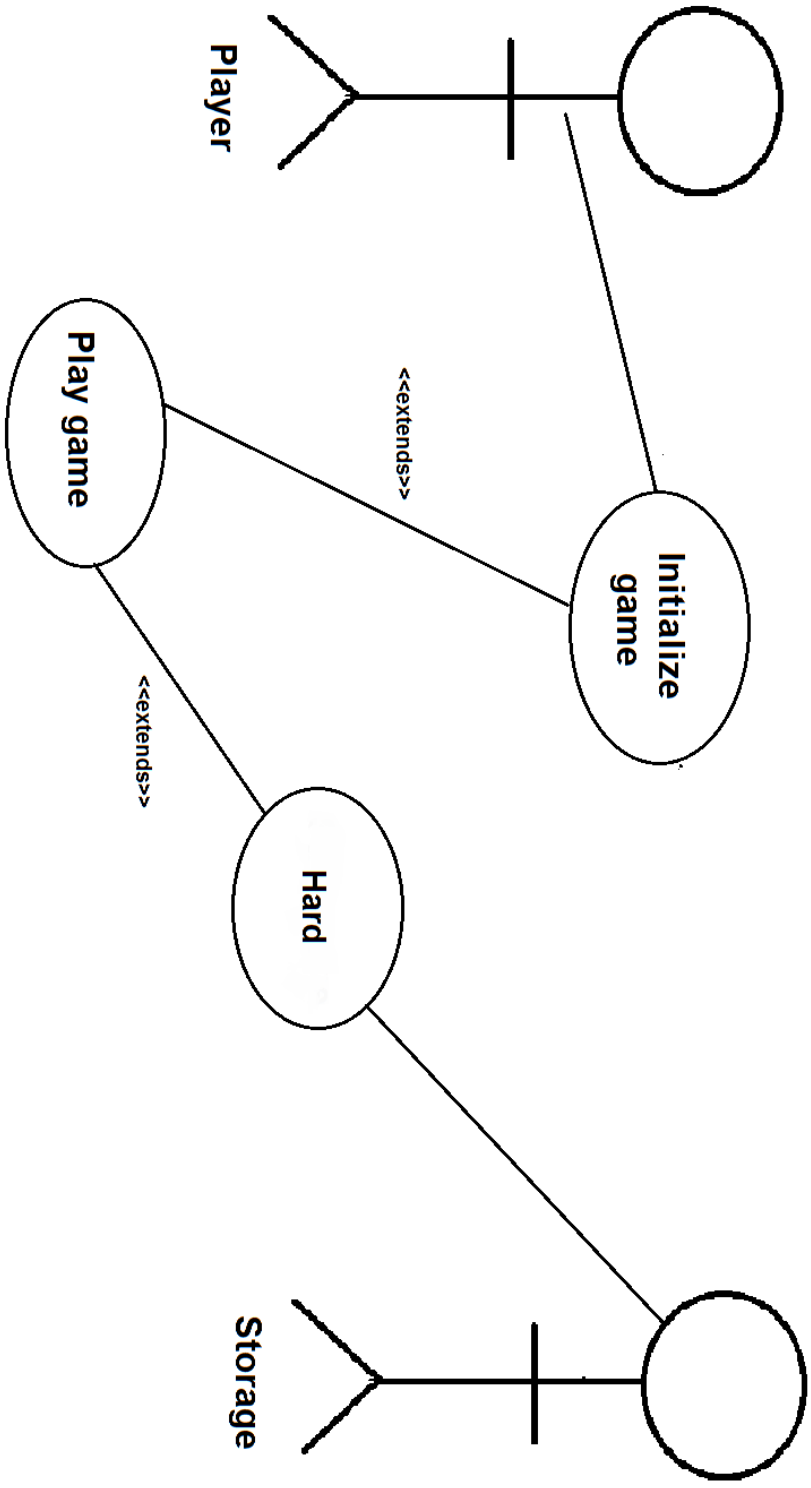

Figure 34: UseCase Hard Mode

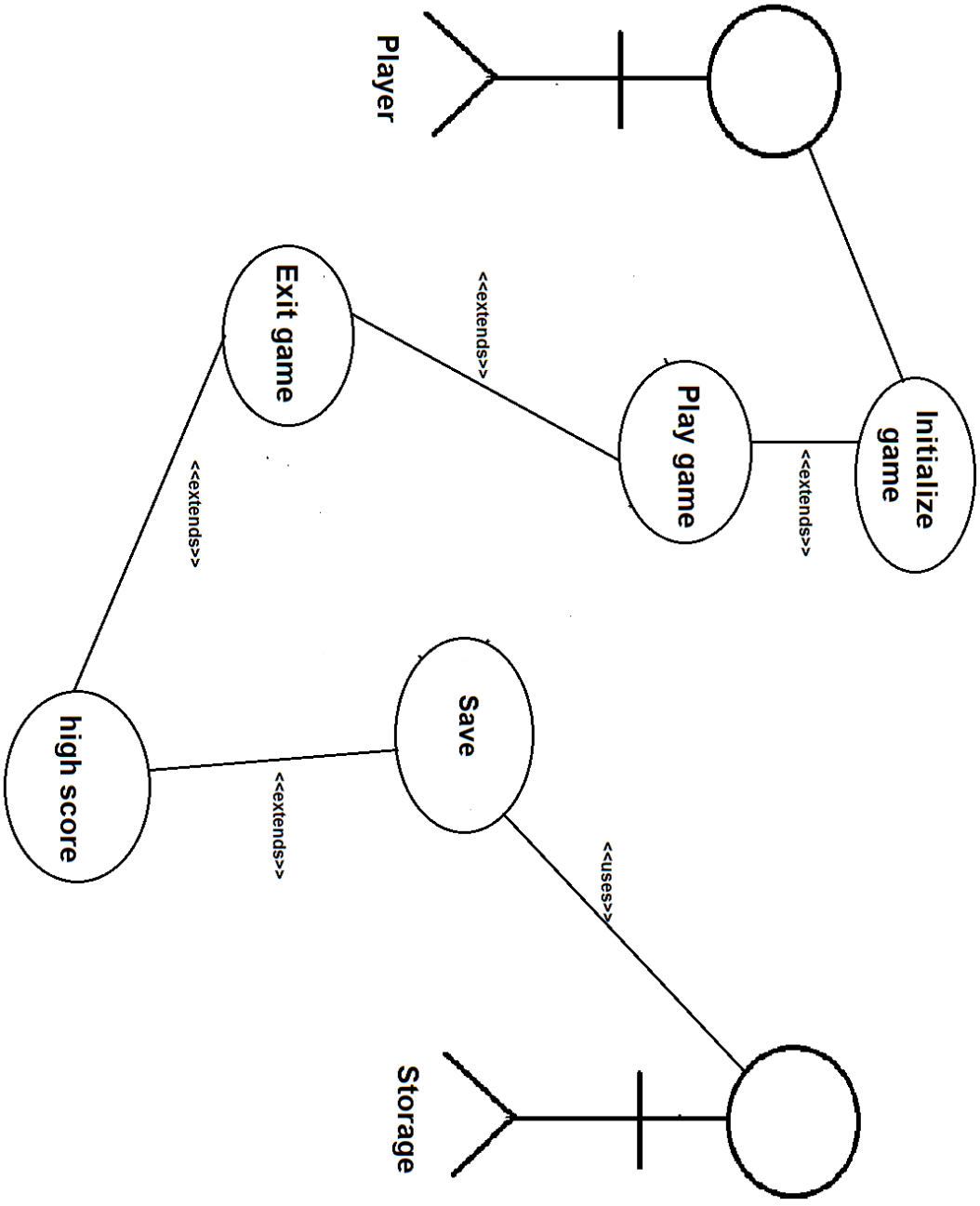

Figure 35: UseCase Exit

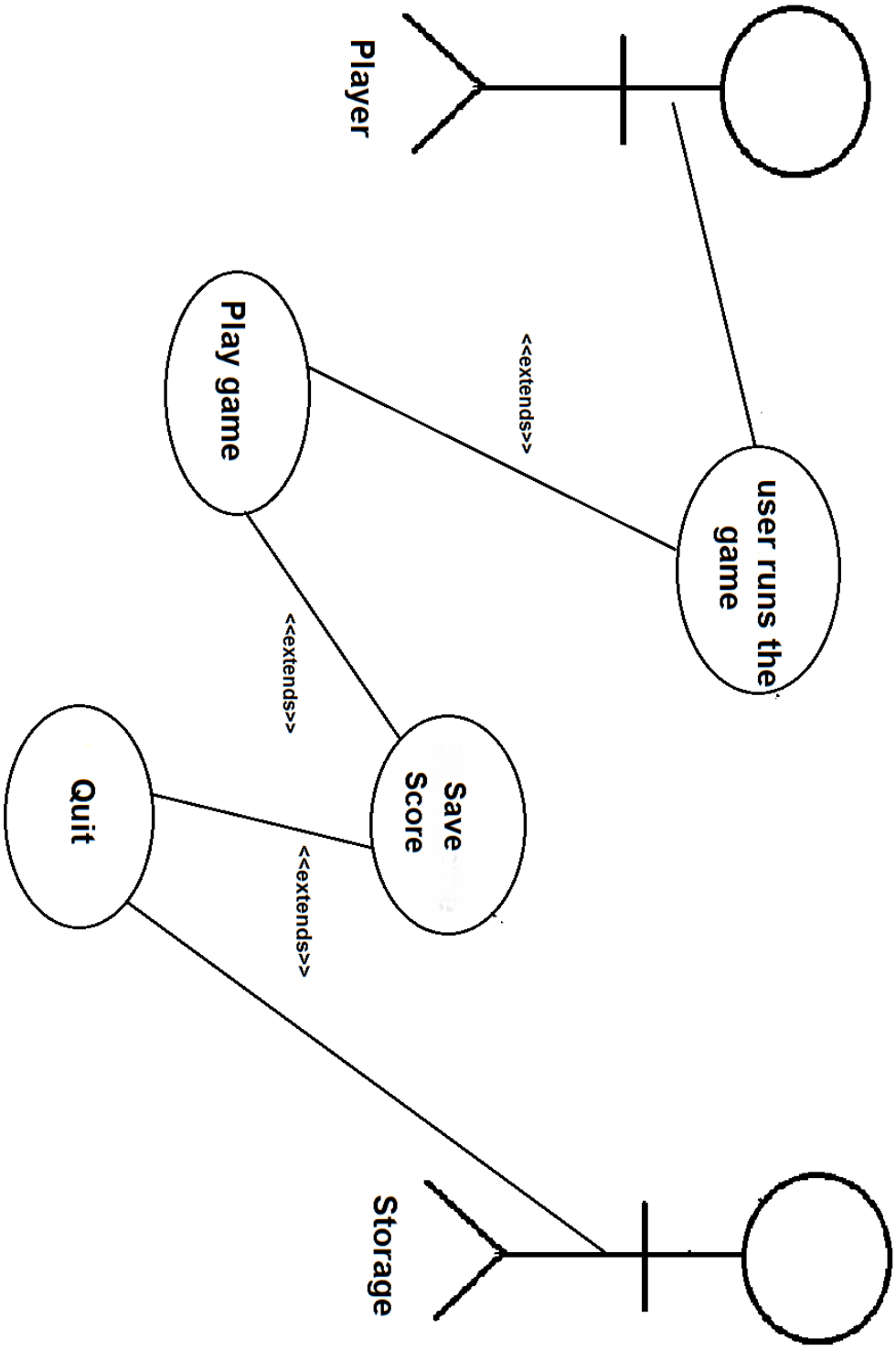

Figure 36: UseCase Game

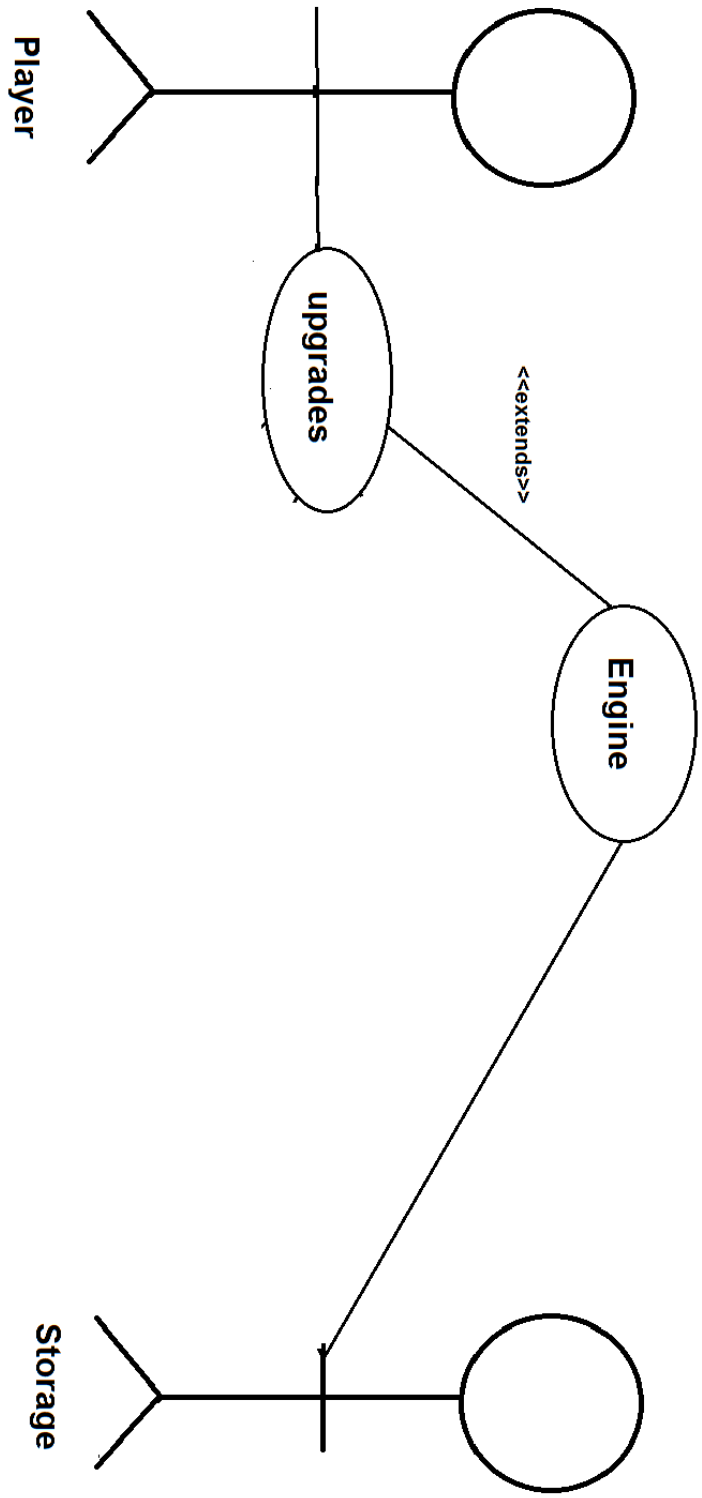

Figure 37: UseCase Upgrade Engine

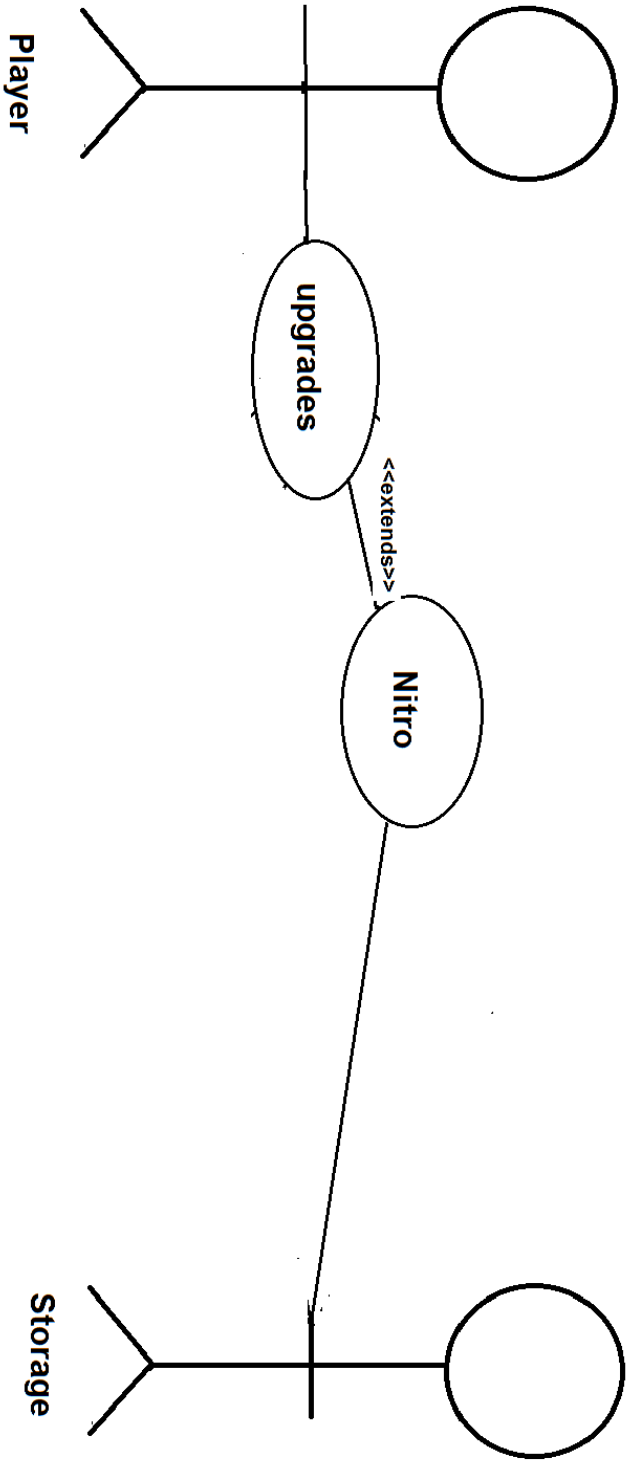

Figure 38: UseCase Upgrade Nitro

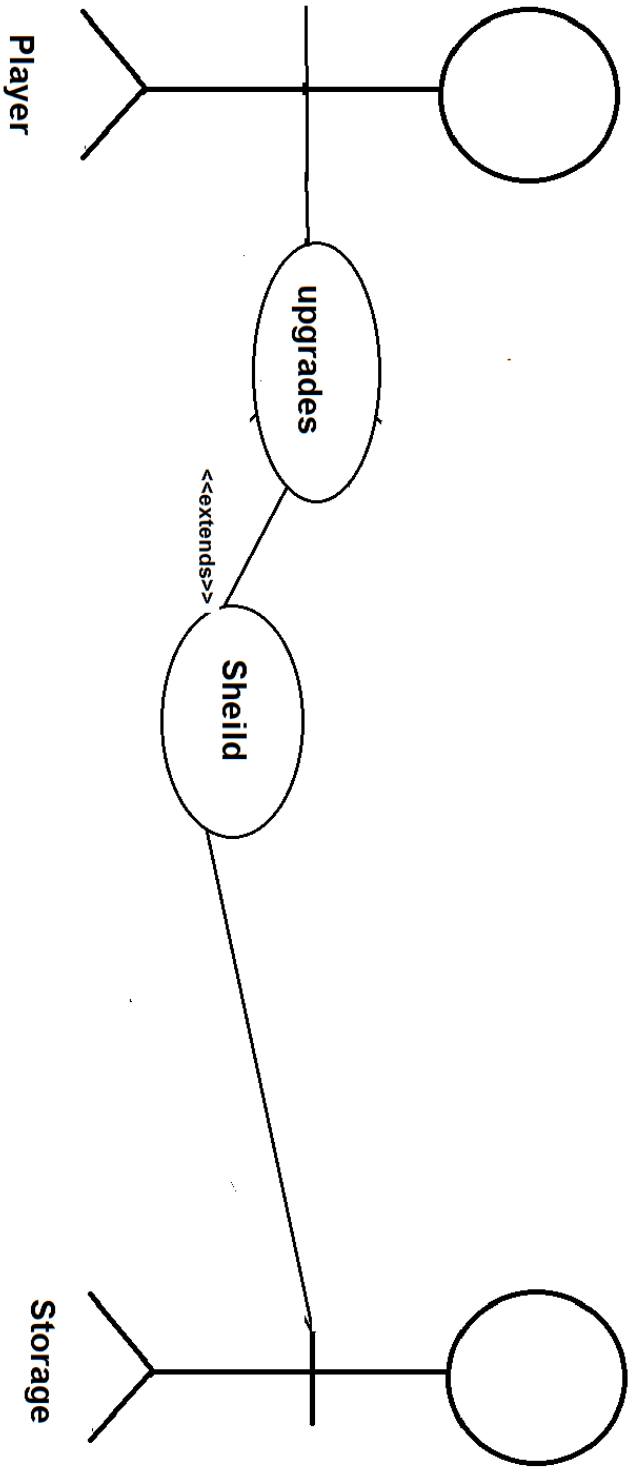

Figure 39: UseCase Upgrade Shield

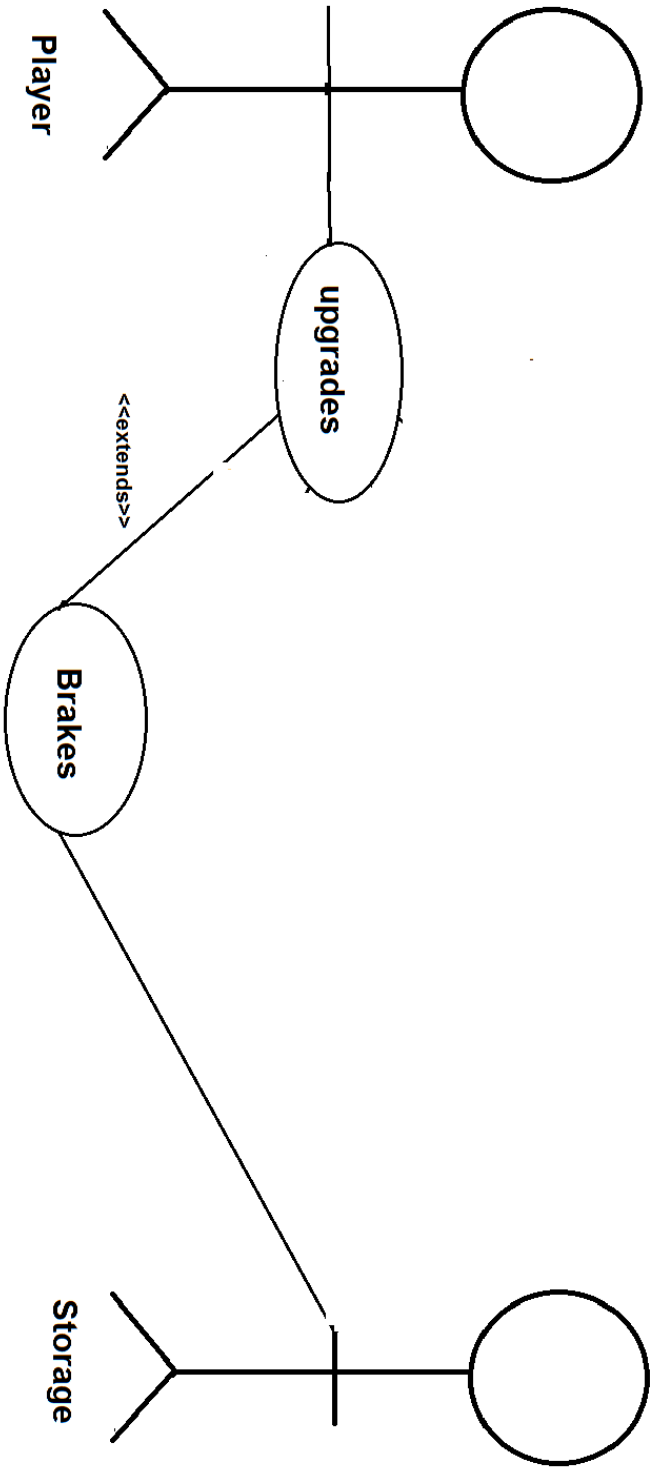

Figure 40: UseCase Upgrade Brakes

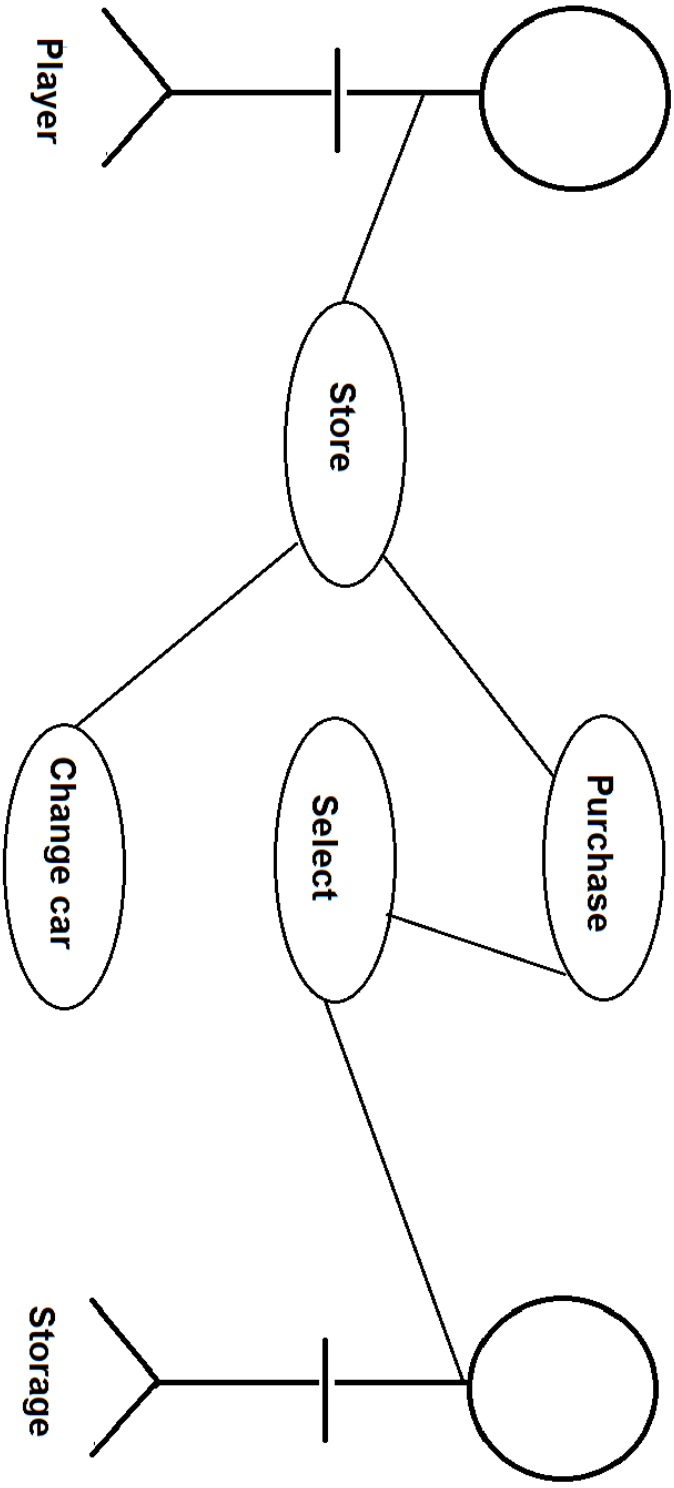

Figure 41: UseCase Change Car

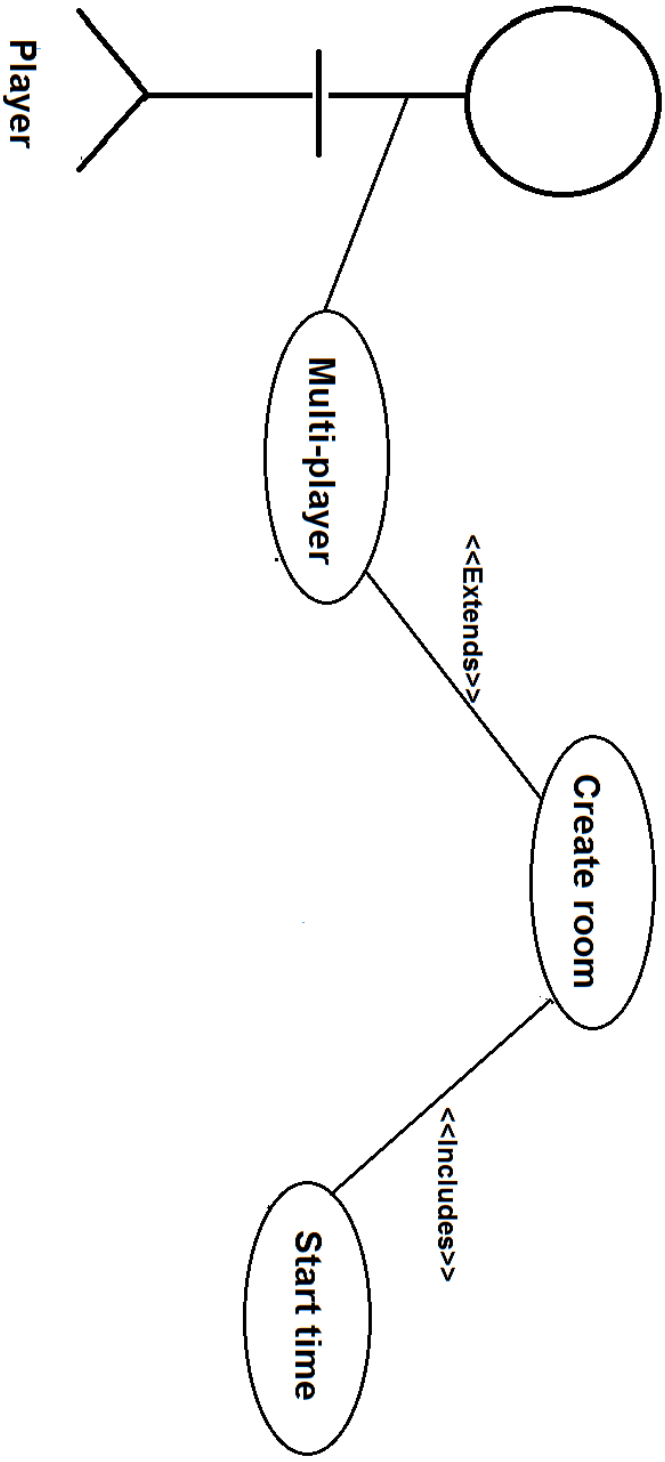

Figure 42: UseCase Create Room

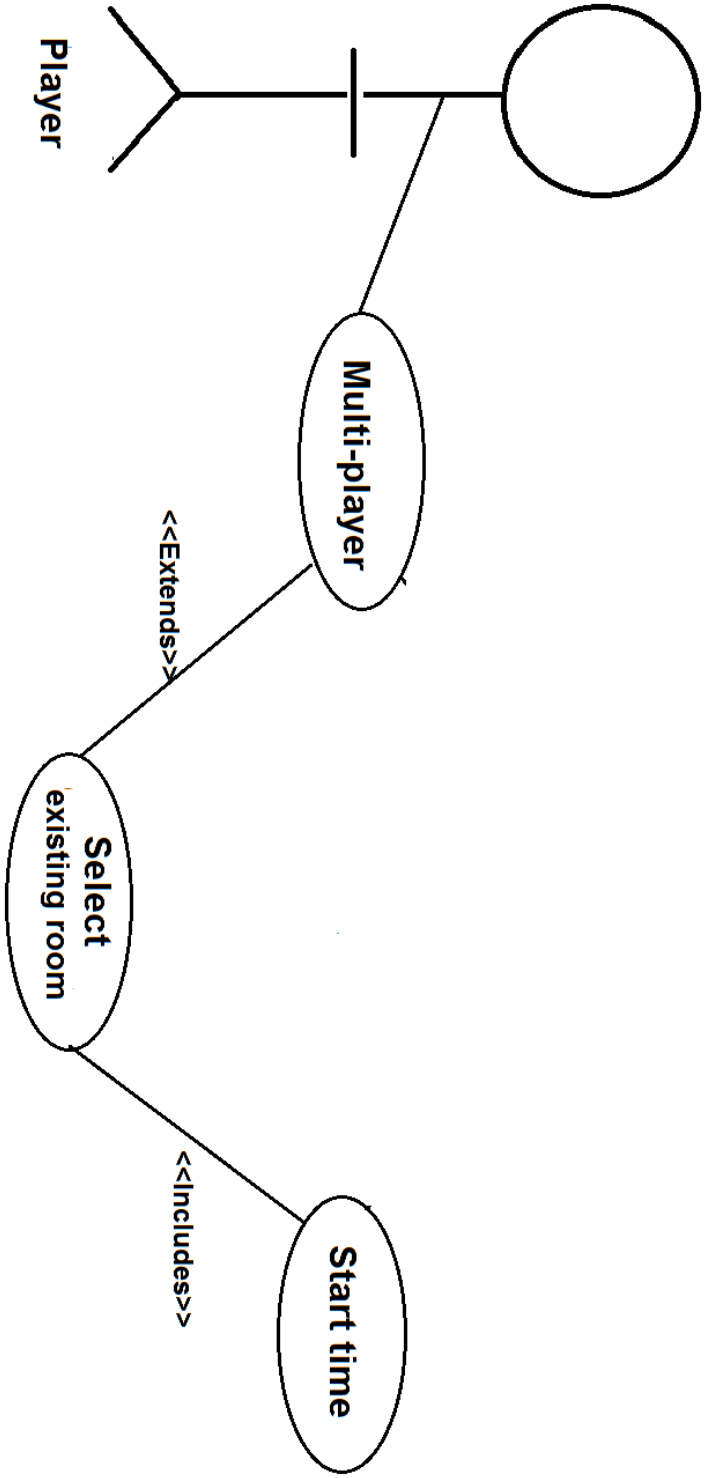

Figure 43: UseCase Join Room

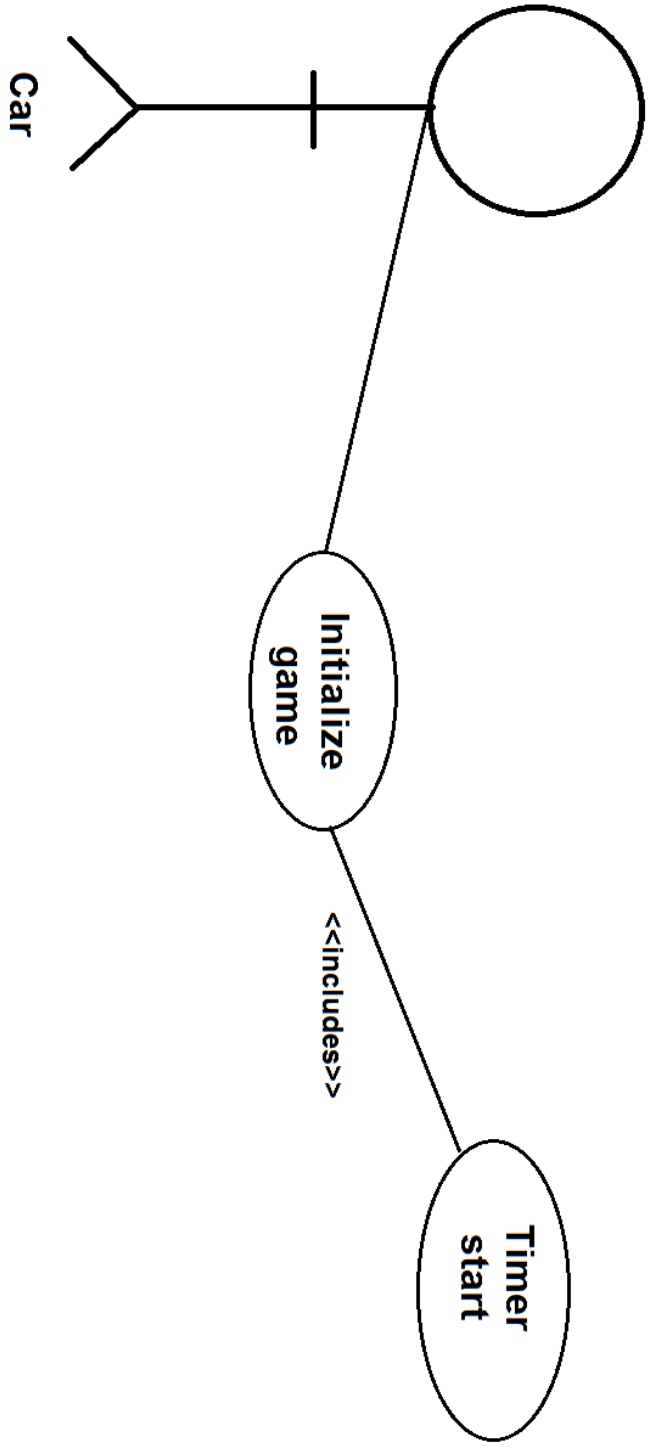

Figure 44: UseCase Race Start

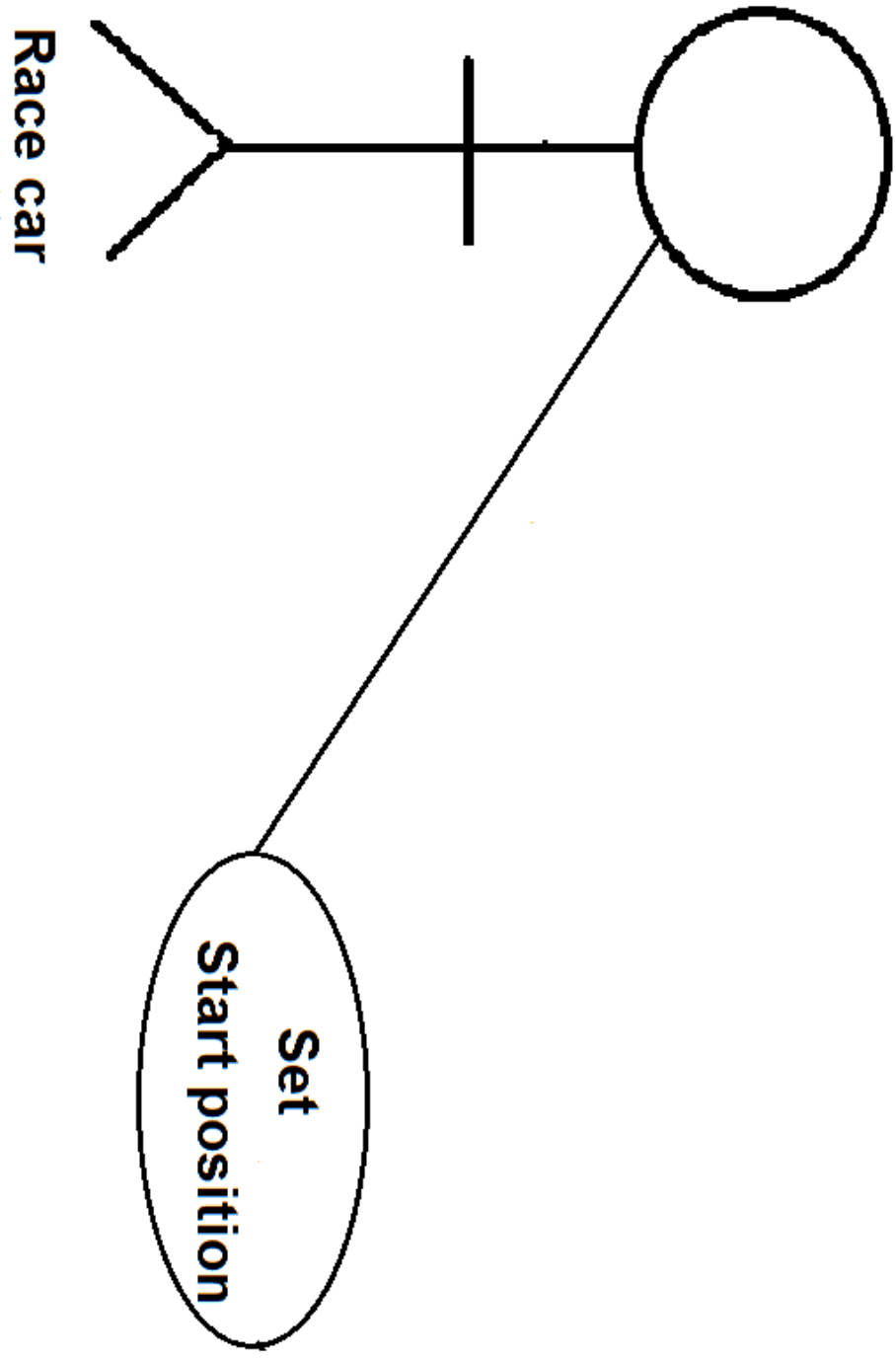

Figure 45: UseCase Start Position

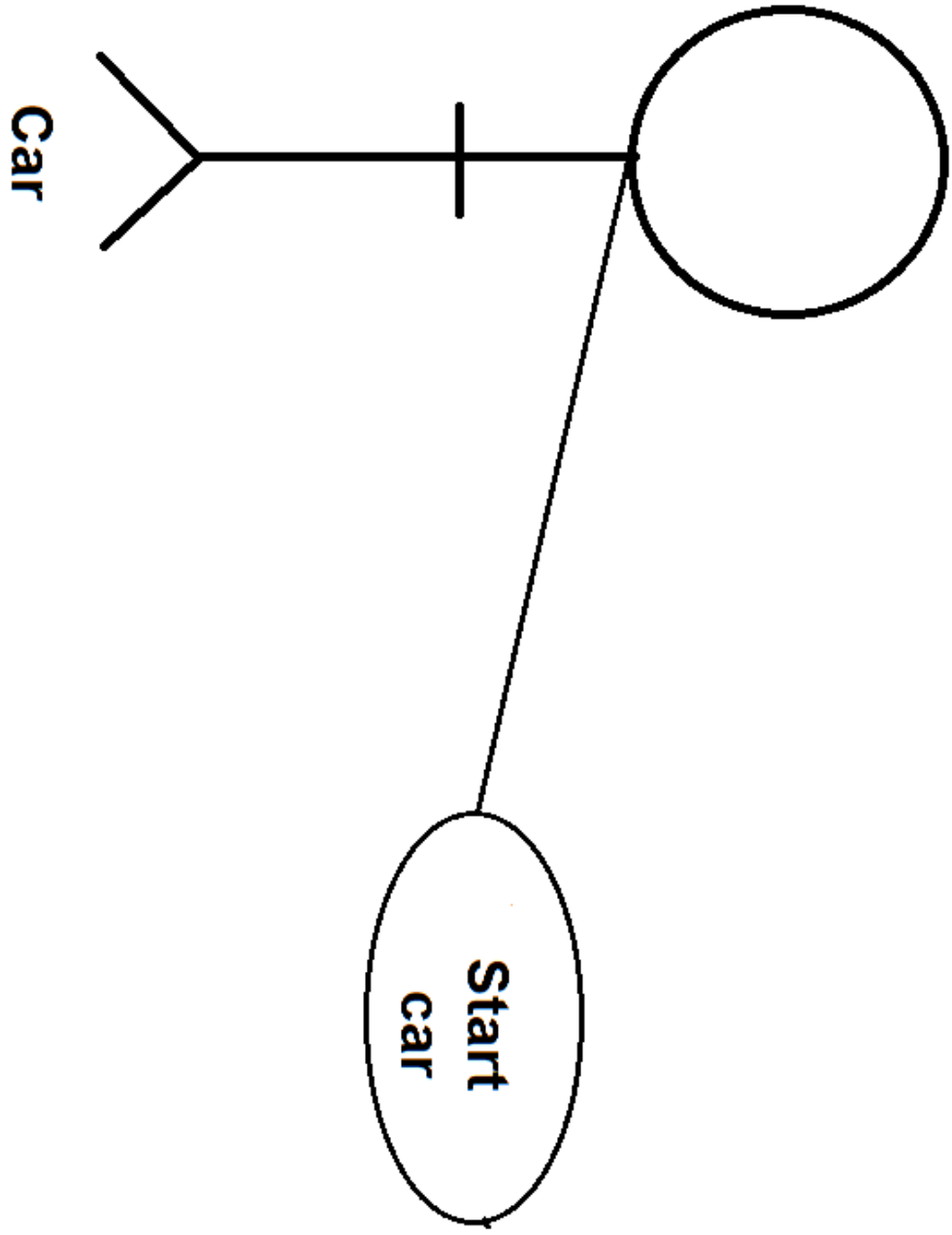

Figure 46: UseCase Start Car

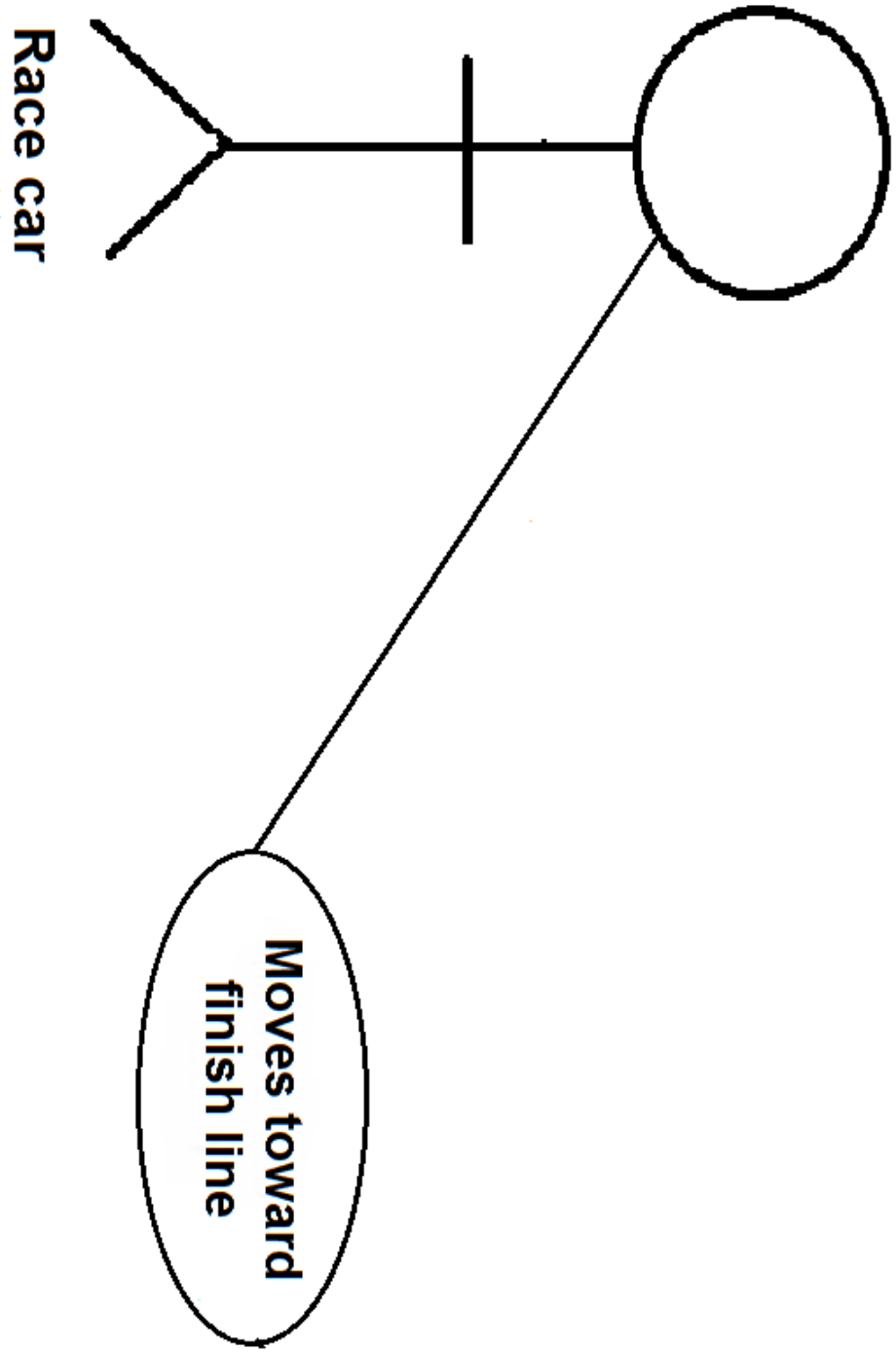

Figure 47: UseCase Move Car

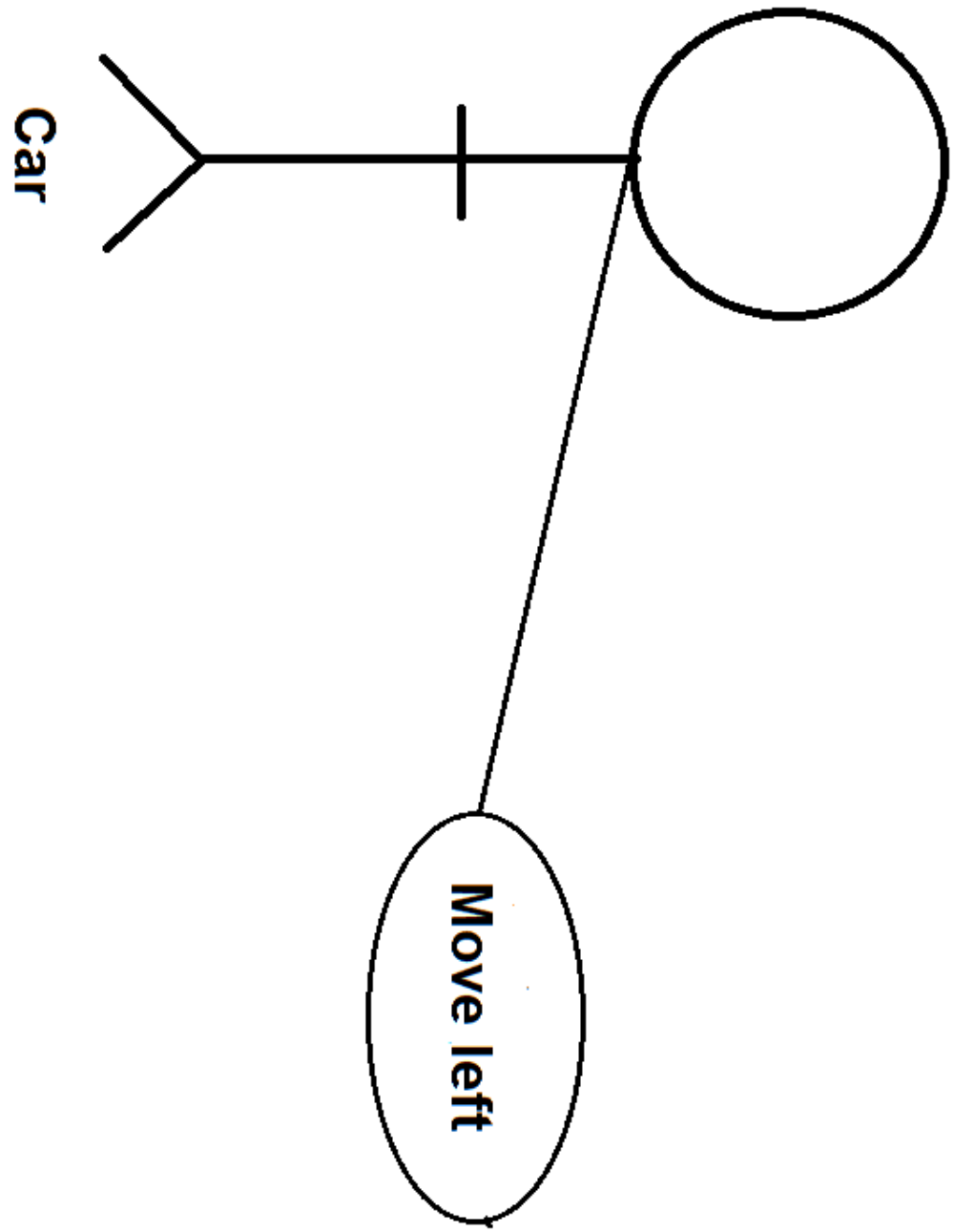

Figure 48: UseCase Left Turn

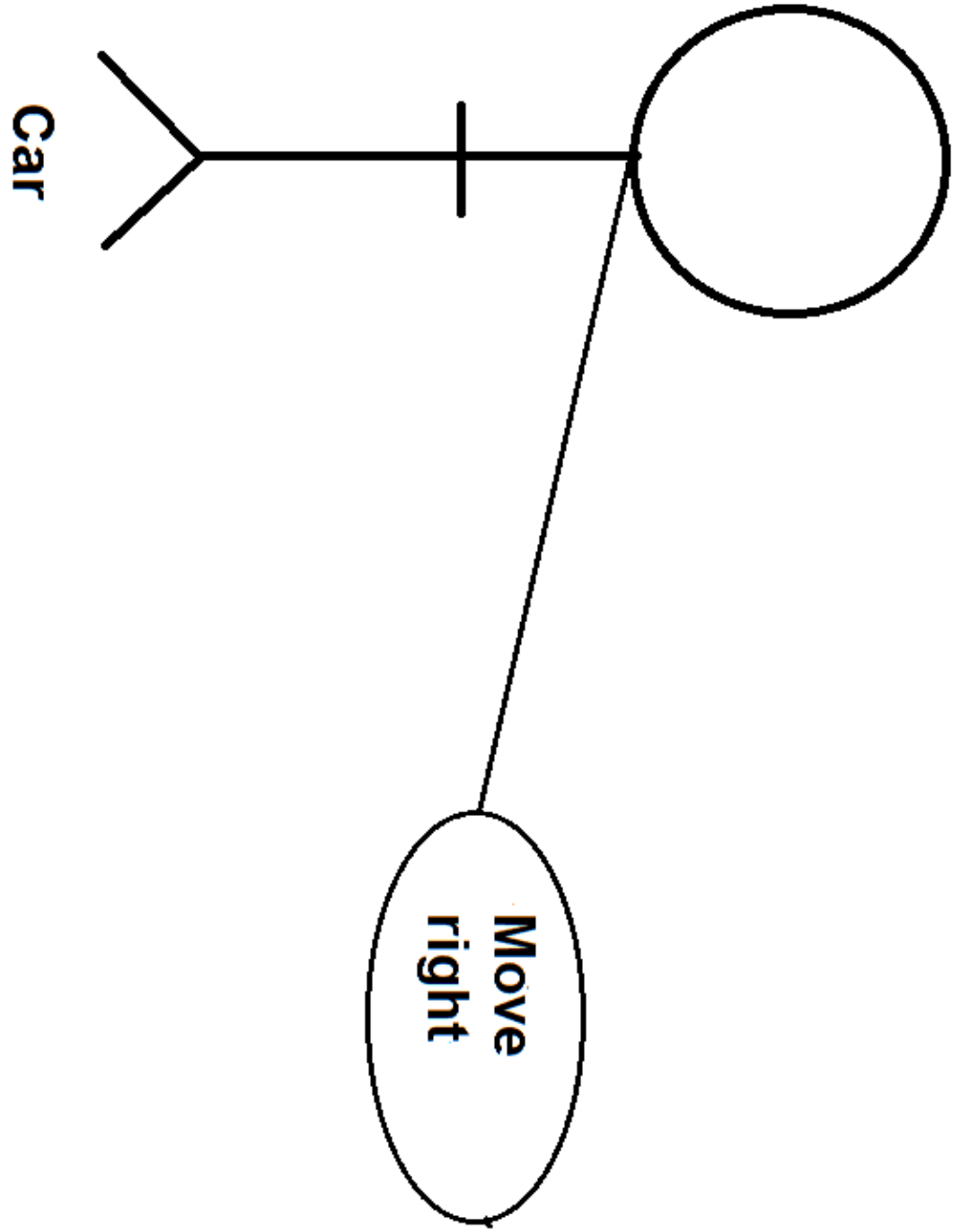

Figure 49: UseCase Right turn

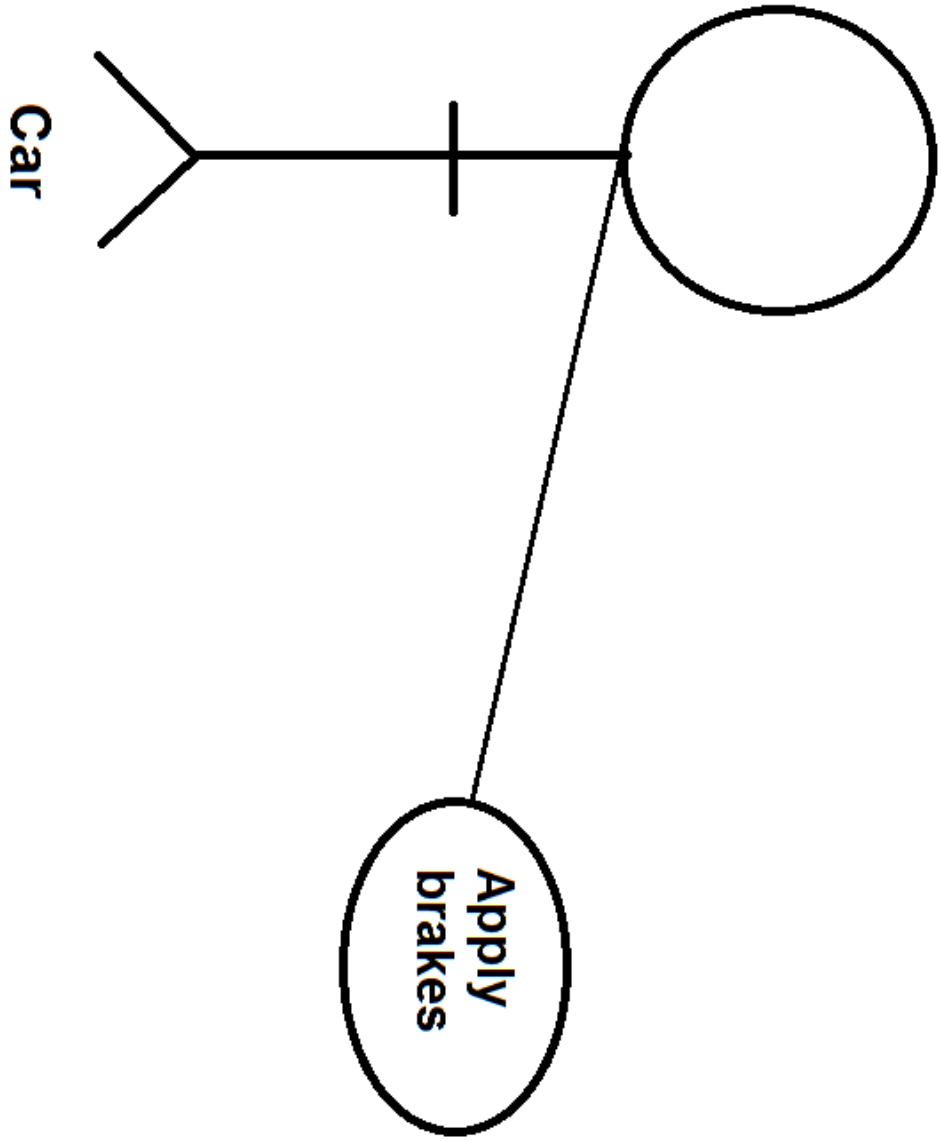

Figure 50: UseCase Apply Brakes

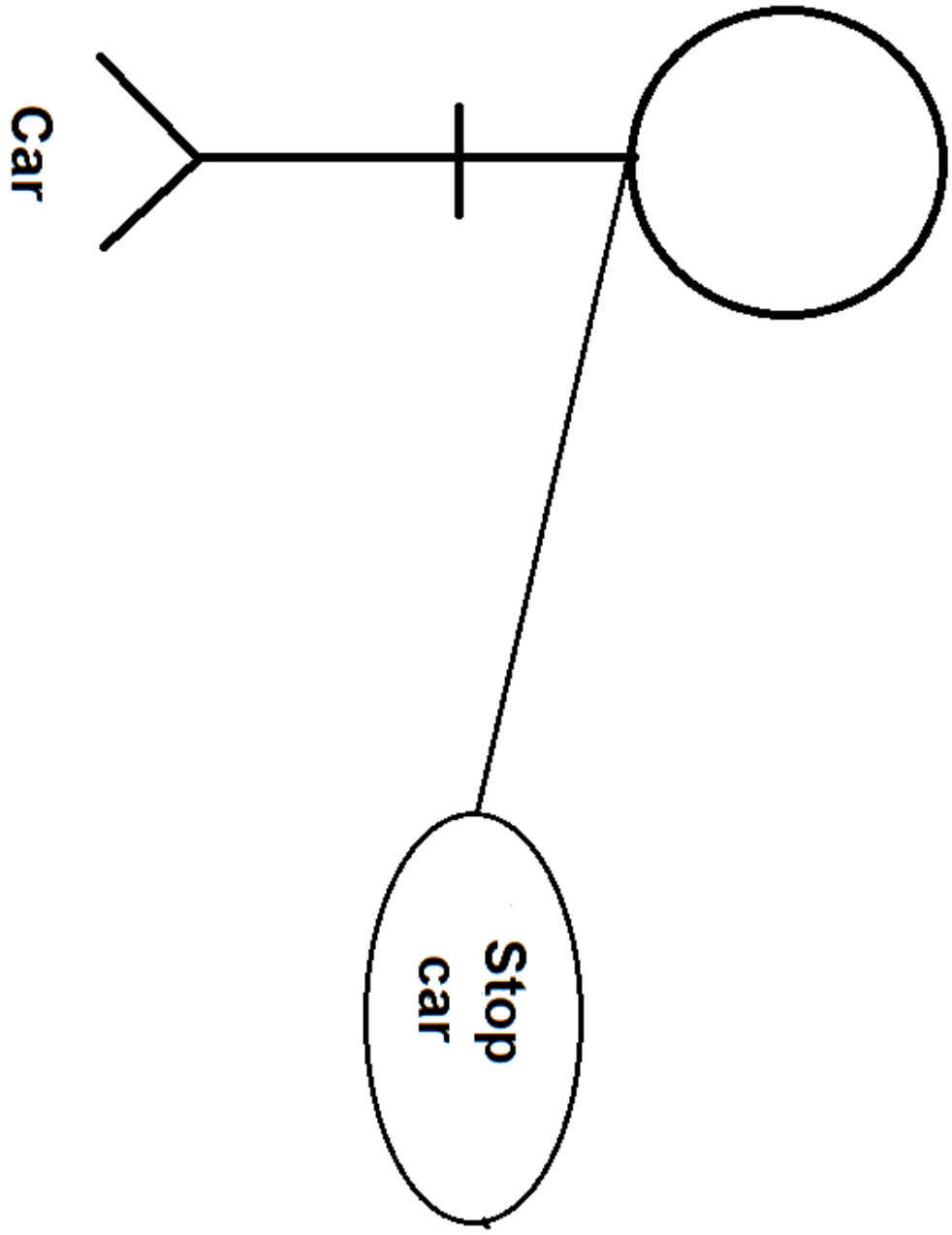

Figure 51: UseCase Stop Car

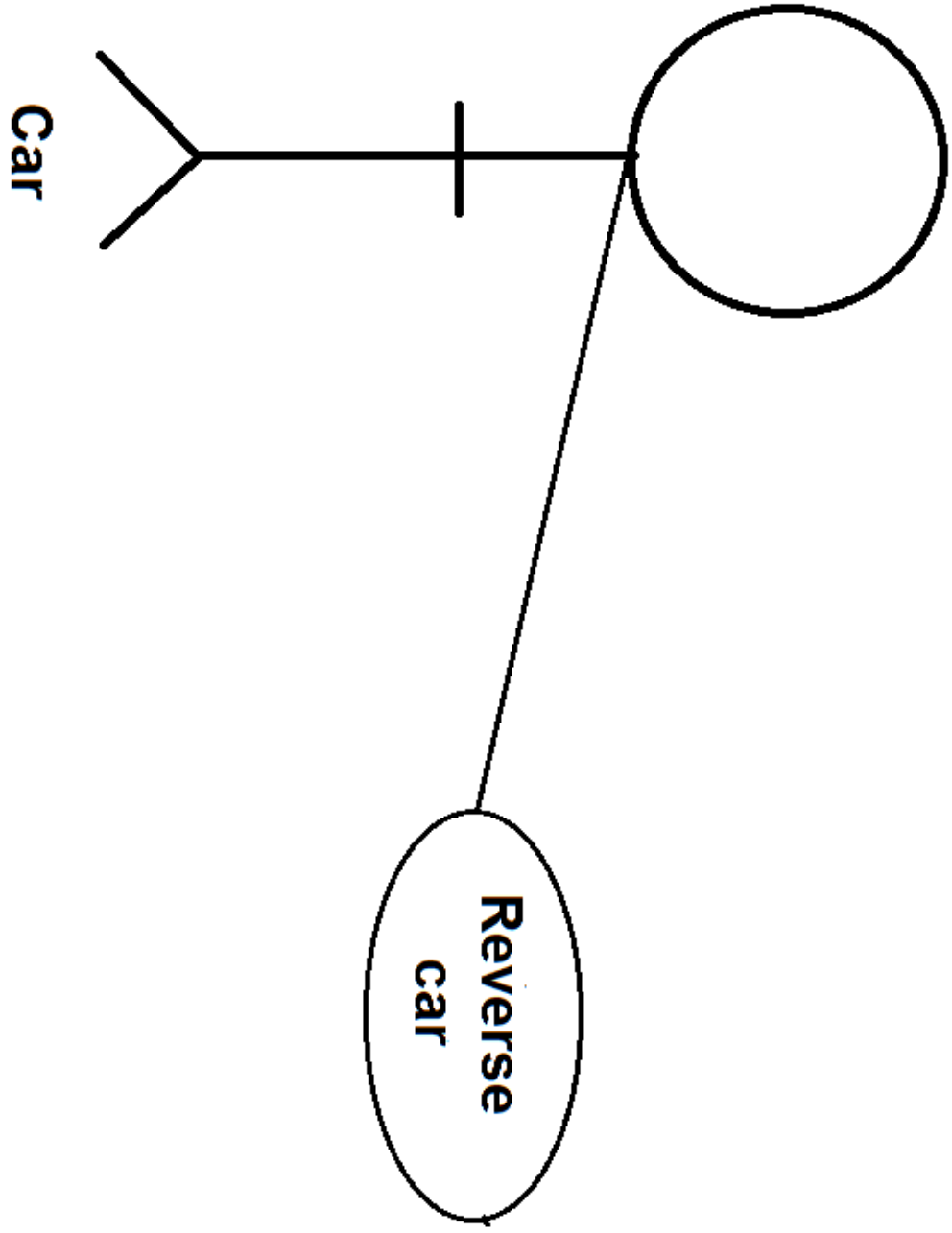

Figure 52: UseCase Reverse Car

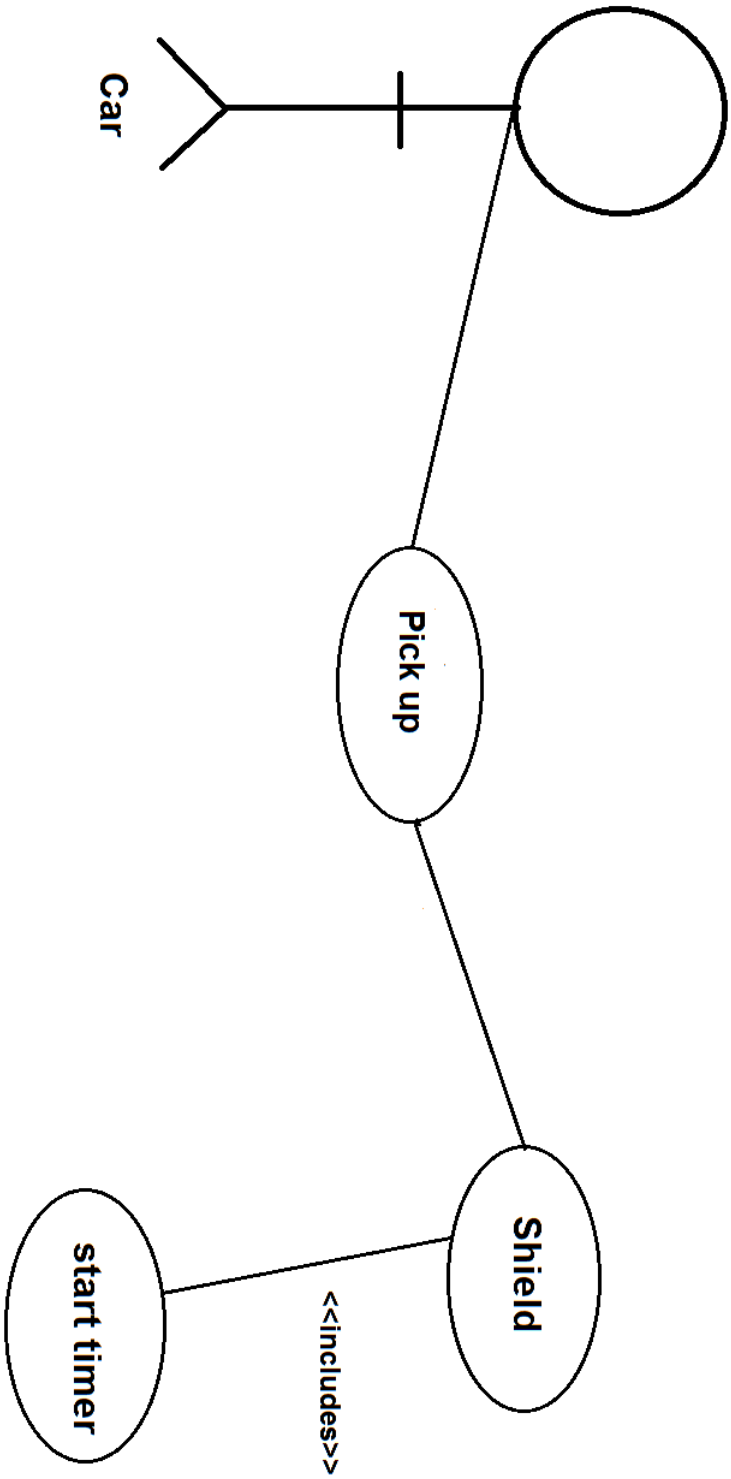

Figure 53: UseCase Pickup Shield

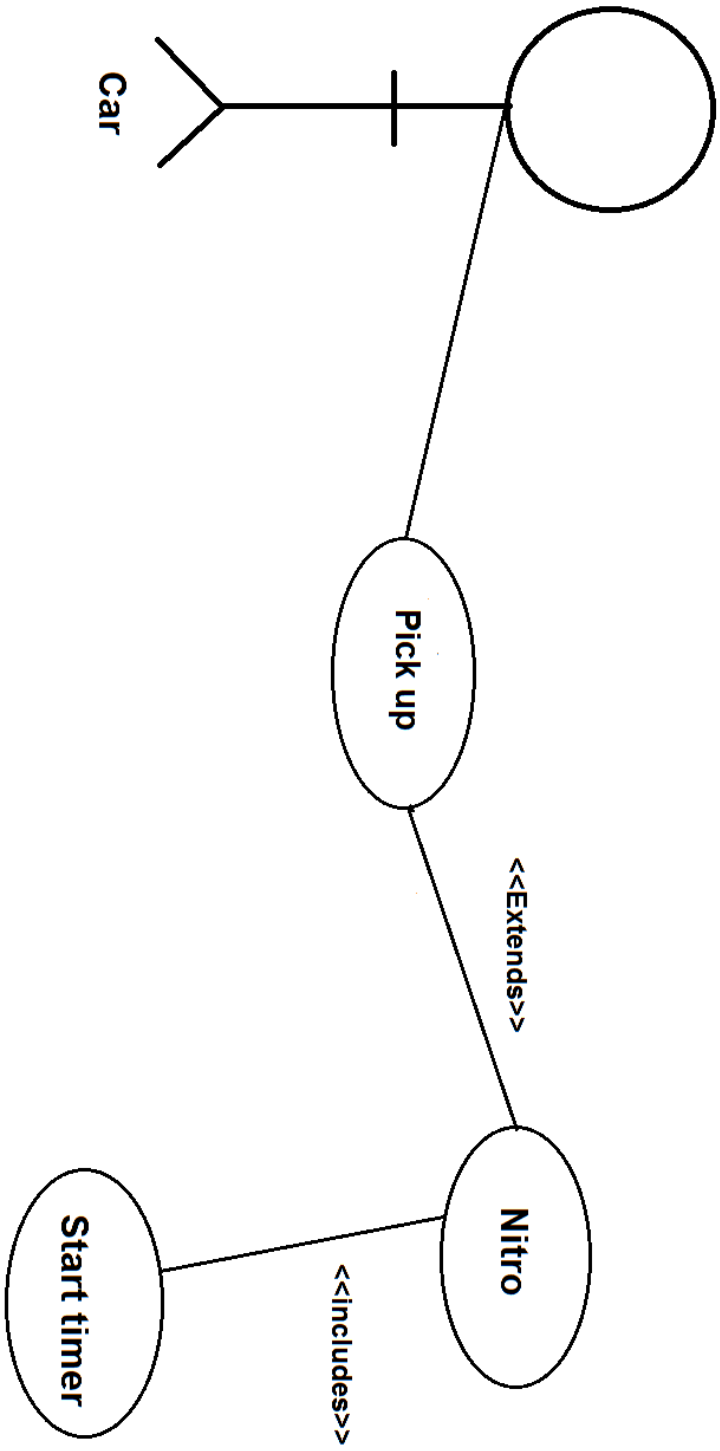

Figure 54: UseCase Pickup Nitro

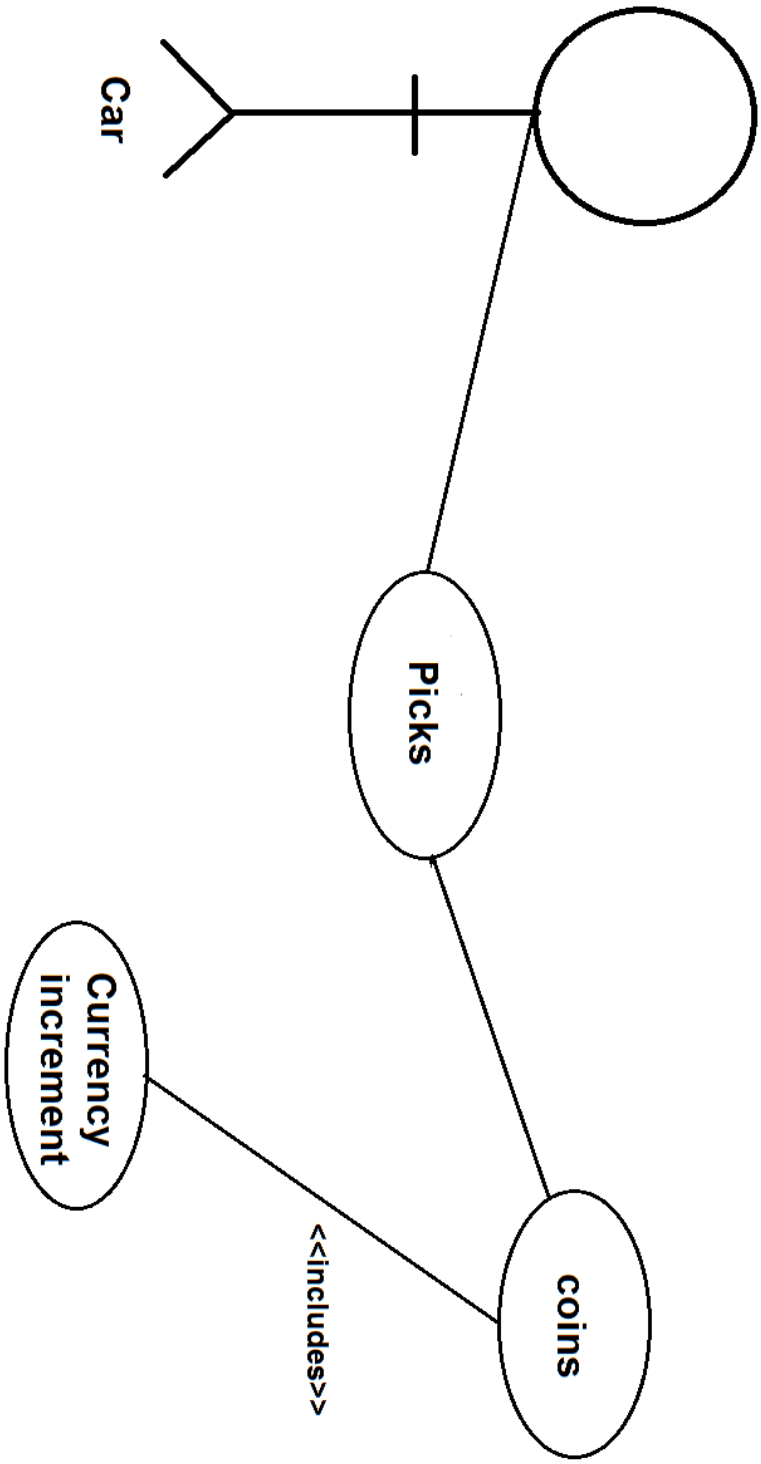

Figure 55: UseCase Pickup Coin

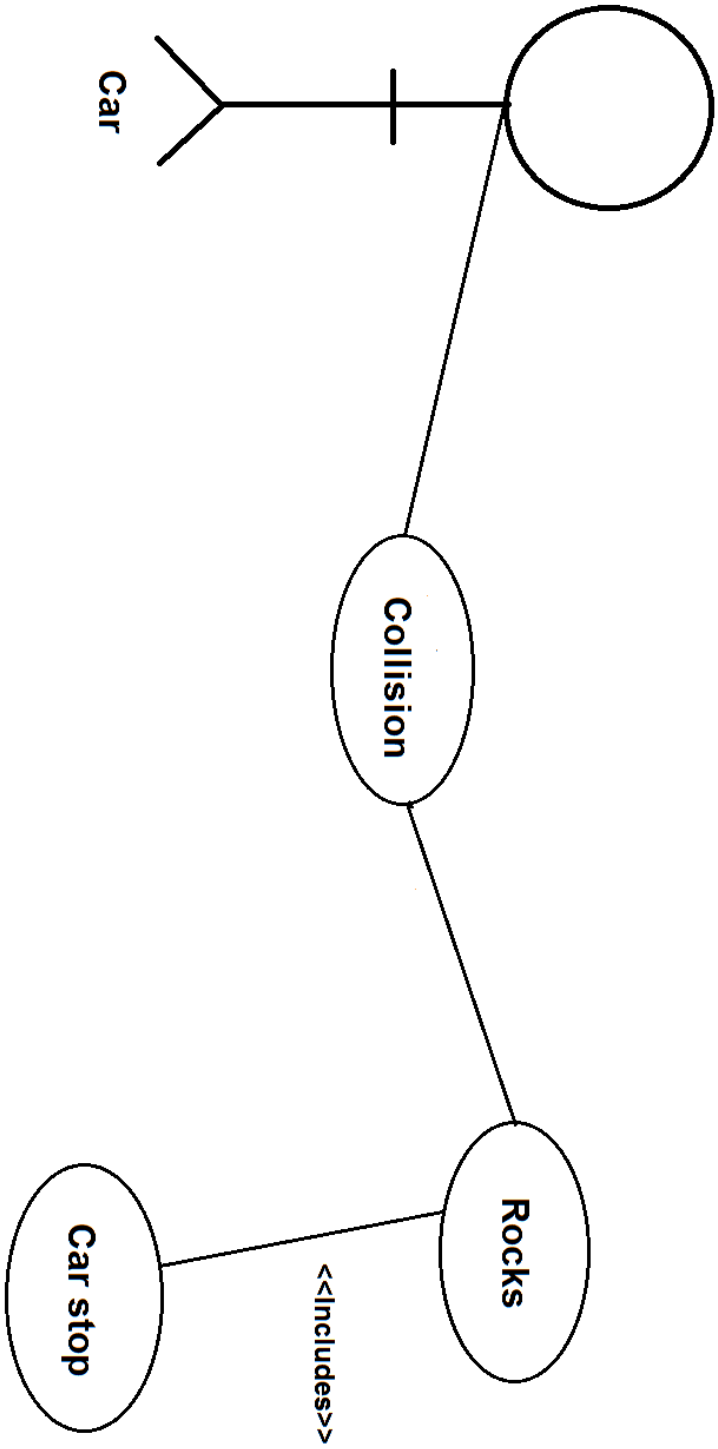

Figure 56: UseCase Collide with Rock

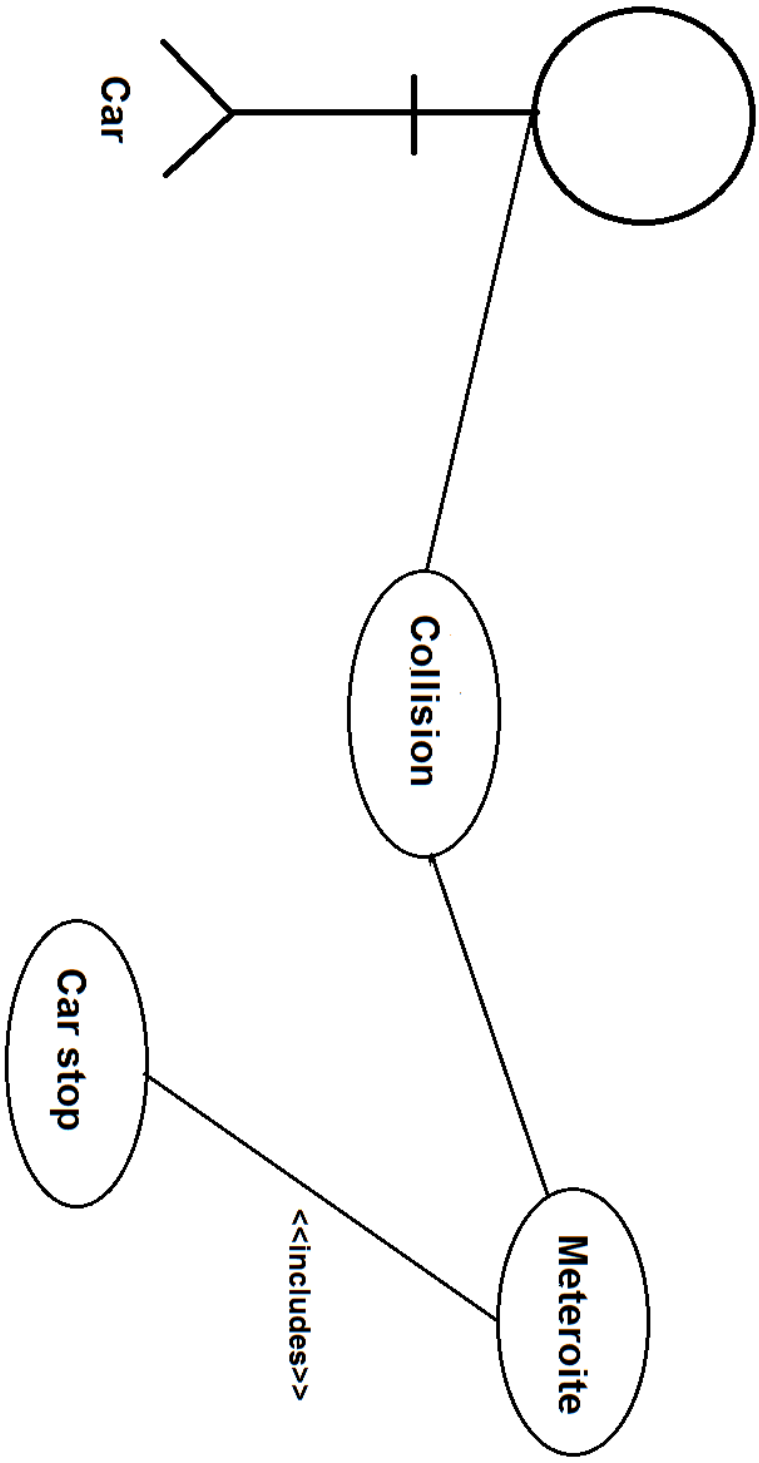

Figure 57: UseCase Collide with Meteriote

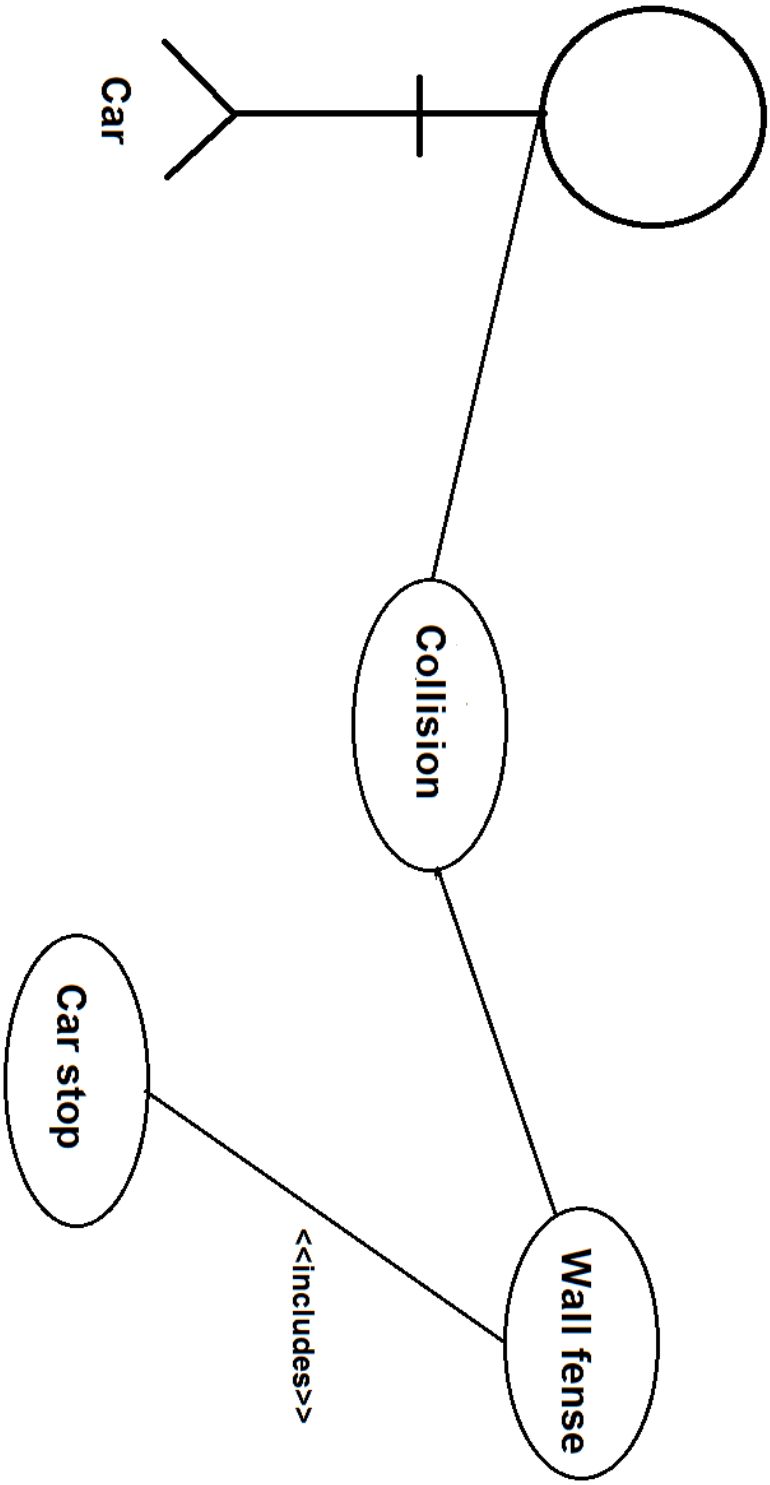

Figure 58: UseCase Collide with Fence

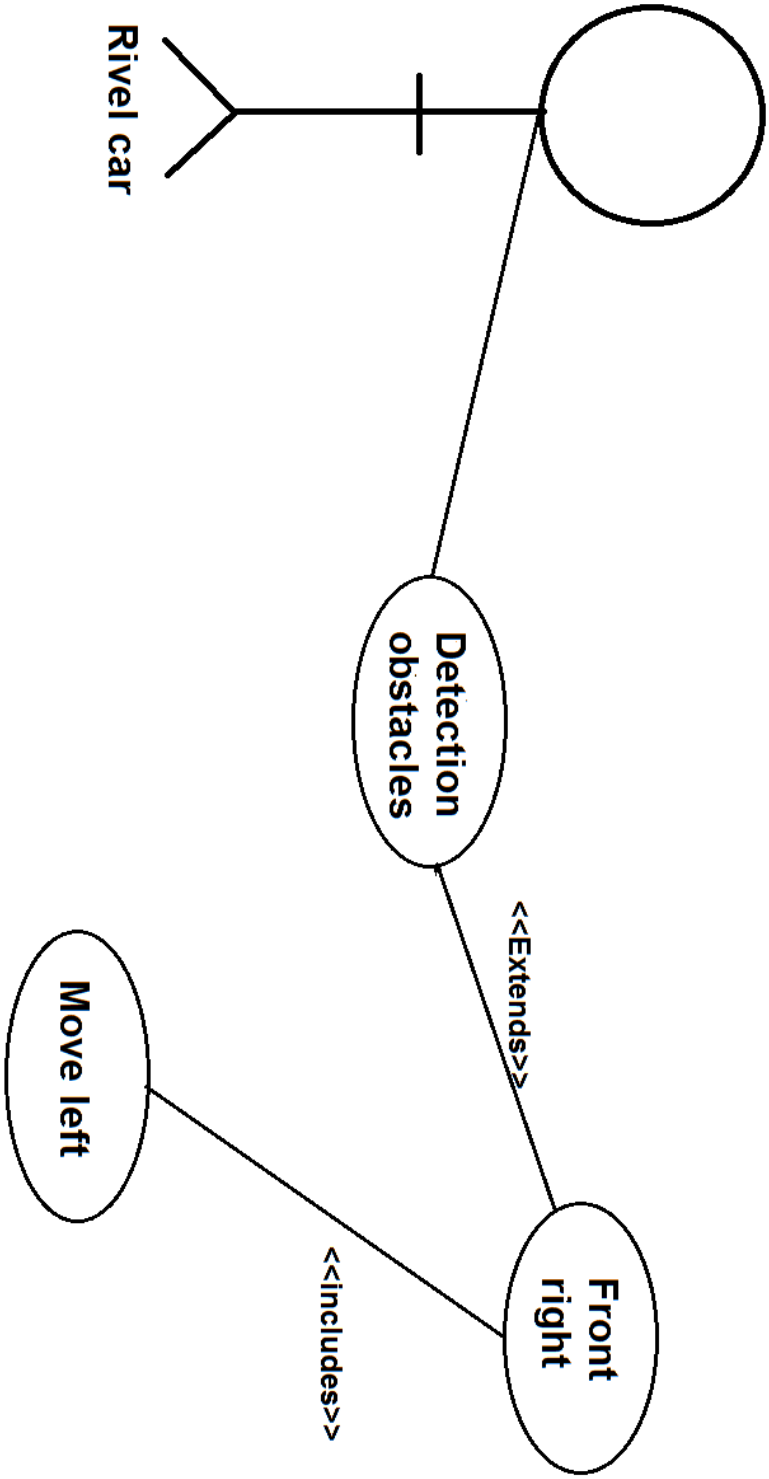

Figure 59: UseCase AI car move left

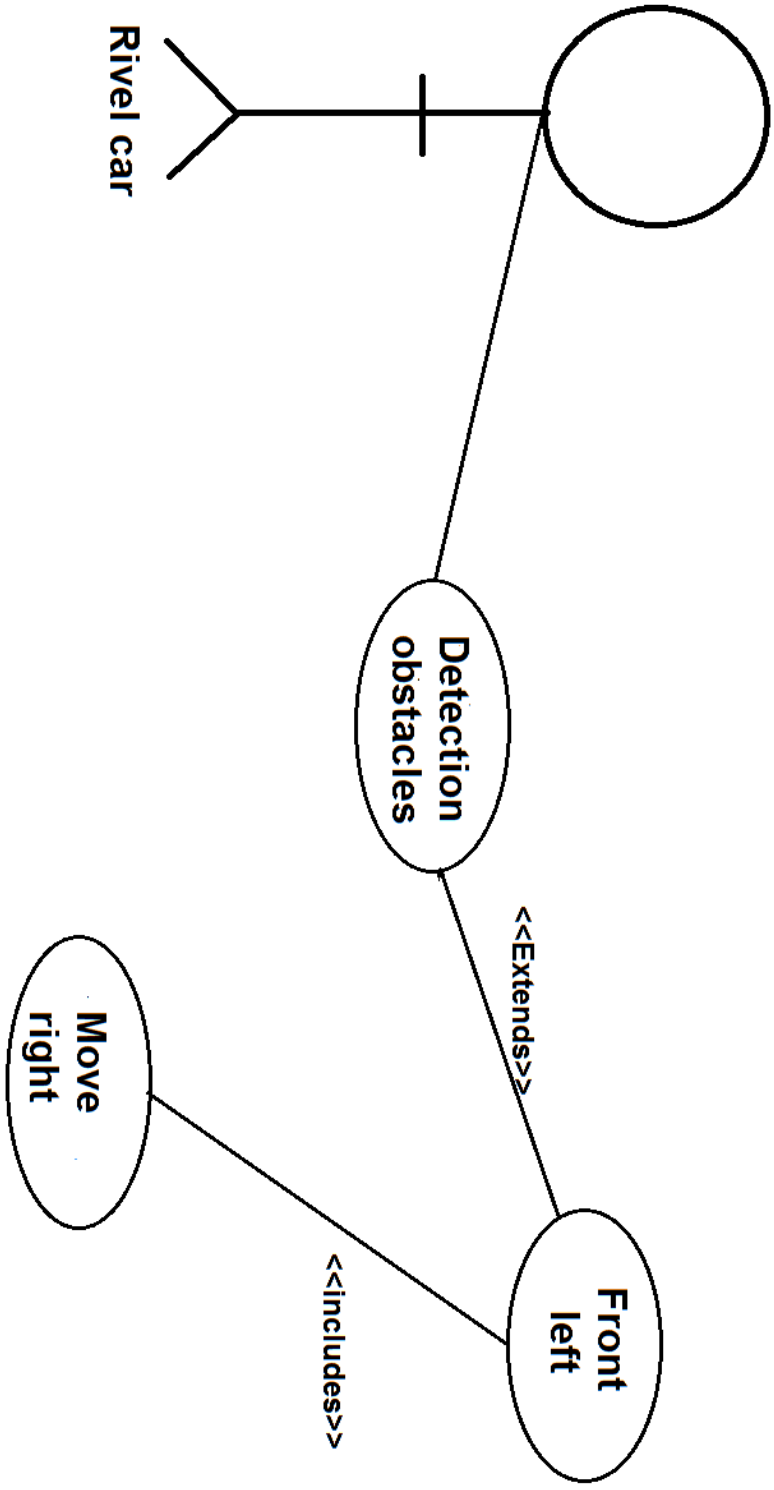

Figure 60: UseCase AI car move right

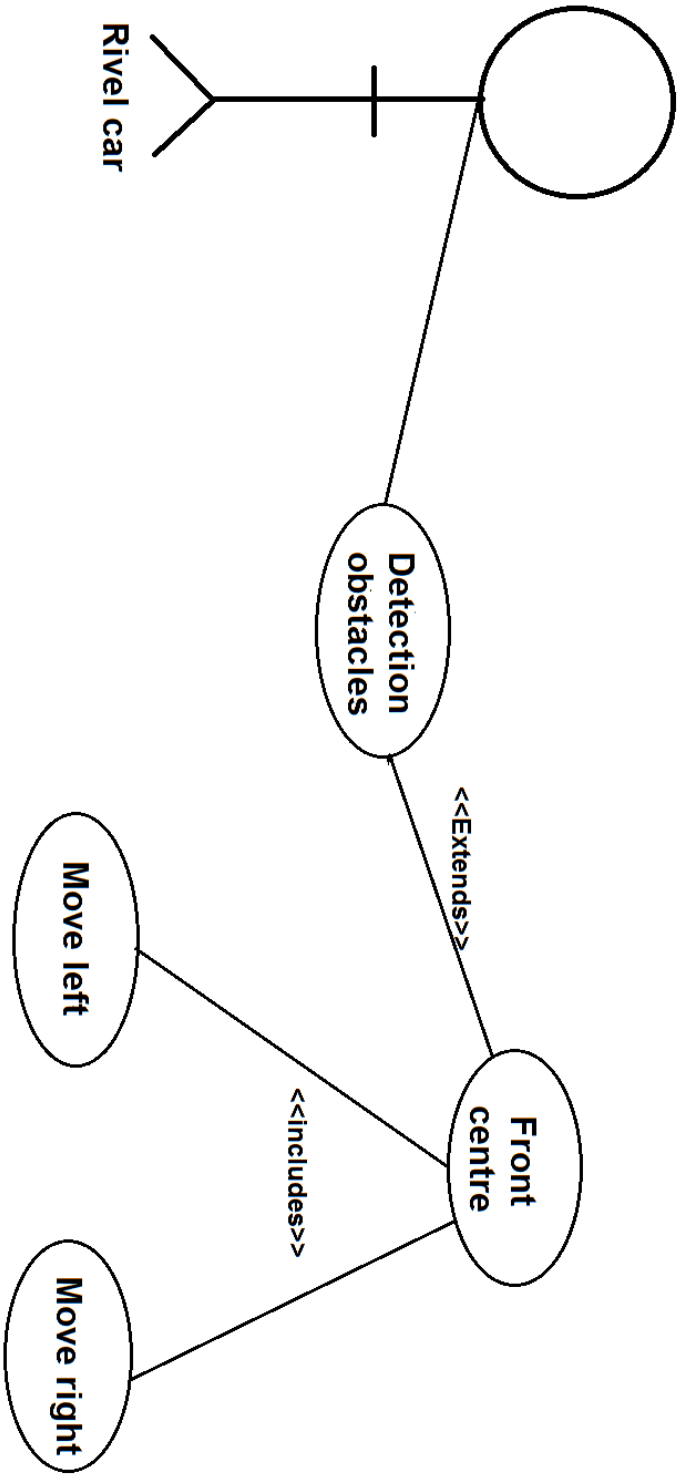

Figure 61: UseCase AI car turn

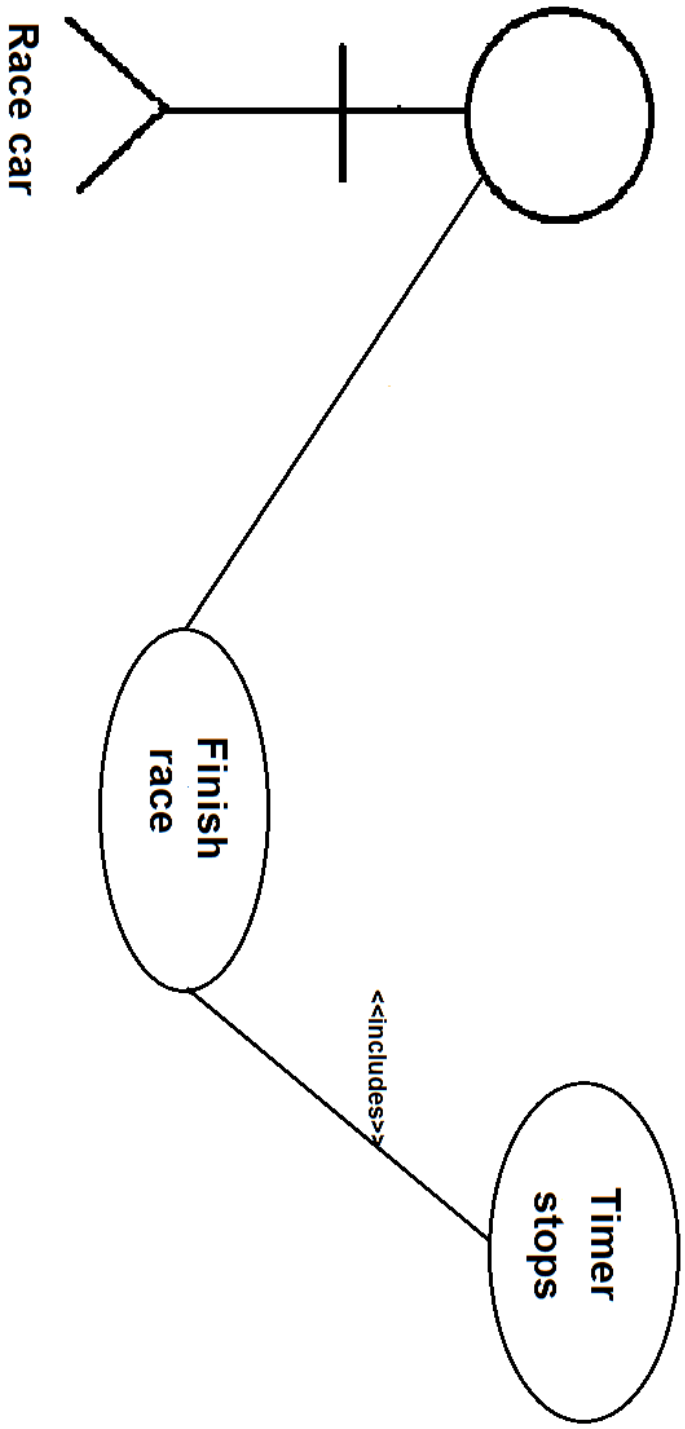

Figure 62: UseCase Finish Race

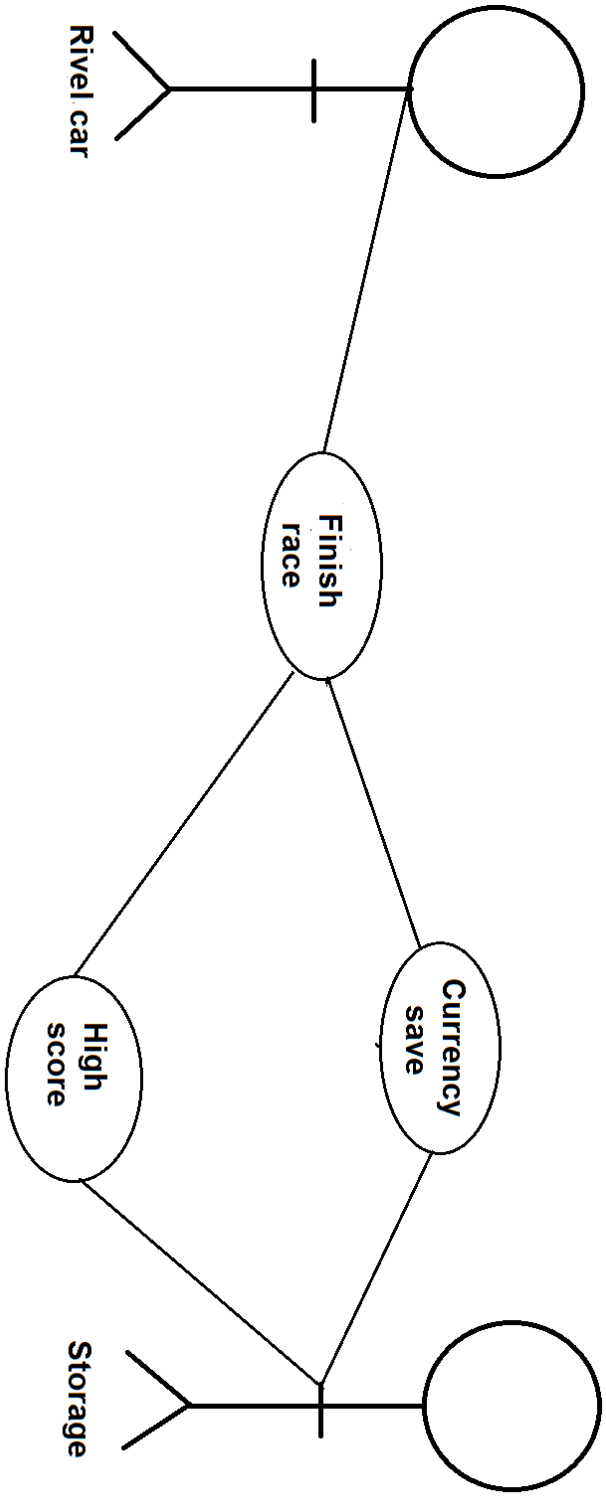

Figure 63: UseCase currency and high score save

6.3 Activity Diagrams :

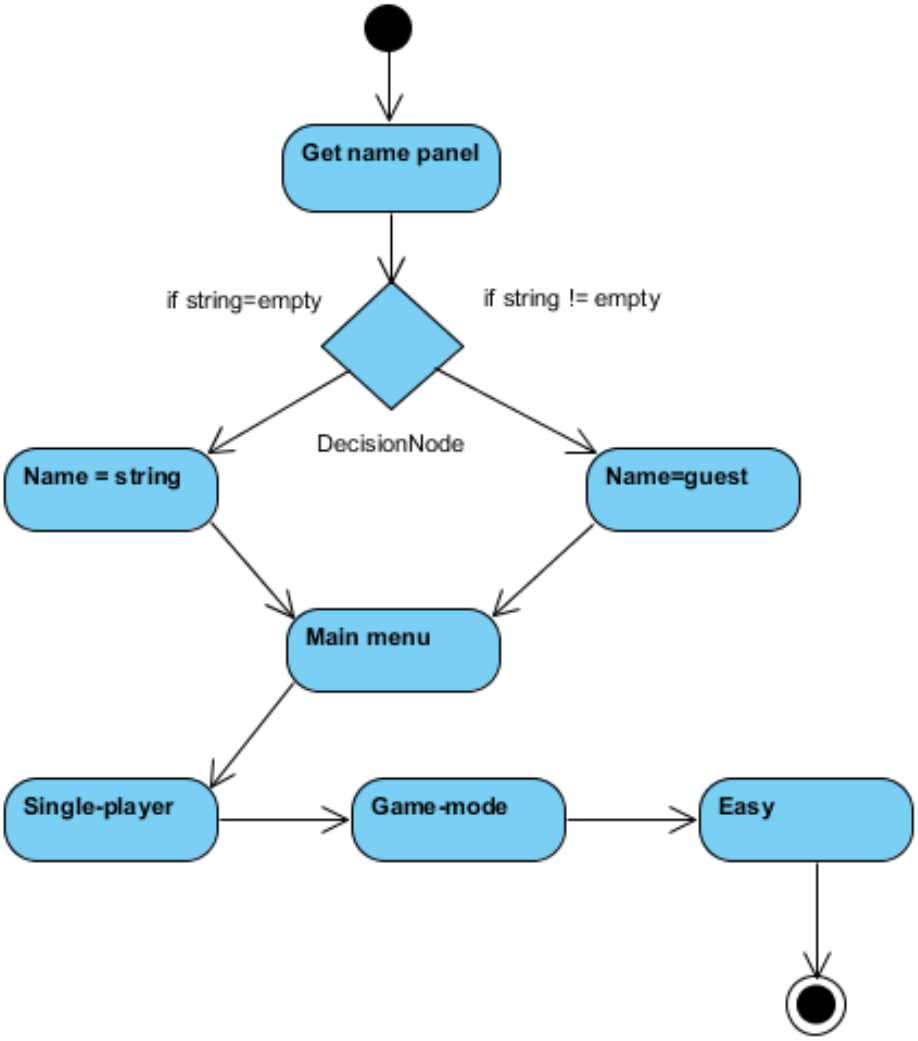

Figure 64: Activity Easy mode

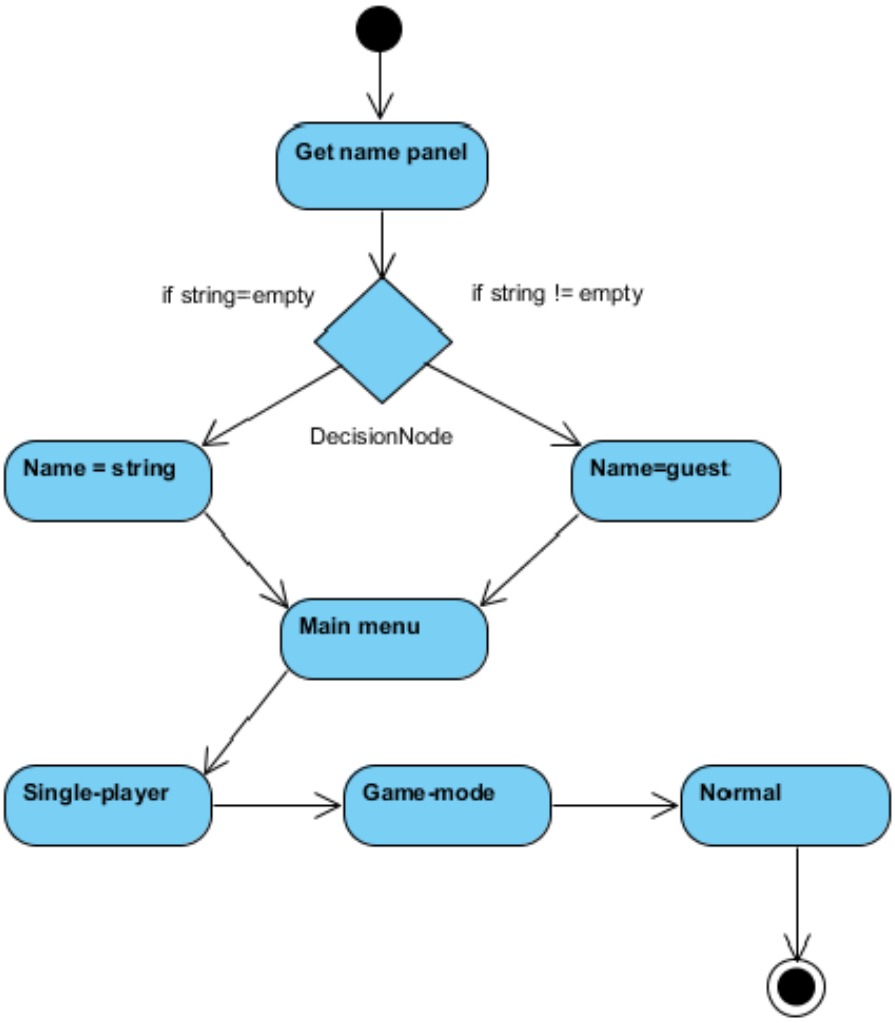

Figure 65: Activity Normal Mode

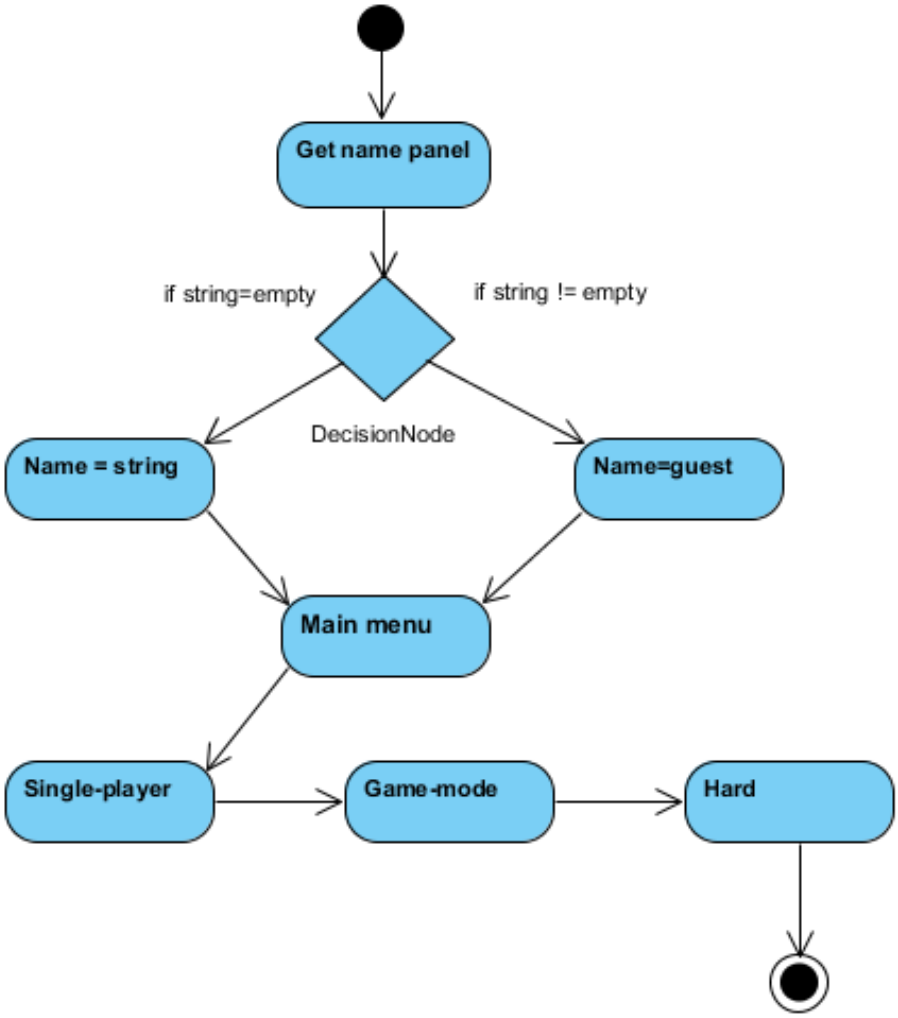

Figure 66: Activity Hard Mode

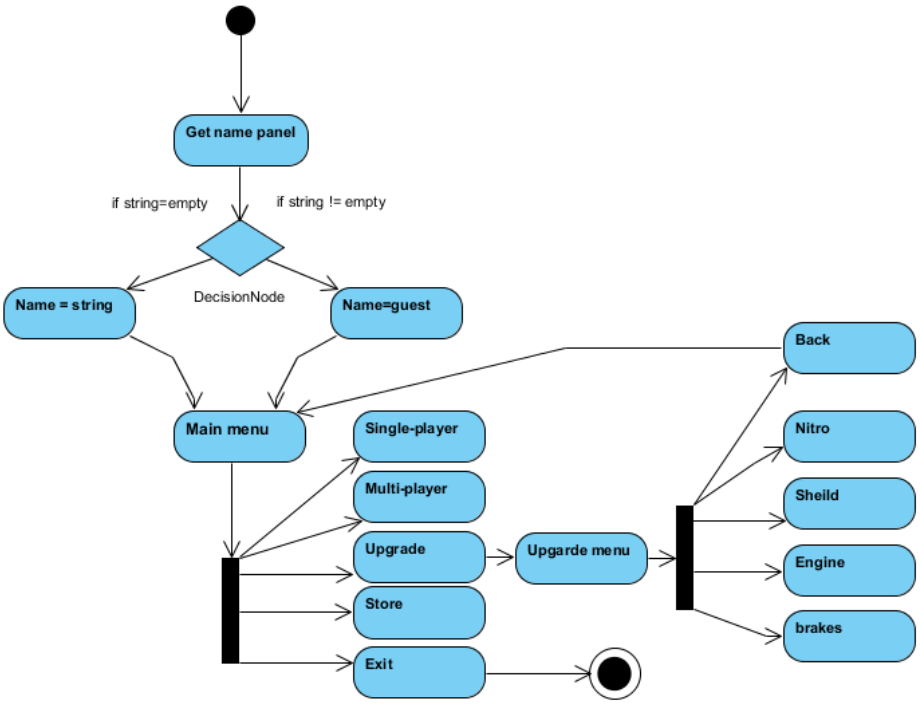

Figure 67: Activity Upgrade Menu

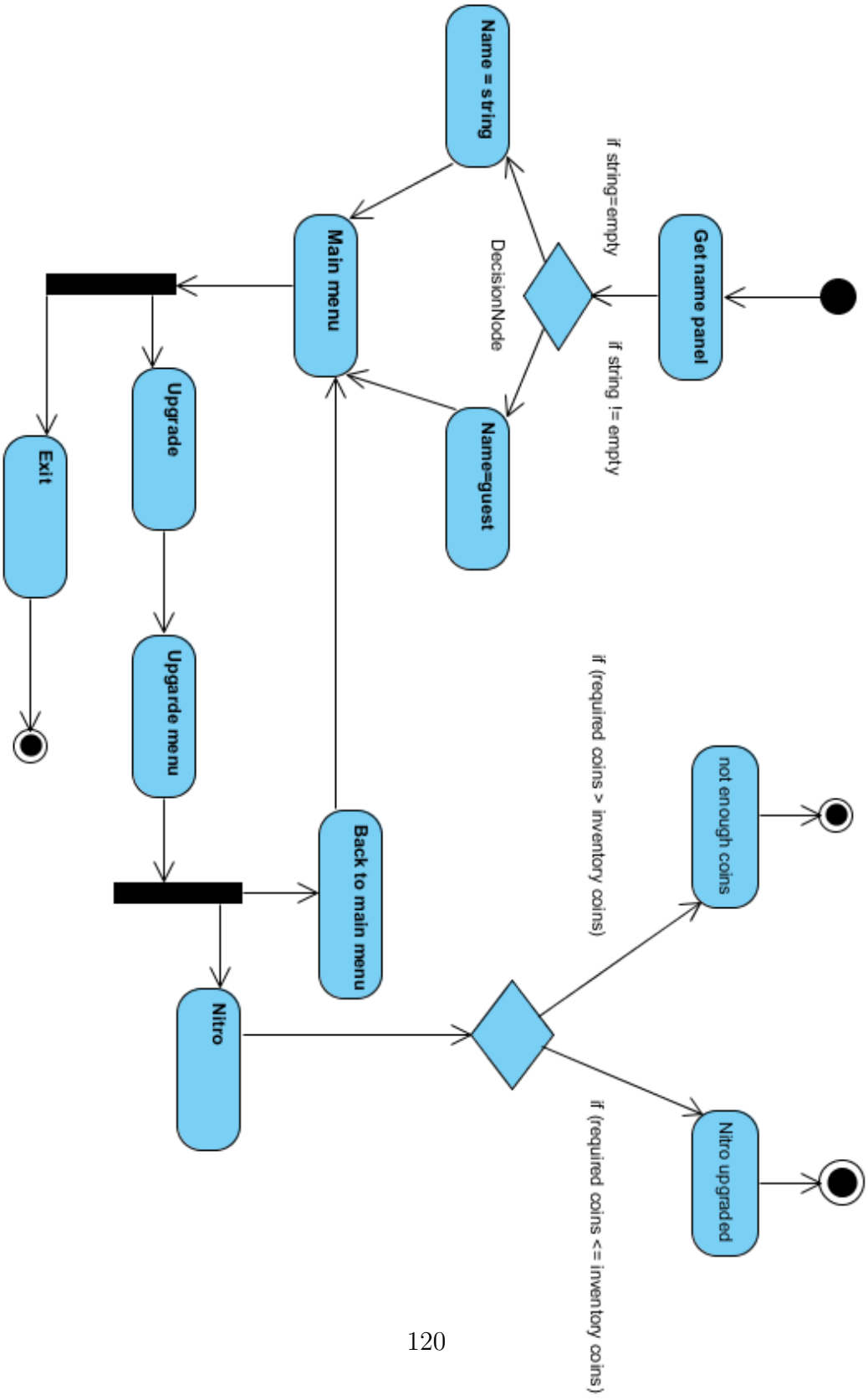

Figure 68: Activity Nitro Upgrade

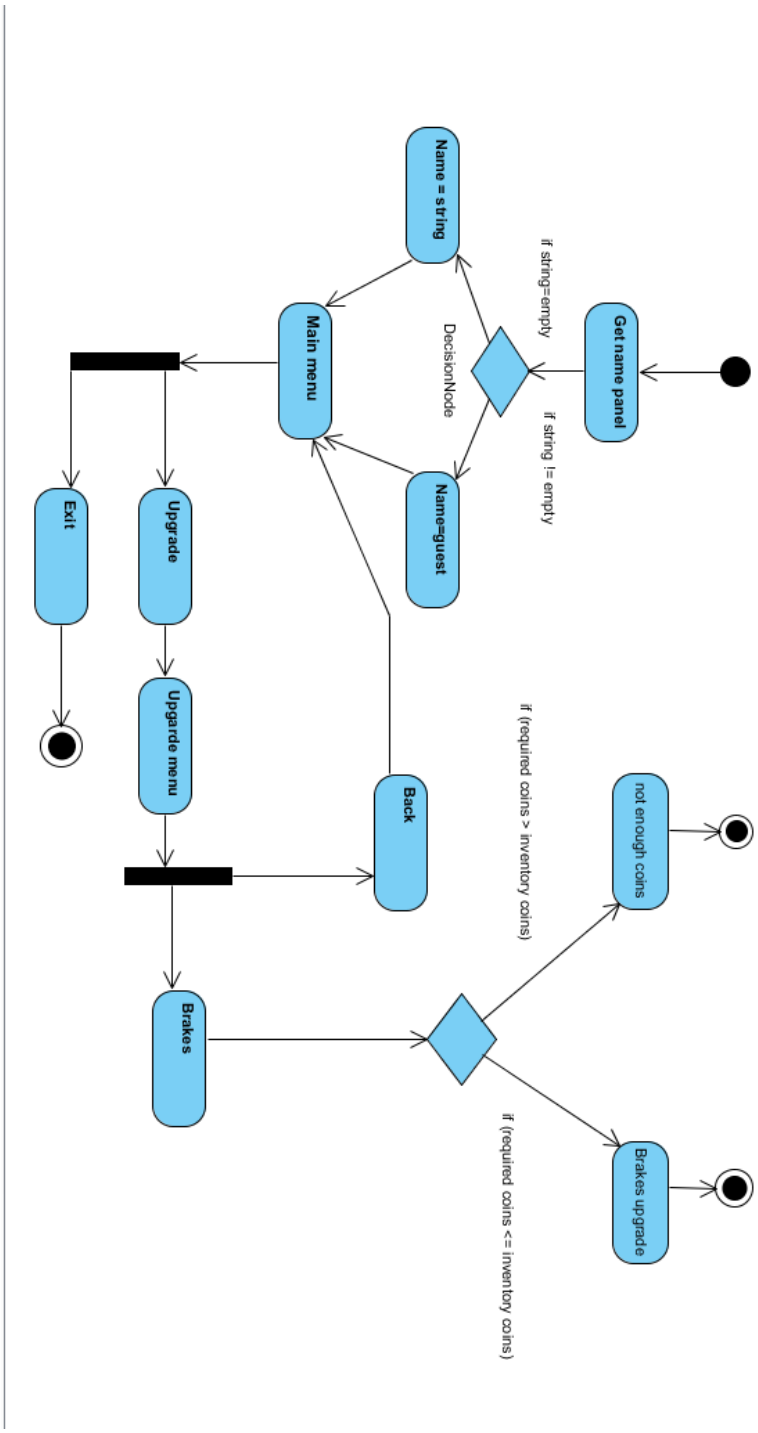

Figure 69: Activity Brakes Upgrade

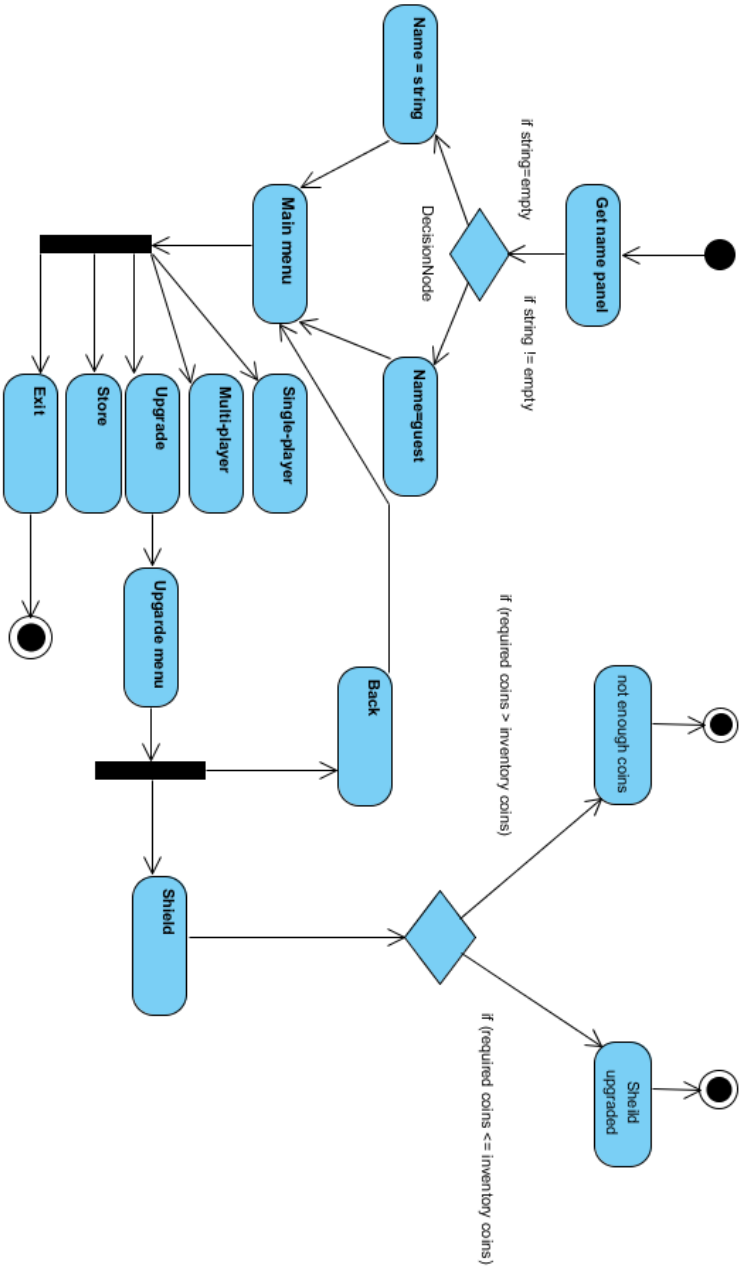

Figure 70: Activity Shield Upgrade

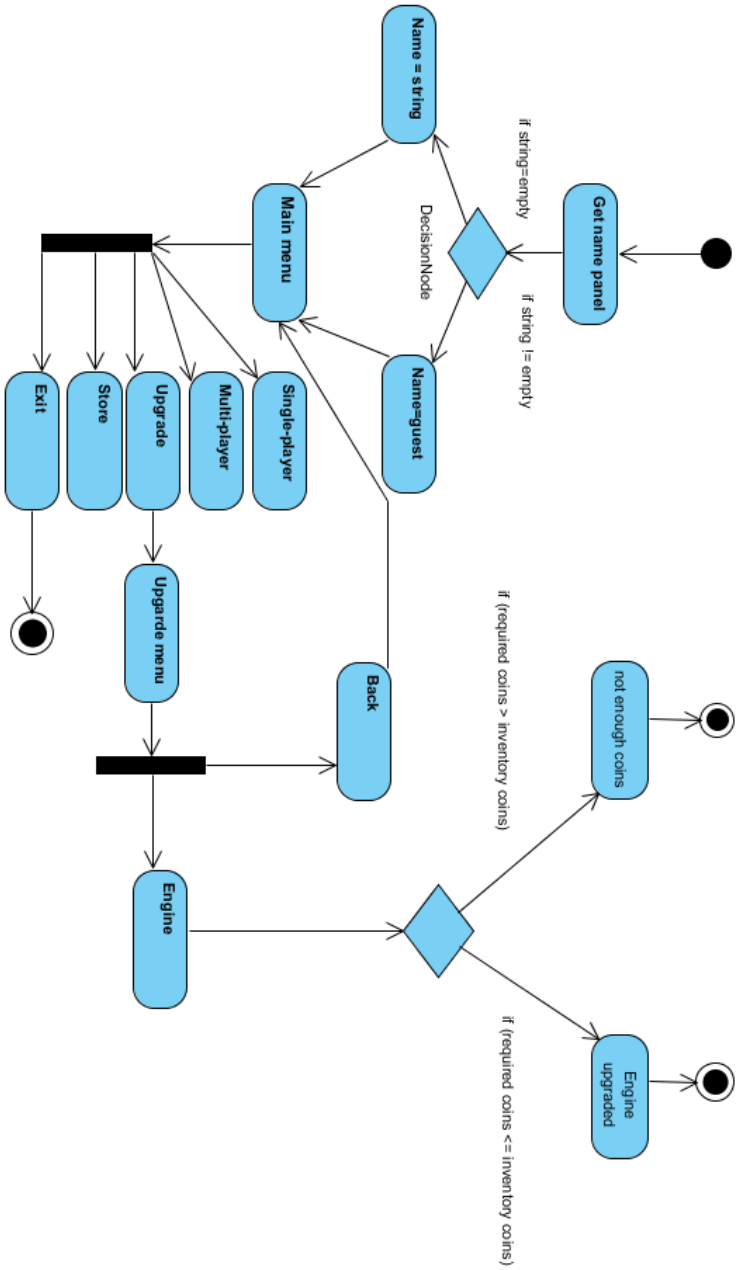

Figure 71: Activity Engine Upgrade

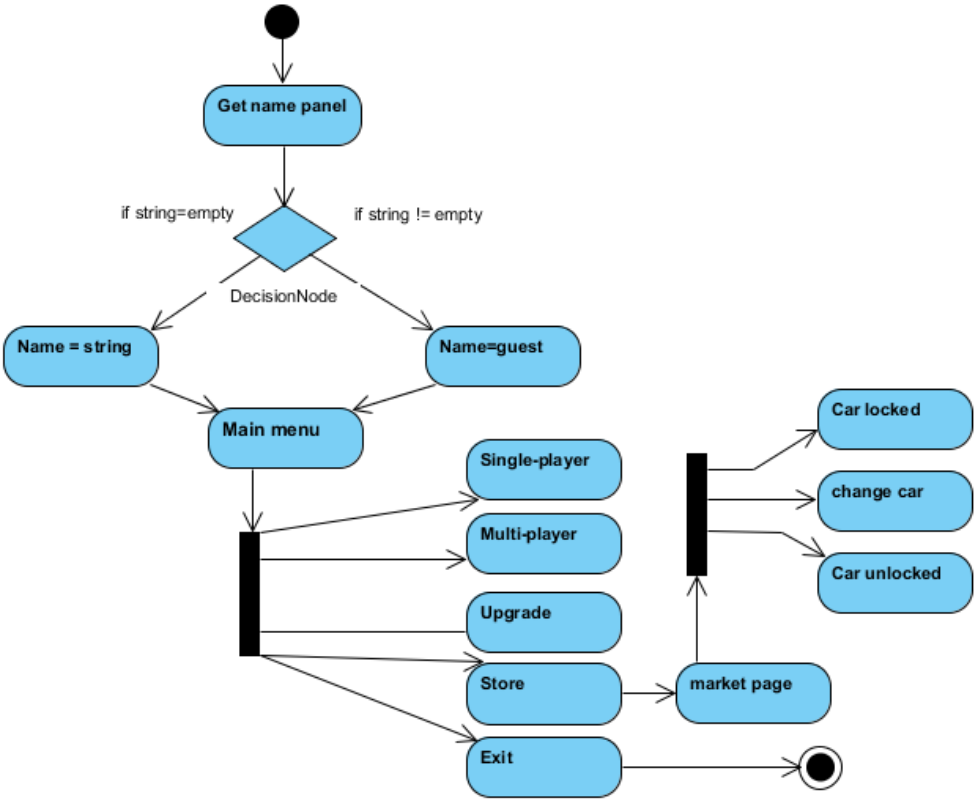

Figure 72: Activity Market Menu

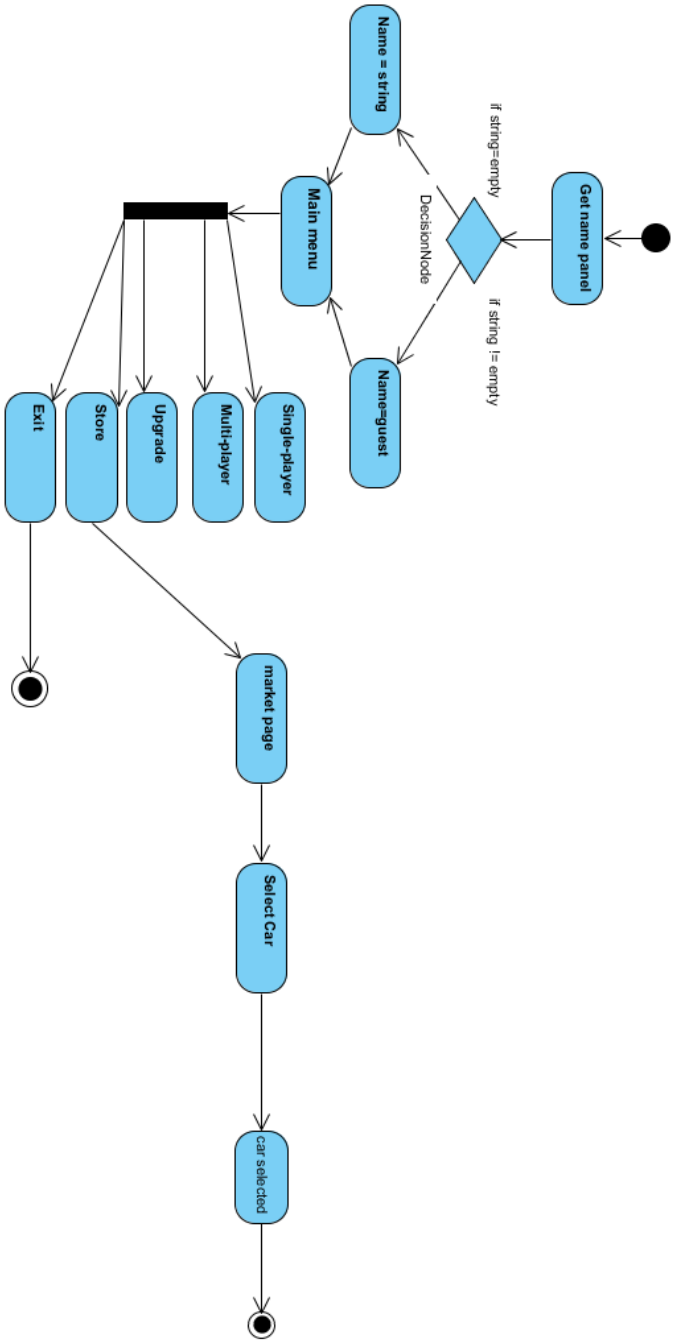

Figure 73: Activity Car Selected

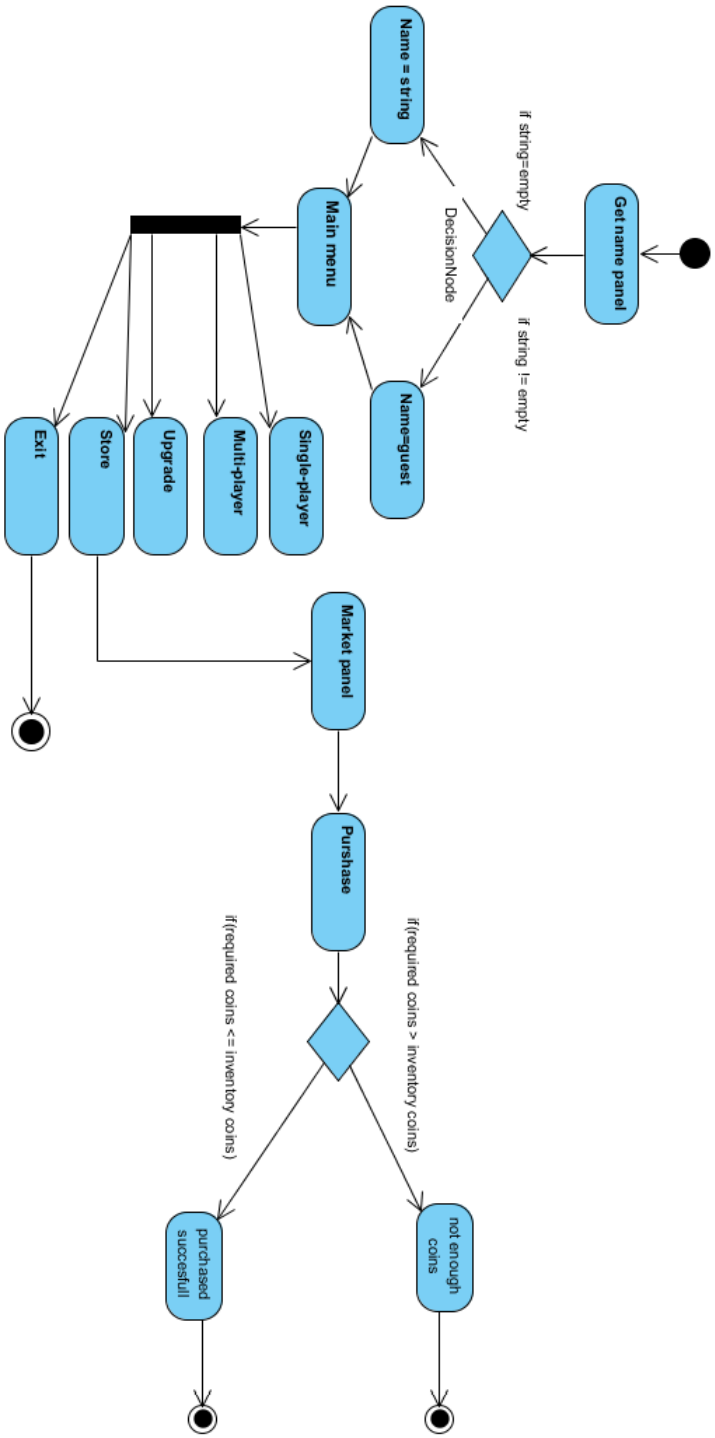

Figure 74: Activity Purchase Car

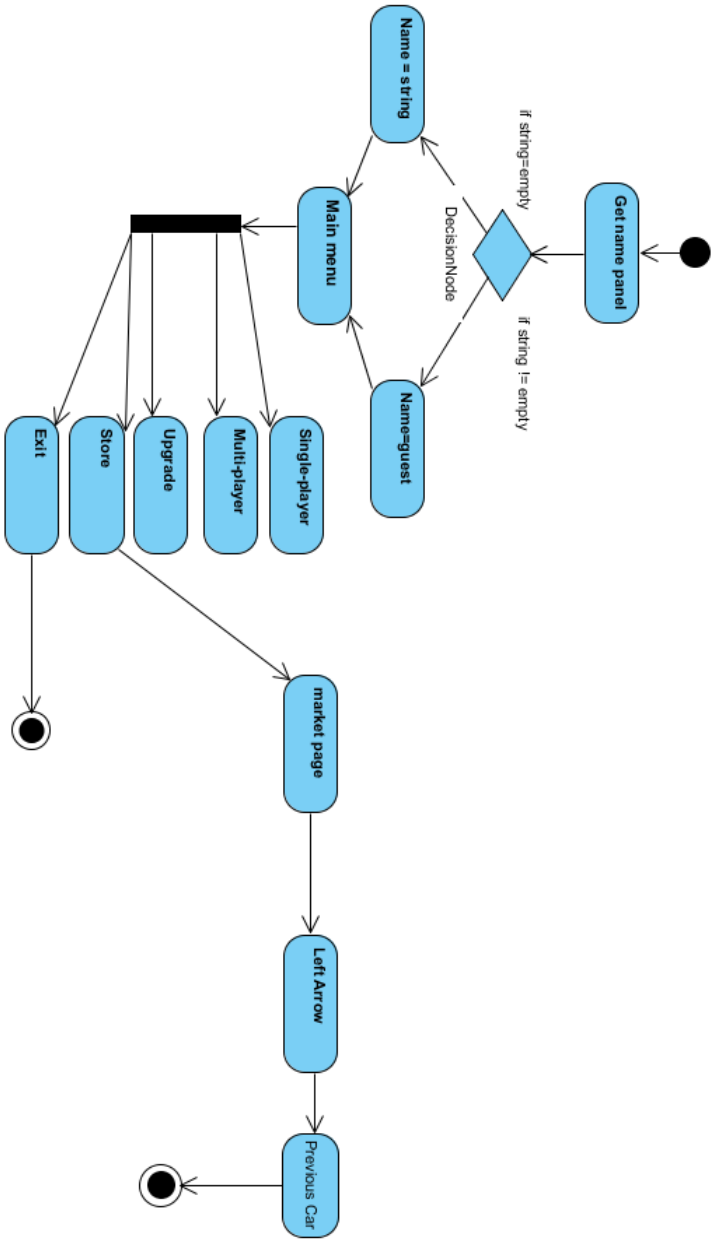

Figure 75: Activity Previous Car

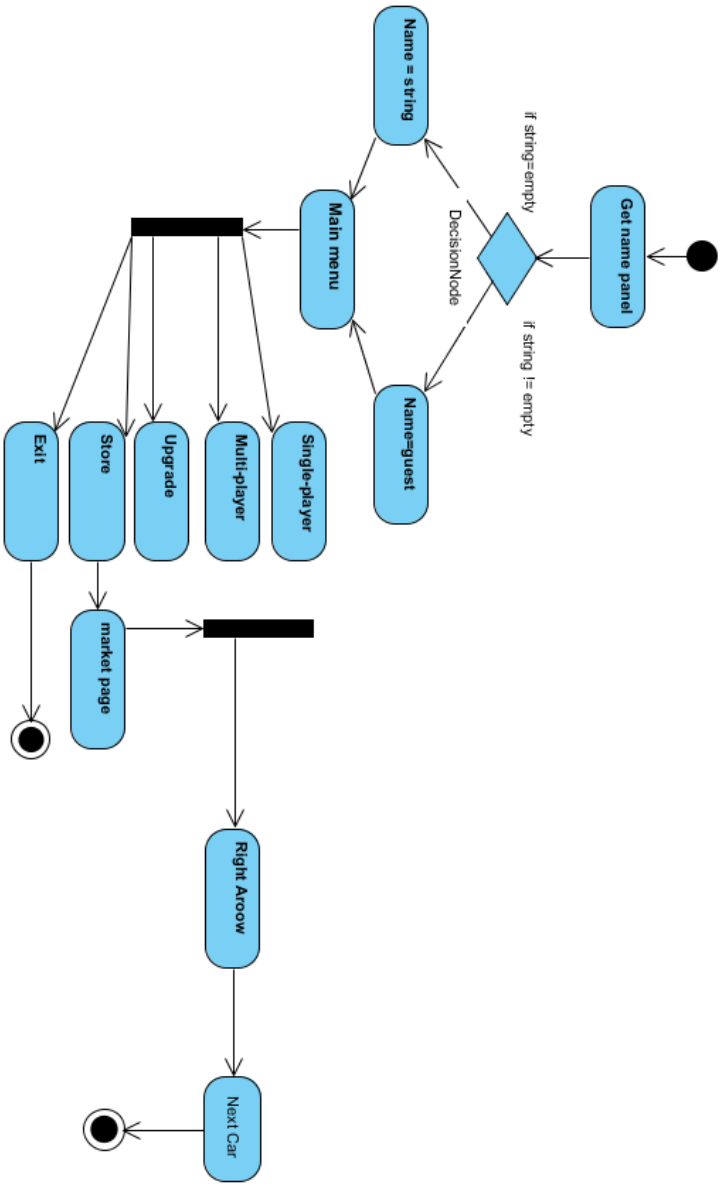

Figure 76: Activity Next Car

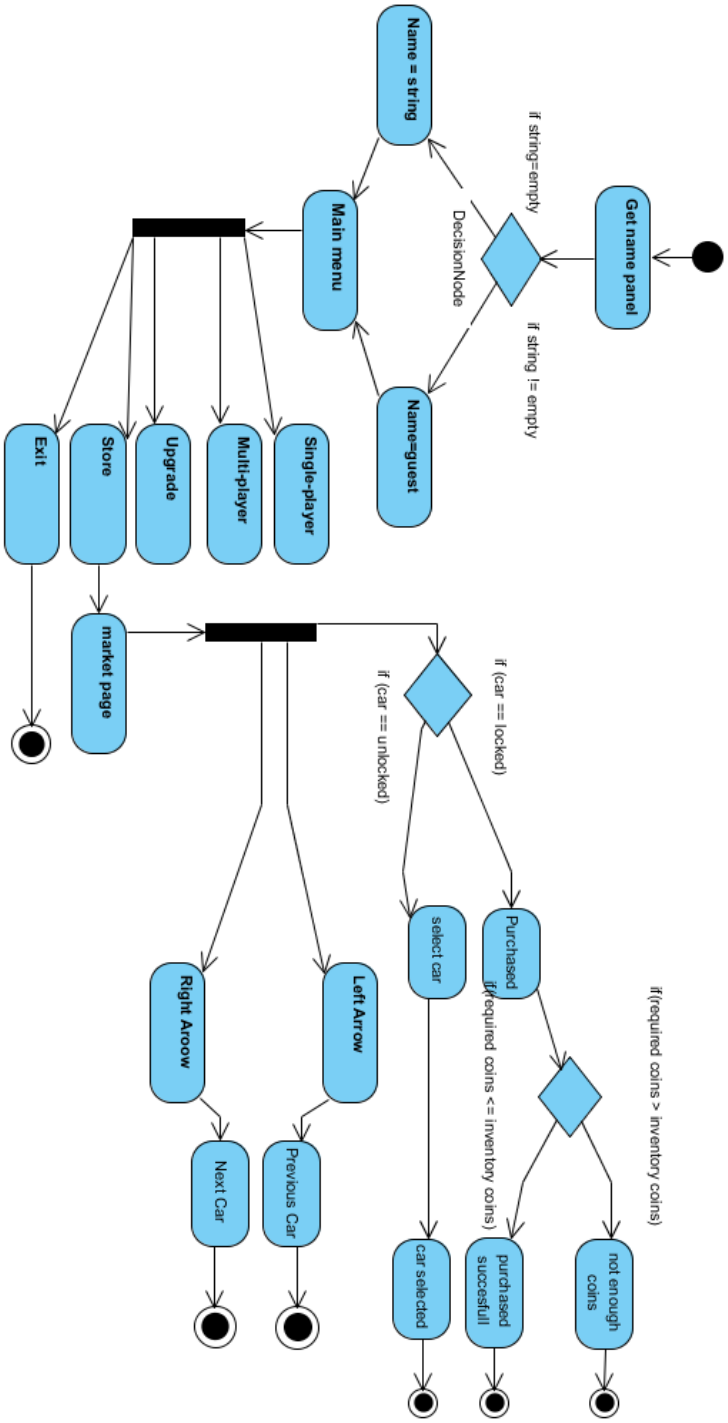

Figure 77: Activity Market Menu

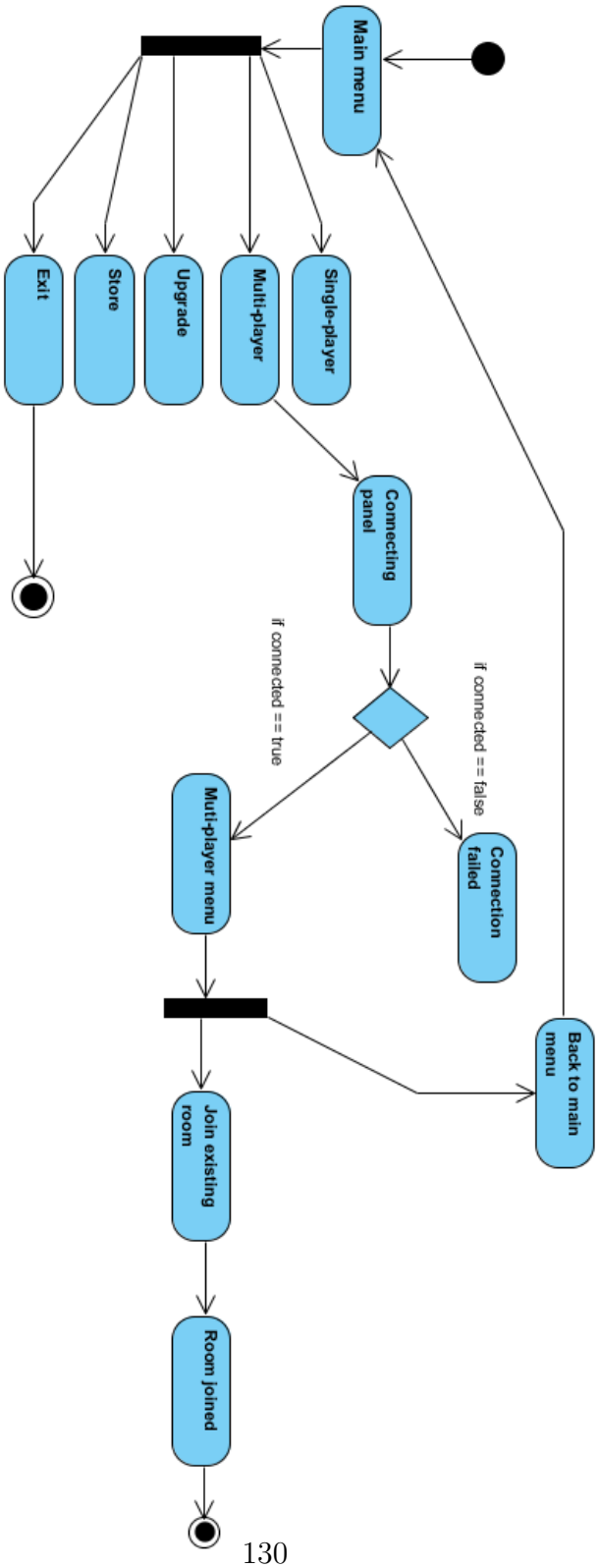

Figure 78: Activity Room Joined

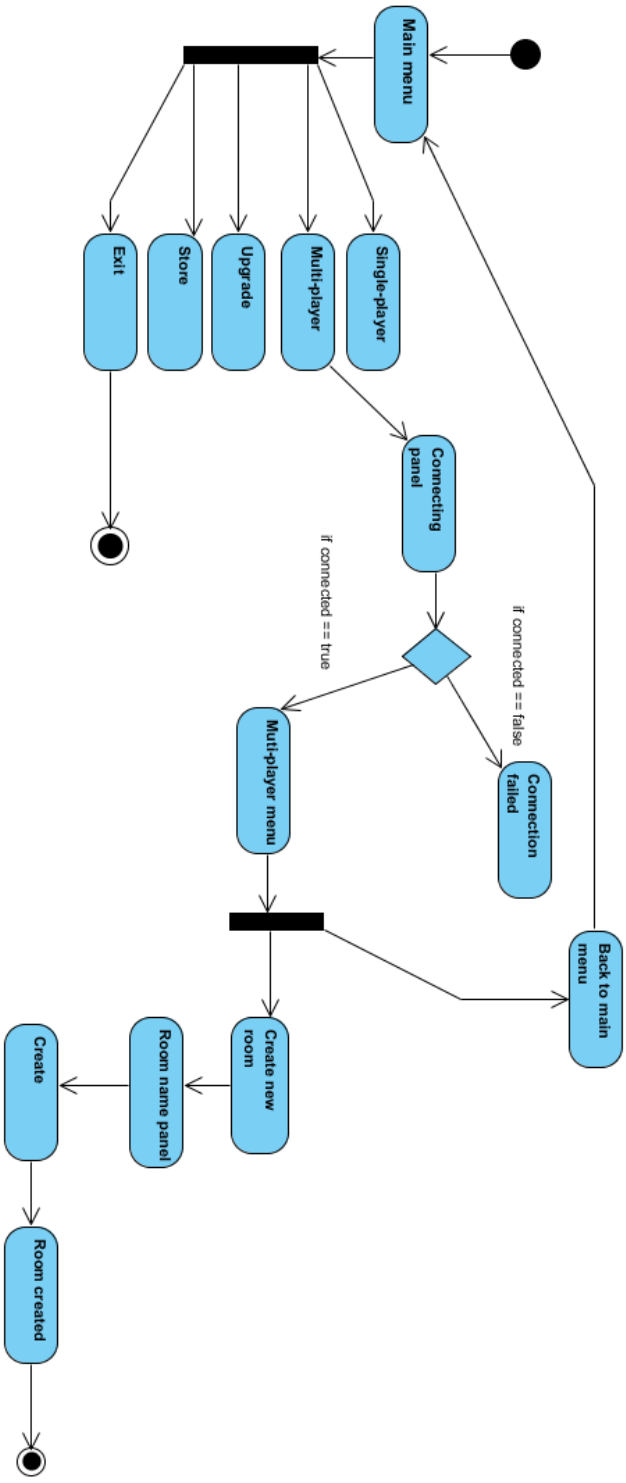

Figure 79: Activity Create Room

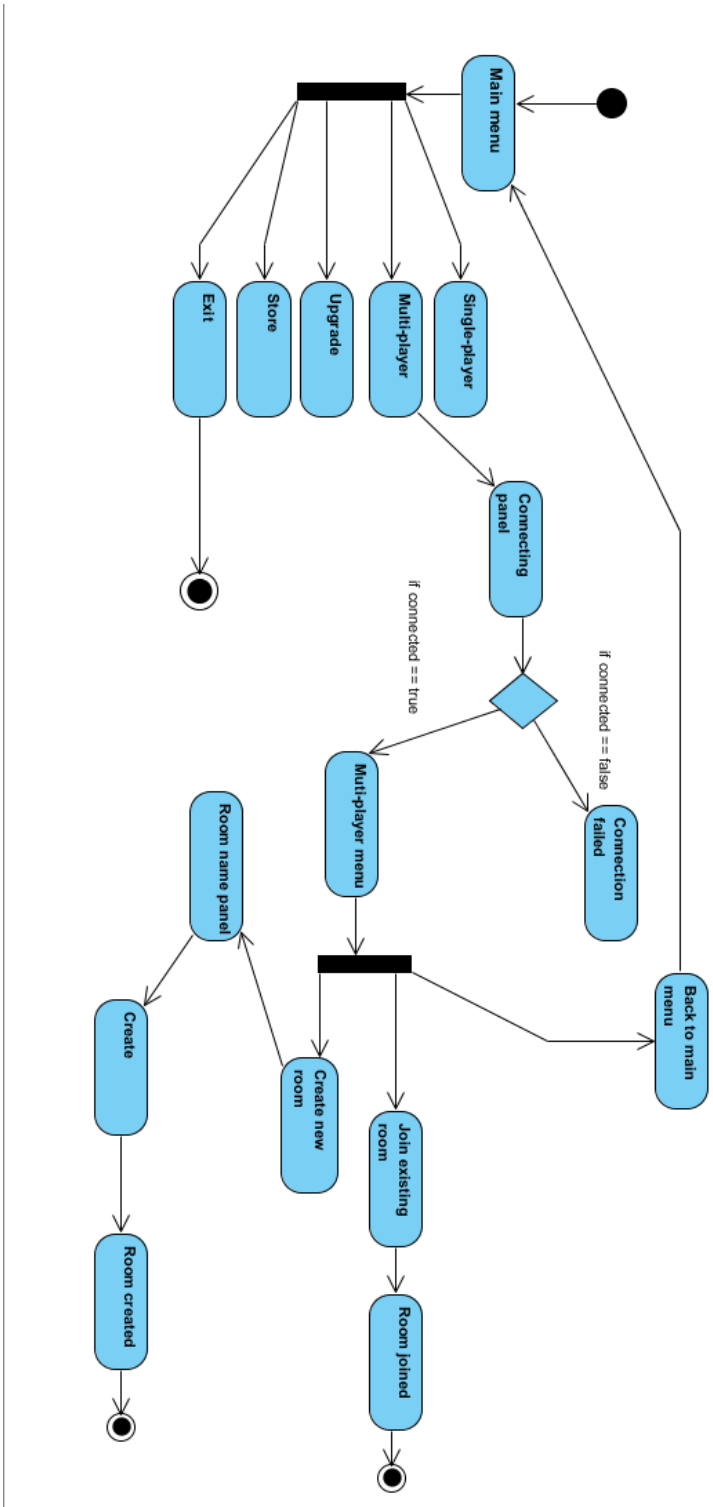

Figure 80: Activity MultiPlayer

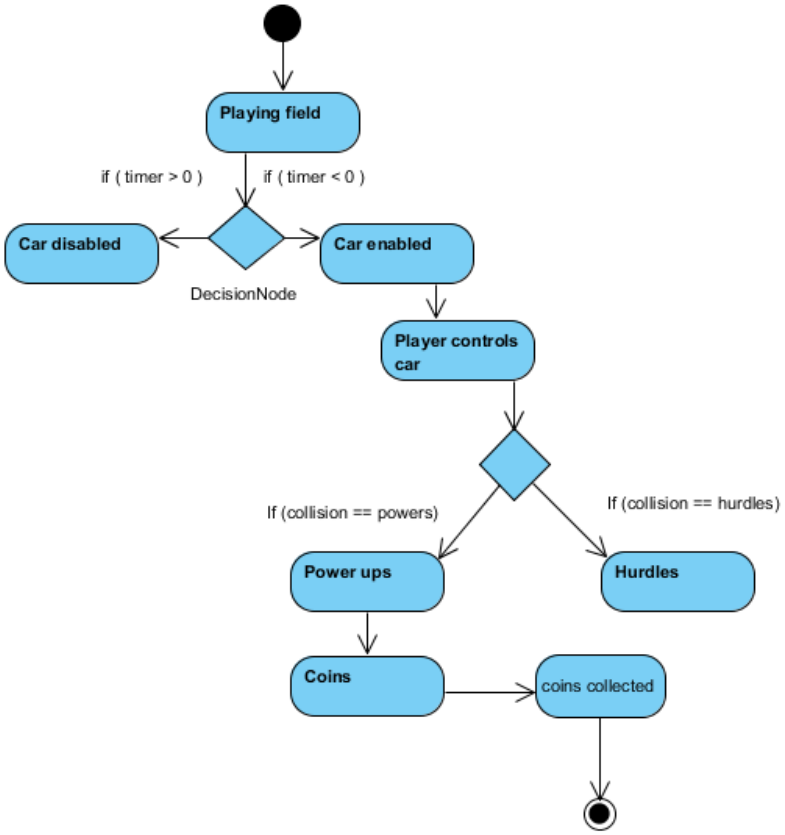

Figure 81: Activity Coin Pickup

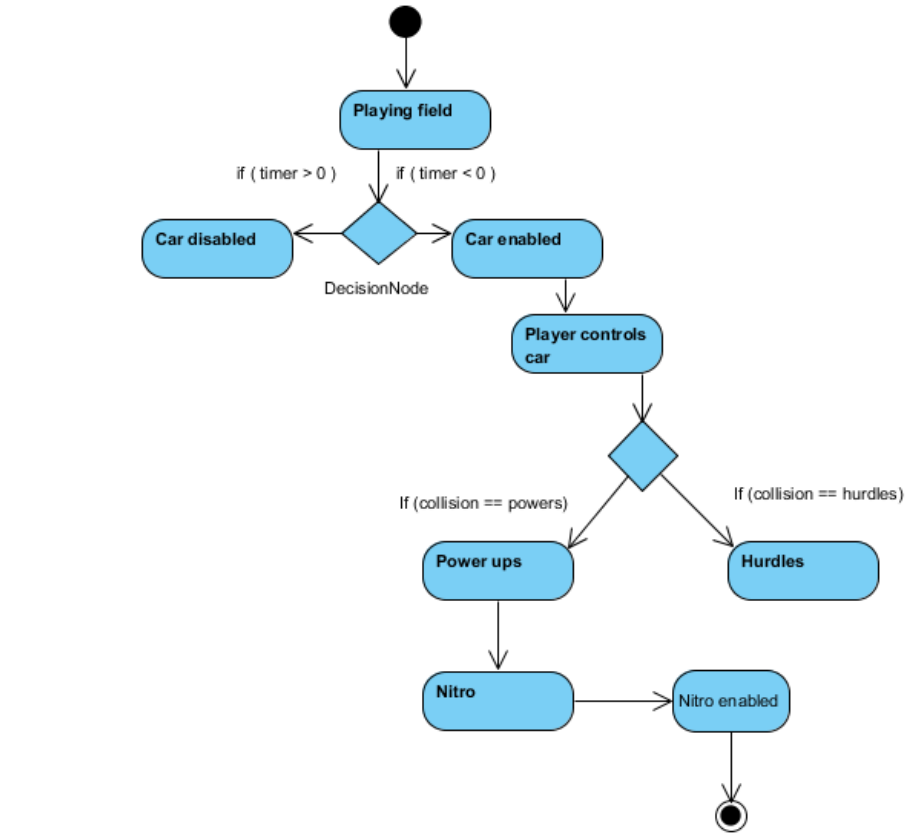

Figure 82: Activity Nitro pickup

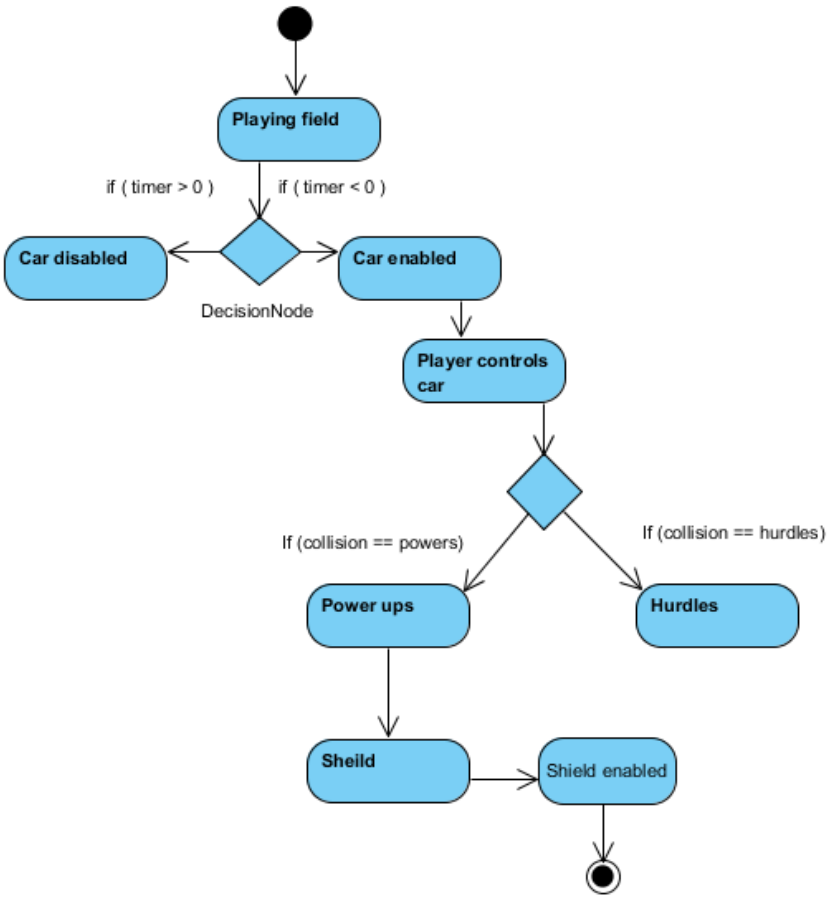

Figure 83: Activity shield pickup

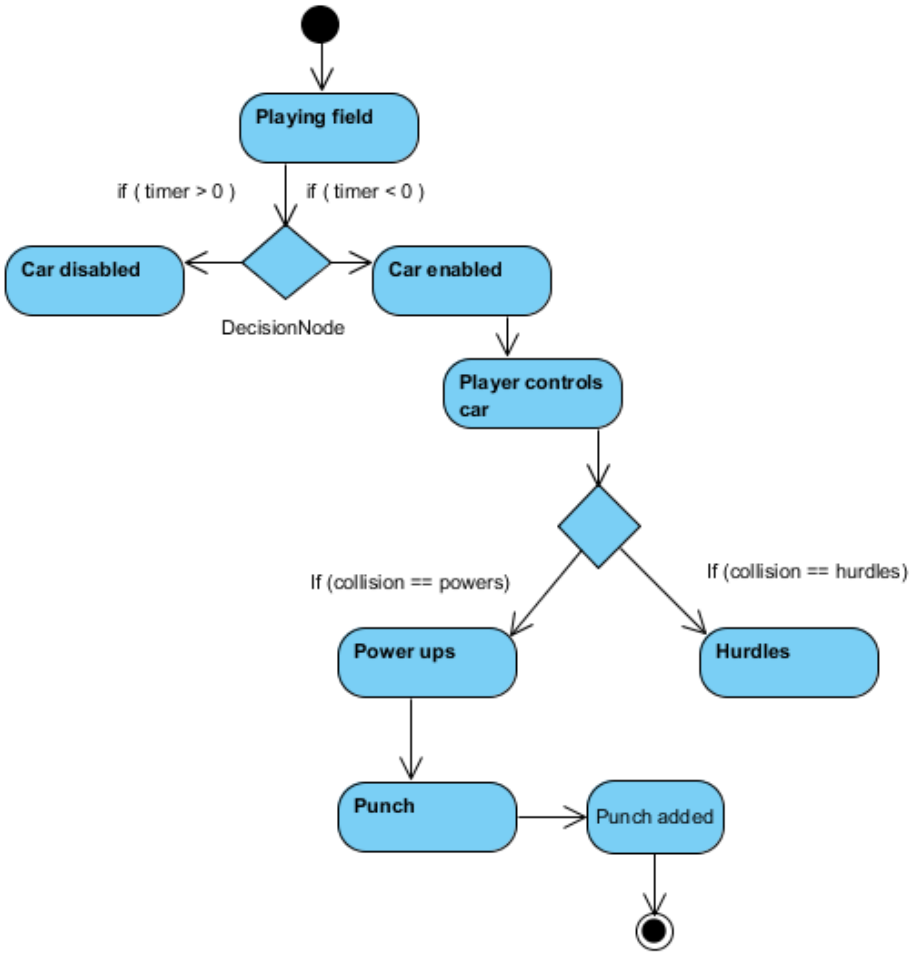

Figure 84: Activity Punch added

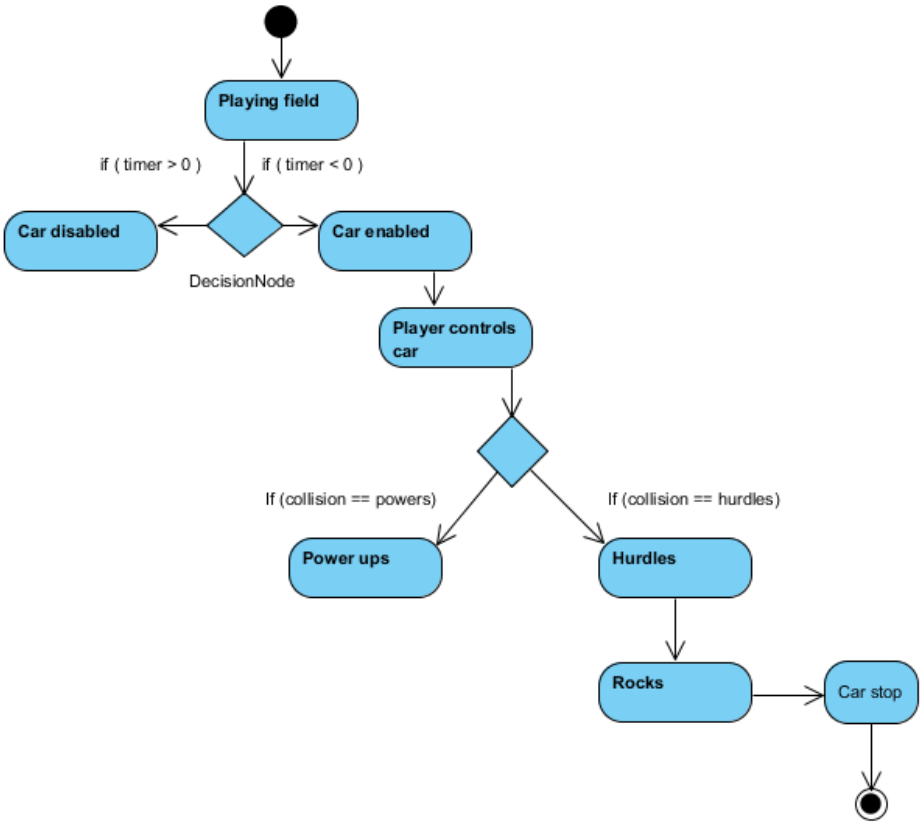

Figure 85: Activity Collision with rock

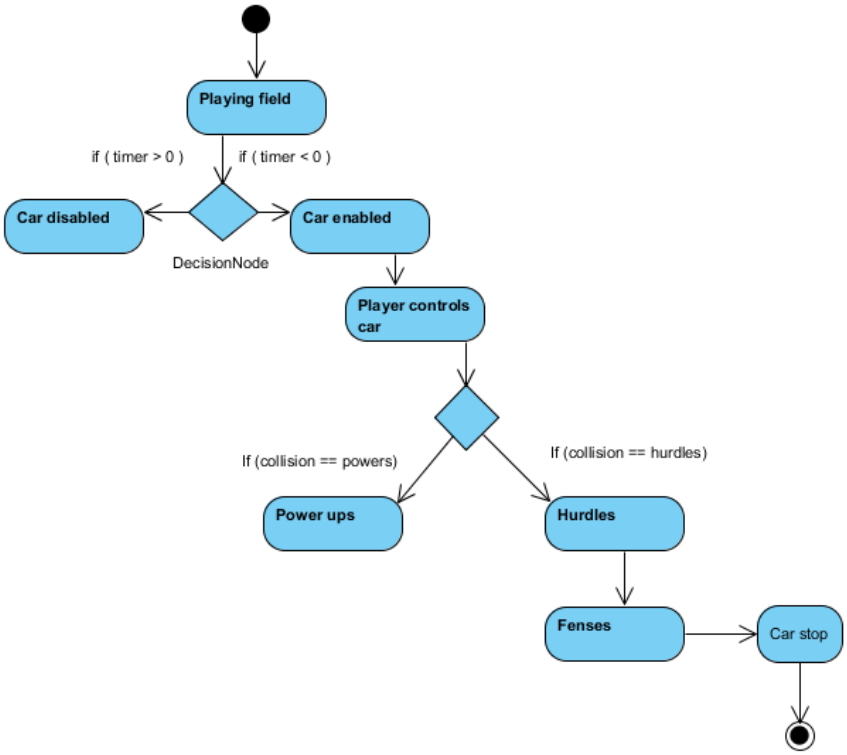

Figure 86: Activity collision with fence

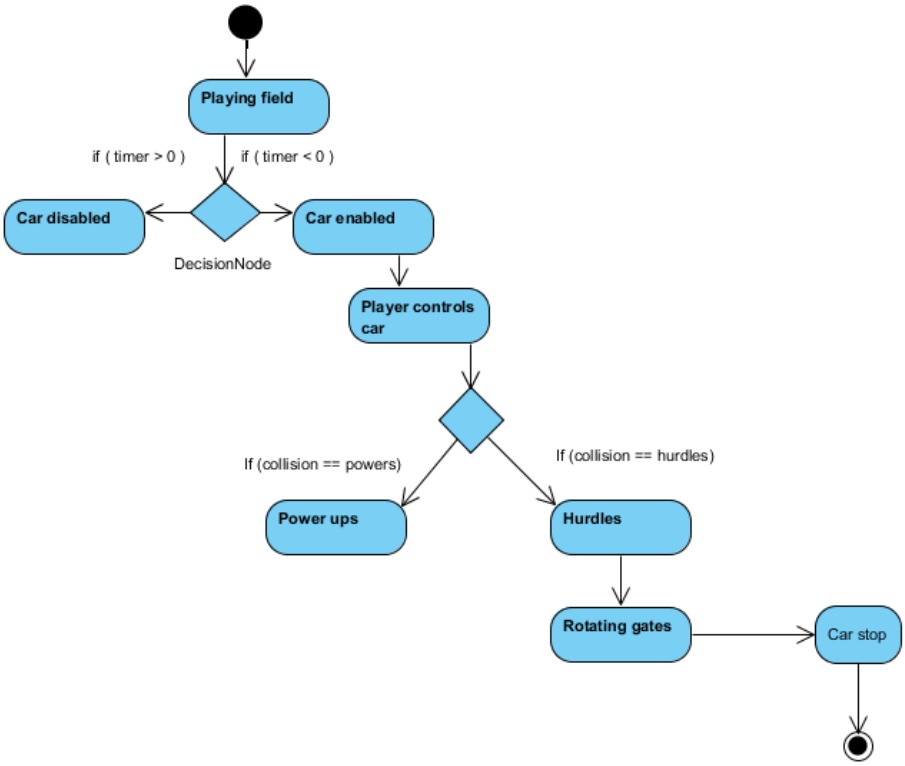

Figure 87: Activity collision with rotating gate

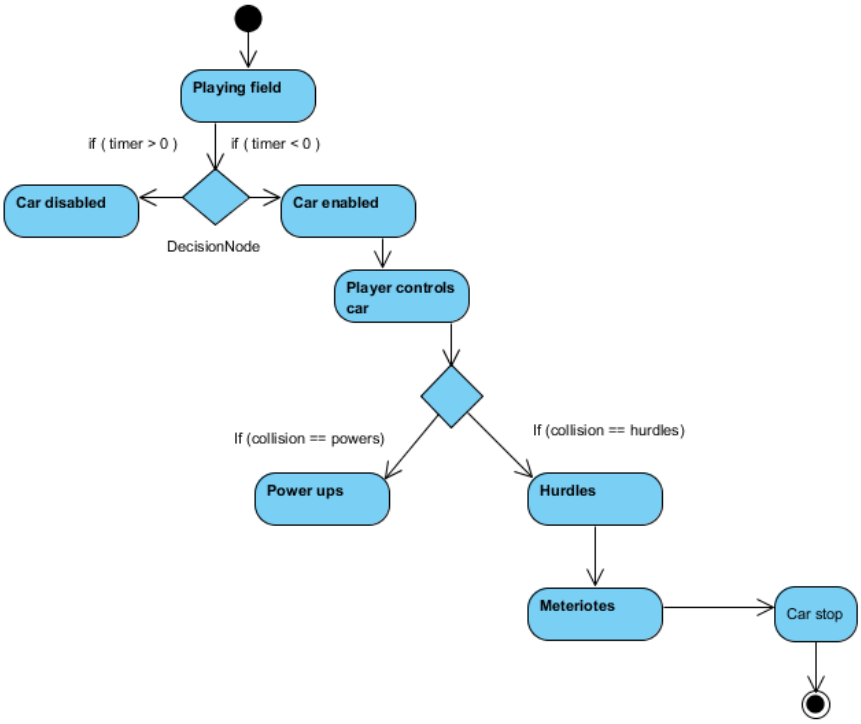

Figure 88: Activity collision with meteorites

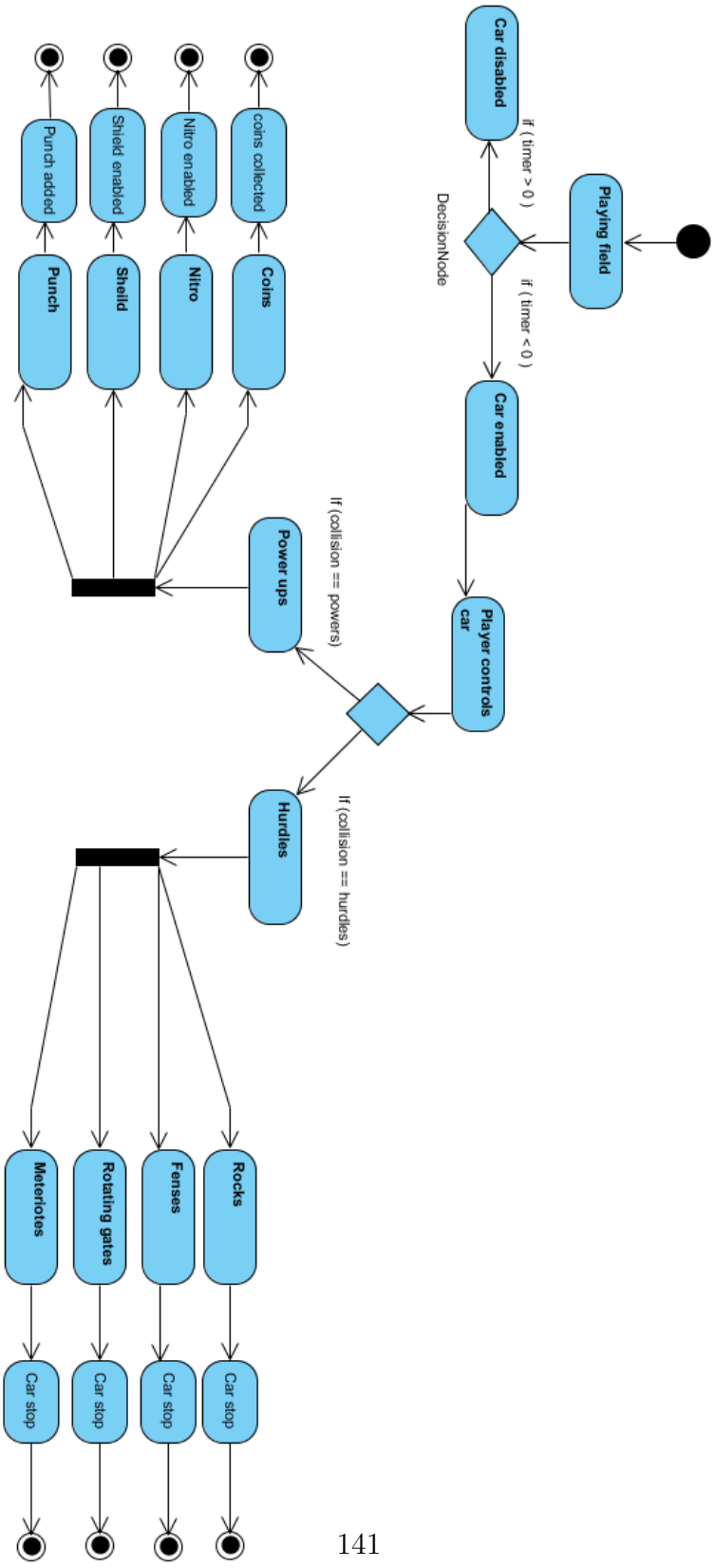

Figure 89: Activity Playing Field

7 Test Cases

7.1 PlayGame

Test case ID	**Play game**
Use case ID	1
Purpose	Game is starting properly
Environment	Unity 3d
Pre requirements	Click on game icon
Expected result	After clicking on play button game should start properly
Actual result	Game is started successfully

Figure 90: TestCase PlayGame

7.2 EasyMode

Test case ID	Start game in Easy mode
Use case reference	2
Purpose	User want to play game in easy mode
Environment	Unity 3d
Pre requirements	Main menu
Expected result	Run the game in easy mode
Actual result	Game is running correctly within easy mode

Figure 91: TestCase EasyMode

7.3 NormalMode

Test case ID	Normal mode
Use case reference	3
Purpose	User want to play game in normal mode
Environment	Unity 3d
Pre requirements	Main menu
Expected result	Run the game within normal mode
Actual result	Game is running correctly in normal mode

Figure 92: TestCase NormalMode

7.4 HardMode

Test case ID	Hard mode
Use case reference	4
Purpose	User want to play within hard mode
Environment	Unity 3d
Pre requirements	Main menu page
Expected result	Run the game within hard mode
Actual result	The game is running successfully in hard mode

Figure 93: TestCase HardMode

7.5 QuitGame

Test case ID	Quite the game and Save high score
Use case reference	5,6
Purpose	Save high score after existing game
Environment	Unity 3d
Pre requirements	Game is running
Expected result	Save the high score of user after leaving game
Actual result	Game is saving high score correctly after existing game

Figure 94: TestCase QuitGame

7.6 UpgradeEngine

Test case ID	Upgrade engine
Use case reference	7.1
Purpose	Upgrade car engine
Environment	Unity 3d
Pre requirements	Upgrade menu
Expected result	Successfully upgrade car engine
Actual result	Engine upgraded successfully

Figure 95: TestCase UpgradeEngine

7.7 UpgradeNitro

Test case ID	Upgrade nitro
Use case reference	7.2
Purpose	Upgrade Nitro
Environment	Unity 3ᵈ
Pre requirements	Upgrade menu page
Expected result	Successfully upgrade Nitro
Actual result	Nitro upgraded successfully

Figure 96: TestCase UpgradeNitro

7.8 UpgradeShield

Test case ID	Upgrade shield
Use case reference	7.3
Purpose	Upgrade Shield
Environment	Unity 3d
Pre requirements	Upgrade menu page
Expected result	Successfully upgrade shield
Actual result	Shield upgraded successfully

Figure 97: TestCase UpgradeShield

7.9 UpgradeBrakes

Test case ID	Upgrade Brakes
Use case reference	7.4
Purpose	Upgrade car brakes
Environment	Unity 3d
Pre requirements	Upgrade menu page
Expected result	Successfully upgrade car brakes
Actual result	Brakes upgraded successfully

Figure 98: TestCase UpgradeBrakes

7.10 ChangeCar

Test case ID	Change & purchase car
Use case reference	8
Purpose	Change current car, purchase new car or select purchased cars
Environment	Unity 3d
Pre requirements	Main menu
Expected result	Successfully change, purchase and selection of already purchased cars
Actual result	Cars change, purchase and selection of already purchased cars are successfully done.

Figure 99: TestCase ChangeCar

7.11 CreateRoom

Test case ID	Multi-player (creating new room)
Use case reference	9
Purpose	Creation of new room in a currently joined lobby
Environment	Unity 3d
Pre requirements	Multiplayer menu
Expected result	Room should be created successfully and waiting for players panel should be displayed.
Actual result	Room is created successfully and waiting for player's panel appeared correctly.

Figure 100: TestCase CreateRoom

7.12 JoinRoom

Test case ID	Multi-player (Joining existing room)
Use case reference	10
Purpose	Join the existing room created by another player
Environment	Unity 3d
Pre requirements	Multiplayer menu
Expected result	Existing room should be joined successfully.
Actual result	Existing room is joined and waiting for players panel appeared.

Figure 101: TestCase JoinRoom

7.13 MoveAICar

Test case ID	Moves toward finish line
Use case reference	14
Purpose	Cars should move towards finish line
Environment	Unity 3d
Pre requirements	Play field
Expected result	Cars should move toward finish line
Actual result	Cars are moving correctly towards finish line

Figure 102: TestCase AI car Moving

7.14 MoveLeft

Test case ID	Move left
Use case reference	15
Purpose	Car moves left correctly
Environment	Unity 3d
Pre requirements	Play field
Expected result	Car should move left
Actual result	Cars are moving in left side correctly

Figure 103: TestCase MoveLeft

7.15 MoveRight

Test case ID	Move right
Use case reference	16
Purpose	Car moves right correctly
Environment	Unity 3d
Pre requirements	Play field
Expected result	Car should move right
Actual result	Cars are moving in right side correctly

Figure 104: TestCase PlayGame

7.16 ApplyBrakes

Test case ID	Brake function
Use case reference	17
Purpose	Stop the car on applying brakes
Environment	Unity 3d
Pre requirements	Play field
Expected result	Car should stop by pressing brake button
Actual result	Car stops without any error by applying brakes

Figure 105: TestCase ApplyBrakes

7.17 ReverseCar

Test case ID	Reverse the car
Use case reference	19
Purpose	Car should reversed by pressing and hold brake button for 2 sec
Environment	Unity 3d
Pre requirements	Play field
Expected result	Car should reverse correctly
Actual result	Car is reversing without any error by pressing and hold brake button for 2 sec

Figure 106: TestCase ReverseCar

7.18 PickupShield

Test case ID	Shield pick up
Use case reference	20
Purpose	Car should pick up shield by colliding it
Environment	Unity 3d
Pre requirements	Play field
Expected result	Car should pick up shield
Actual result	Car picked up the shield successfully

Figure 107: TestCase PickupShield

7.19 PickupNitro

Test case ID	Nitro pick up
Use case reference	21
Purpose	Car should pick up nitro power by colliding it
Environment	Unity 3d
Pre requirements	In game menu
Expected result	Car should pick up nitro power
Actual result	Car picked up the nitro successfully

Figure 108: TestCase PickupNitro

7.20 PunchPickup

Test case ID	Punch pick up
Use case reference	22
Purpose	Car should pick up punch by colliding it
Environment	Unity 3d
Pre requirements	In game menu
Expected result	Car should pick up punch
Actual result	Car picked up the punch successfully

Figure 109: TestCase PickupPunch

7.21 CollectCoins

Test case ID	Collecting Coins
Use case reference	23
Purpose	Car should pick up coins by colliding it
Environment	Unity 3d
Pre requirements	In game menu
Expected result	Car should pick up coins
Actual result	Car picked up the coins and incremented in coins collection successfully

Figure 110: TestCase CollectCoins

7.22 CollisionWithRocks

Test case ID	Collision with rocks
Use case reference	24
Purpose	Car should stop after colliding rocks
Environment	Unity 3d
Pre requirements	Play Field
Expected result	Car should stop after collision with rocks
Actual result	Car stops correctly after colliding with rocks

Figure 111: TestCase CollisionWithRocks

7.23 CollisionWithMeteorites

Test case ID	Collision with meteorites
Use case reference	25
Purpose	Car should stop after colliding with meteorites
Environment	Unity 3d
Pre requirements	Play Field
Expected result	Car should stop after collision with meteorites
Actual result	Car stops correctly after colliding with meteorites

Figure 112: TestCase CollisionWithMeteorites

7.24 CollisionWithFences

Test case ID	Collision with wall fences
Use case reference	26
Purpose	Car should stop after colliding wall fences
Environment	Unity 3d
Pre requirements	Play field
Expected result	Car should stop after collision with wall fences
Actual result	Car stops correctly after colliding wall fences

Figure 113: TestCase CollisionWithFences

7.25 AICarObstacleDetectionMoveLeft

Test case ID	Detection obstacles (For Rival cars)
Use case reference	27
Purpose	Rival car should move left if obstacles are in front right
Environment	Unity 3d
Pre requirements	Play field
Expected result	Rival cars should move left side if cars detect obstacles in front right
Actual result	Rival cars are detecting obstacles without any error

Figure 114: TestCase AICarObstacleDetectionMoveLeft

7.26 AICarObstacleDetectionMoveRight

Test case ID	Detection obstacles (For Rival cars)
Use case reference	28
Purpose	Rival car should move right if they found obstacles in front left
Environment	Unity 3d
Pre requirements	Play field
Expected result	Rival cars should move right side if cars detect obstacles in front left
Actual result	Rival cars are detecting obstacles without any error

Figure 115: TestCase AICarObstacleDetectionMoveRight

7.27 AICarObstacleDetection

Test case ID	Detection obstacles (For Rival cars)
Use case reference	29
Purpose	Rival car should move left or right if obstacles are in front center
Environment	Unity 3d
Pre requirements	Play field
Expected result	Rival cars should move left or right side if cars detect obstacles in front center
Actual result	Rival cars are detecting obstacles without any error

Figure 116: TestCase AICarObstacleDetection

7.28 RaceFinish

Test case ID	Save high score and coins
Use case reference	32
Purpose	After finishing the race game should save score and currency
Environment	Unity 3d
Pre requirements	Play field
Expected result	After finishing the race game should save score and collected currency
Actual result	Collected coins and high score saved successfully

Figure 117: TestCase RaceFinish

8 Conclusion and Future Work

This game took several ideas from previously developed games and few movies and successfully integrated them, there is still a lot of work that can be done more power-ups and obstacles could be added. Cars could be mounted by machine guns and rocket launchers. Different types of vehicles could be integrated, separate motorbike race levels could be implemented. Separate water boat race levels could also be implemented.

References

[1] Csar Villacs, Walter Fuertes, Andrs Bustamante, Daniel Almachi, Carlos Procel, Susana Fuertes, and Theofilos Toulkeridis. *Multi-player Educational Video Game over Cloud to Stimulate Logical Reasoning of Children*. 2014 IEEE/ACM 18th International Symposium.

[2] Anderson, Joshua. *Innovative navigation artificial intelligence for motor racing games*. 2016, University of Bedfordshire.

[3] Dayan, Shangguan, and Huang Xinyuan. *Development of Unity3D Network Functionality on Android*. International Journal of Hybrid Information Technology 9.2 (2016): 359-370.

[4] Dhuri, Shroneet, et al. *Game Development for Android Device using Unity 3D*.

[5] Lee, Byungjoo, Yunsil Heo, and Hyunwoo Bang. *Racetime: Telepresence Racing Game with Multi-user Participation*. Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play. ACM, 2015.

[6] Chan, Marvin T., Christine W. Chan, and Craig Gelowitz. *Development of a car racing simulator game using artificial intelligence techniques*. International Journal of Computer Games Technology 2015 (2015): 12.