# CS 491 Senior Design
# Analysis Report


## DriveSim

**Group Members**

Bora Ecer

Oğuz Kaan Ağaç

Alp Ege Baştürk

Alptuğ Albayrak

Alperen Erkek

**Supervisor:**

Prof. Uğur Güdükbay

**Jury:**

Prof. İbrahim Körpeoğlu

Prof. Özgür Ulusoy

**Innovation Expert:**

Cem Çimenbiçer (TaleWorlds Entertainment)

**November 12, 2018**

# 1. Introduction

Our project will be a machine learning based simulation game where player will design a vehicle and place sensors that will feed the environmental data to our training system. Depending on vehicle design, placement and type of sensors our training system will teach itself how to drive without crashing into walls/pedestrians/road blocking objects. With the trained vehicle, user will be able to play one of the four game mod. The game mods will allow the user to see how well the designed car works. Depending on the score the player gets from the game mods, the player can evaluate the design of the car, and change the design if he/she chooses to do so.

This report will provide detailed information about the DriveSim. We will provide other similar titles and what differs our game from them. Also, we will explain the system and its design through some system models and provide some possible scenarios that we anticipate to be encountered by the players. Finally, we will provide some mockups of the game interface in order to give a quick look to the game. However, these mockups are just representations of what the final game will look like so in the final product the game interface will be improved and will have better quality.

# 2. Current System

Neural networks and genetic algorithms have been applied to various games. There are car design games such as Dream Car Builder where players design their cars and join the races against other players Dream Car Builder is similar to our system in terms of vehicle design feature [1]. In addition, there are mix of designer and tycoon games like Cars Incorporated where players design vehicles, develop technologies for improved parts to compete in races and sell their designs in game.

Cars Incorporated is similar in terms of design feature and race feature. Race simulation and the business part of this game differs from our system [2].

# 3. Proposed System

Technical details of the DriveSim will be explained in this section. More detaily there will be qualifying specifications to each component of the system and system models will be provided. These models will clarify limitations of the user as well as our game. In-game mockups and screenshots will be used in order to explain these. However they may change if better design has been found.

## 3.1 Overview

The goal of DriveSim is to make players to think creatively and helps them to understand how a car can see and understand its environment. They will be in charge of creating their car and putting necessary tools to detect its surrounding. This means that wrong combination of these can result in failure. For this reason they need to find a good balance while creating these. After this, they will see how a neural network can learn to drive using the car they created. There will be different kind of variables such as time, points etc that will detect how good the design was. This will create a competition between each car and force them to think in a better way.

Game mechanics will be simple in terms of user perspective. User simply needs to create a car using DriveSim's editor. In this editor there will be different kind of materials, engines, sensors and accelerometer. Using these they can create a car however they want. Although final version of the DriveSim will have bugs and glitches as minimum as it can have, remaining bugs and glitches will be fixed through game updates.

## 3.2 Requirements

The requirements needs to be cleared in early stage [3]. Because it is more difficult to change the structure of the system afterwards and it may introduce additional bugs. However, it not possible to finalize every requirement and some requirement can change during the implementation. A successful project needs to adapt these changes [4]. In this section DriveSim's requirements will be explained.

### 3.2.1 Functional Requirements

- The player will be able to design vehicles using some basic building blocks.
- The player will be able to select sensors with different specifications such as distance, angle of vision, etc. or different sensors such as accelerometer sensors and etc.
- The player will be able to add sensors to the vehicles. These sensors will be used to guide the vehicle.
- The player will be able deploy the designed vehicle to the training phase.
- The player will be able to preview the training phase of the vehicle with simulation mode.
- The player will be able to skip to a specific generation of the training phase.
- The player will be able to save and load a trained vehicle design.
- The player will be able to select other game modes and play them with the trained vehicle.
- The player will be able to pause/cancel the game modes.
- The player will be able to restart the current game mode.
- The player will be able to configure sound options from options panel.
- The player will not be able to control the vehicles.

### 3.2.2 Nonfunctional Requirements

- The game interface should be user-friendly.
- Vehicle design mode should be user-friendly and easy to understand.
- Training phase should be completed in a limited time.
- Vehicle movement physics should be realistic.
- Sensor placement should affect the success of the vehicle.
- The game should be bug free.

### 3.2.3 Pseudo Requirements

- The should be developed with Unreal Engine 4 or higher.
- The game should not require active internet connection.

- The game may require GPU to train the network.
- The language should be in English.

## 3.3 System Models

### 3.3.1 Scenarios

Scenarios define use of the system. In this section, main scenarios which user can use will be defined.

**Scenario 1:**

The user is in the game mode  and clicks the “Designs” button. The design menu loads and the user can look at the previous designs that are saved on the computer. Then the user returns to the main menu and quits the game.

**Use Case Name:** Viewing the previous designs

**Actors:** User

**Entry Conditions:**

- User opens the game and game loads successfully

**Exit Conditions:**

- User clicks the “Exit” button

**Main Flow of Events:**

1. User opens the game
2. Main screen loads
3. User clicks the “Design” button
4. Design menu loads
5. User returns to main menu
6. User exits the game by clicking the “Exit” button

**Scenario 2:**

The user opens the game and clicks the “Play” button. Gameplay menu opens. The user selects the game mode and a level from the selected game mode. Then the user opens the designs menu and selects a previously created design. After selecting the design the user returns to the gameplay menu and starts the level. The level loads and training starts.

After the completion of the training, the game shows the results screen and the user decides to quit without saving.

**Use Case Name:** Training an Existing Design but not Saving

**Actors:** User

**Entry Conditions:**

- User opens the game and game loads successfully
- User selects a game mode
- User selects a level for the game mode
- User selects a design

**Exit Conditions:**

- The user clicks “Exit Without Saving” button after the training

**Main Flow of Events:**

1. User opens the game
2. Main screen loads
3. User clicks the “Play” button
4. Game play menu loads
5. User selects a game mode
6. User selects a level for the game mode
7. User opens the design menu
8. User selects a previously created design
9. User continues with the training
10. After the completion of the training the results are shown to the user
11. User clicks “Exit Without Saving” and exits to the main menu

**Scenario 3:**

The user opens the game and clicks the “Play” button. Gameplay menu opens. The user selects the game mode and a level from the selected game mode. Then the user opens the designs menu and clicks “Create”. After completing the new design the user saves it

and selects it. The level loads and training starts. After the completion of the training, the game shows the results screen and the user selects a specific generation to watch. After watching the generation, the user decides to update the design and clicks the “Edit Design” button. After editing the design the training starts again. After the completion of the training, the game shows the results screen once again. This time the user clicks “Save and Exit” button and saves the current network with the design.

**Use Case Name:** Training a Newly Created Design

**Actors:** User

**Entry Conditions:**

- User opens the game and game loads successfully
- User selects a game mode
- User selects a level for the game mode
- User selects a design

**Exit Conditions:**

- The user clicks “Save and Exit” button after the training

**Main Flow of Events:**

1. User opens the game
2. Main screen loads
3. User clicks the “Play” button
4. Game play menu loads
5. User selects a game mode
6. User selects a level for the game mode
7. User opens the design menu
8. User clicks “Create”
9. Design editor opens
10. User creates a new design
11. User saves the new design

12. User selects this newly created design
13. User continues with the training
14. After the completion of the training the results are shown to the user
15. User selects a specific generation to watch
16. The game renders this generation's performance to user
17. The game returns to the result screen
18. User clicks "Edit Design" button
19. Editor opens
20. User makes some changes in the design
21. The editor closes and training starts again
22. After the completion of the training the results are shown to the user
23. User clicks "Save and Exit" button and exits to the main menu

**Scenario 4:**

The user opens the game and clicks the "Options" button. In the options menu the user turns off the music and returns to the main menu.

**Use Case Name:** Turning Off the Music

**Actors:** User

**Entry Conditions:**

- User opens the game and game loads successfully

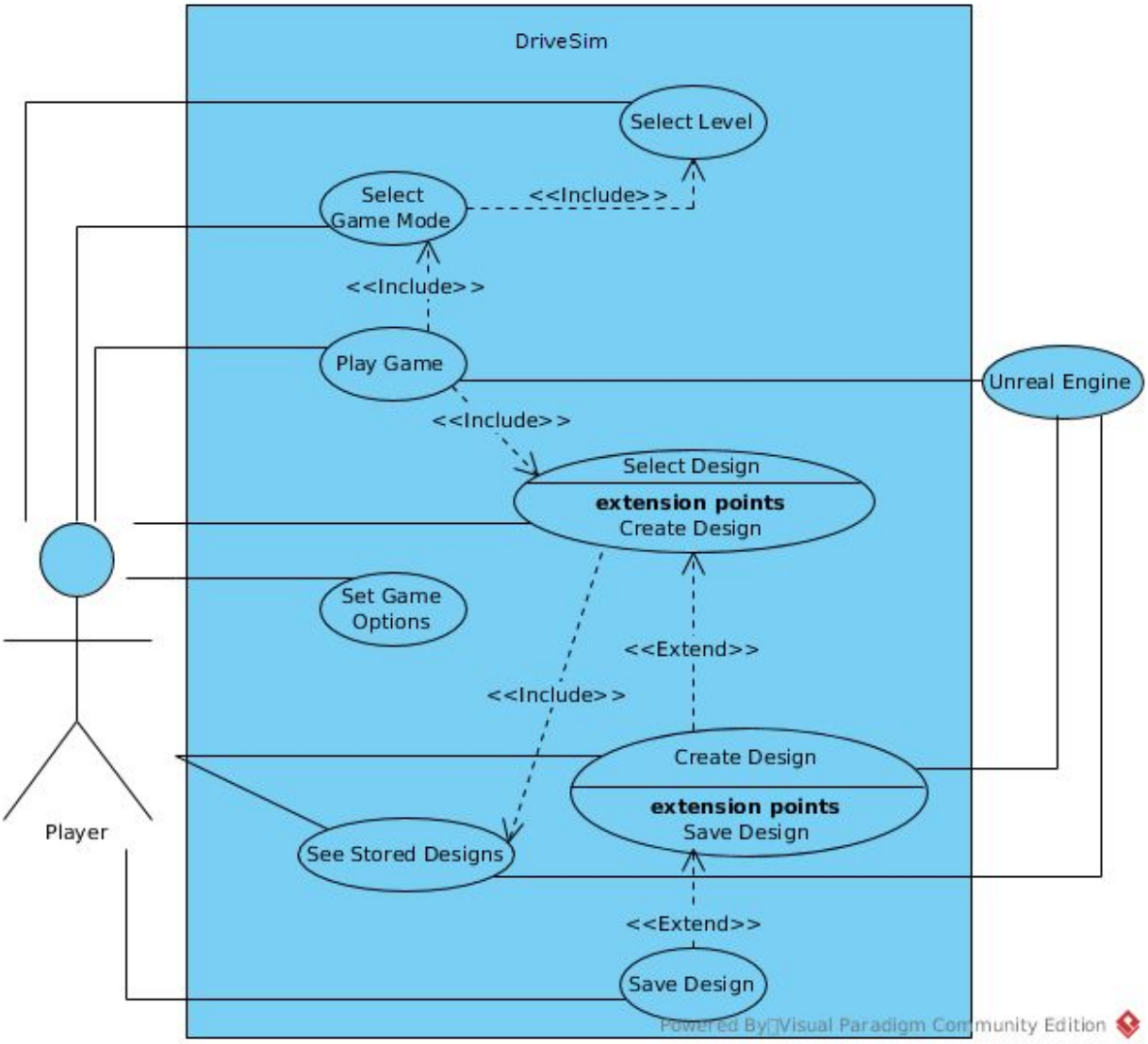
**Exit Conditions:**

- The user clicks "Return" button from options menu and returns to the main menu

**Main Flow of Events:**

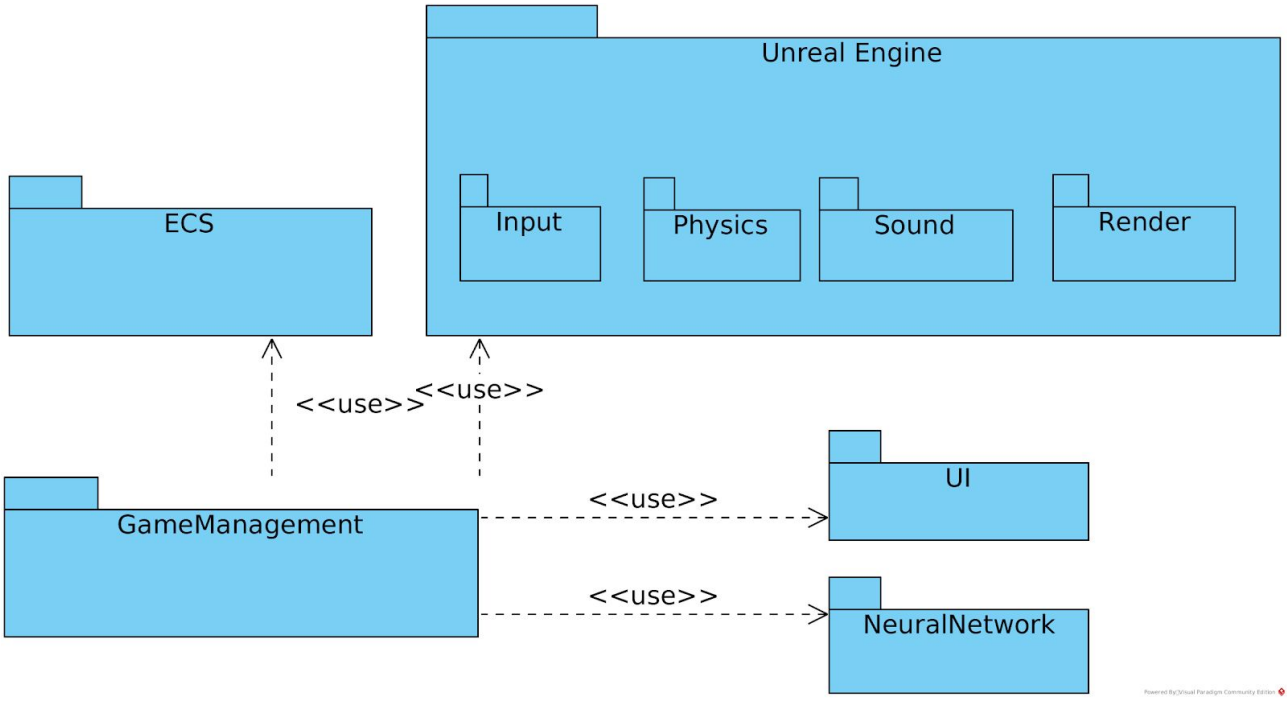
1. User opens the game
2. Main screen loads
3. User clicks the "Options" button
4. Options menu loads
5. User clicks the "Music" toggle and turns off the music
6. User clicks the "Return" button and returns to the main menu

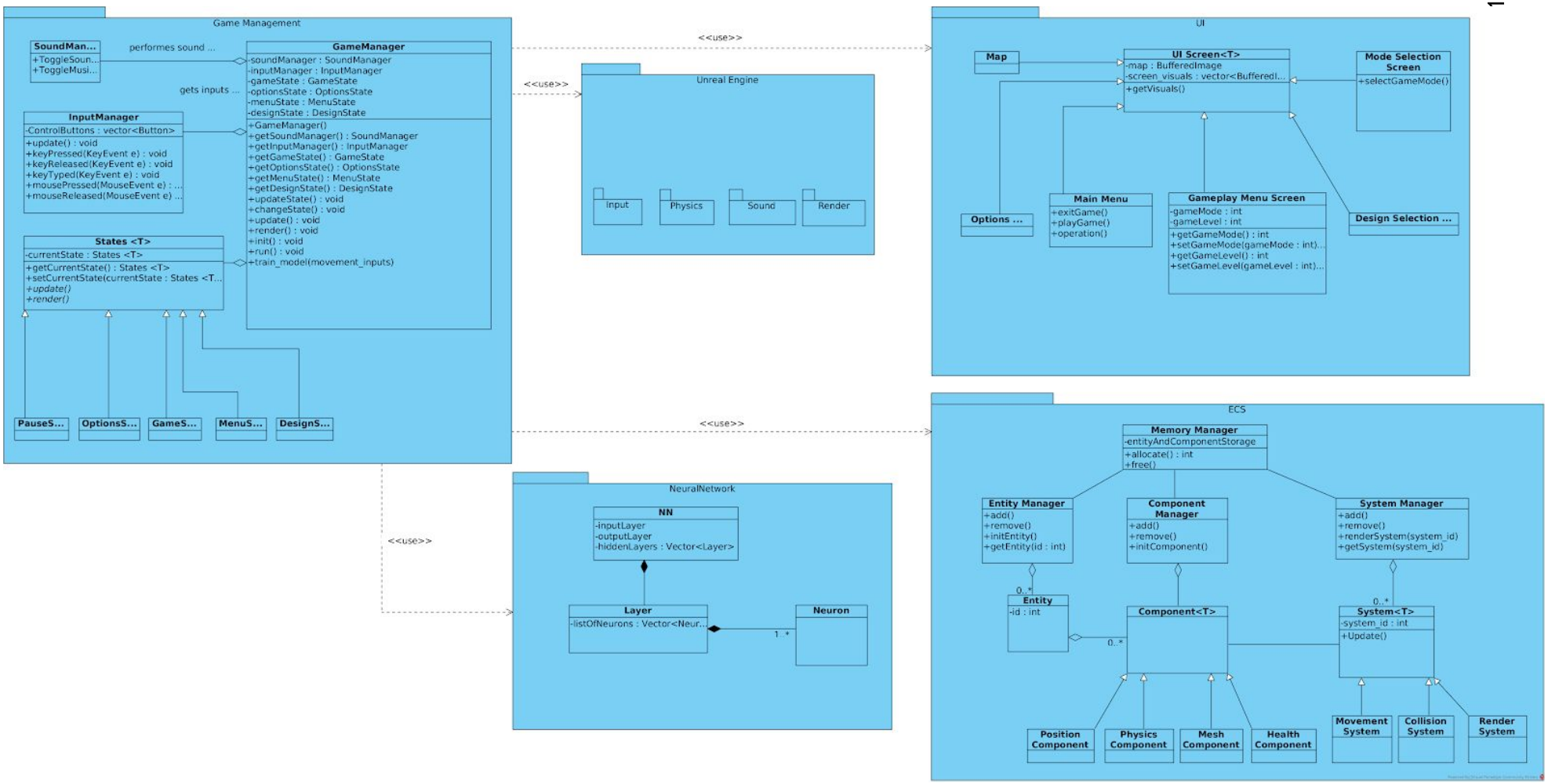
### 3.3.2 Use Case Model

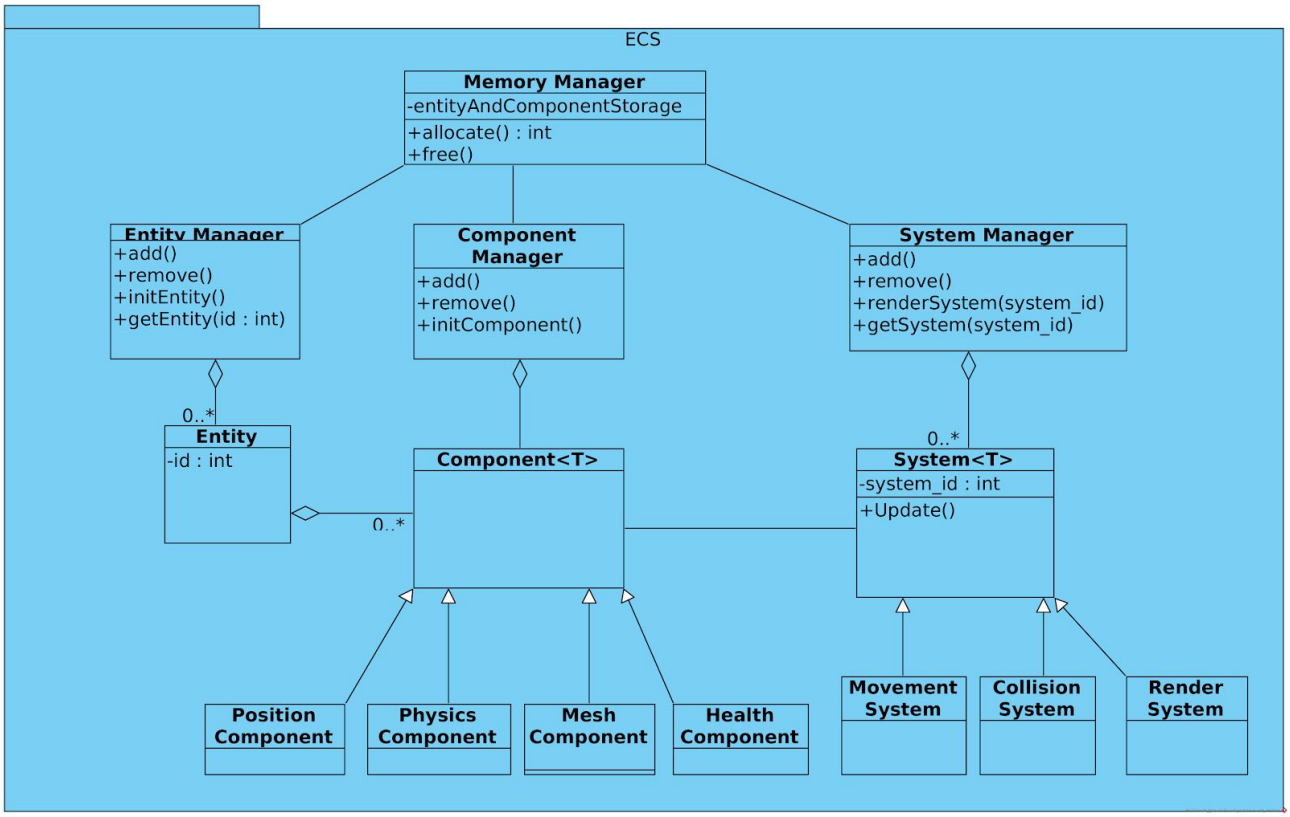


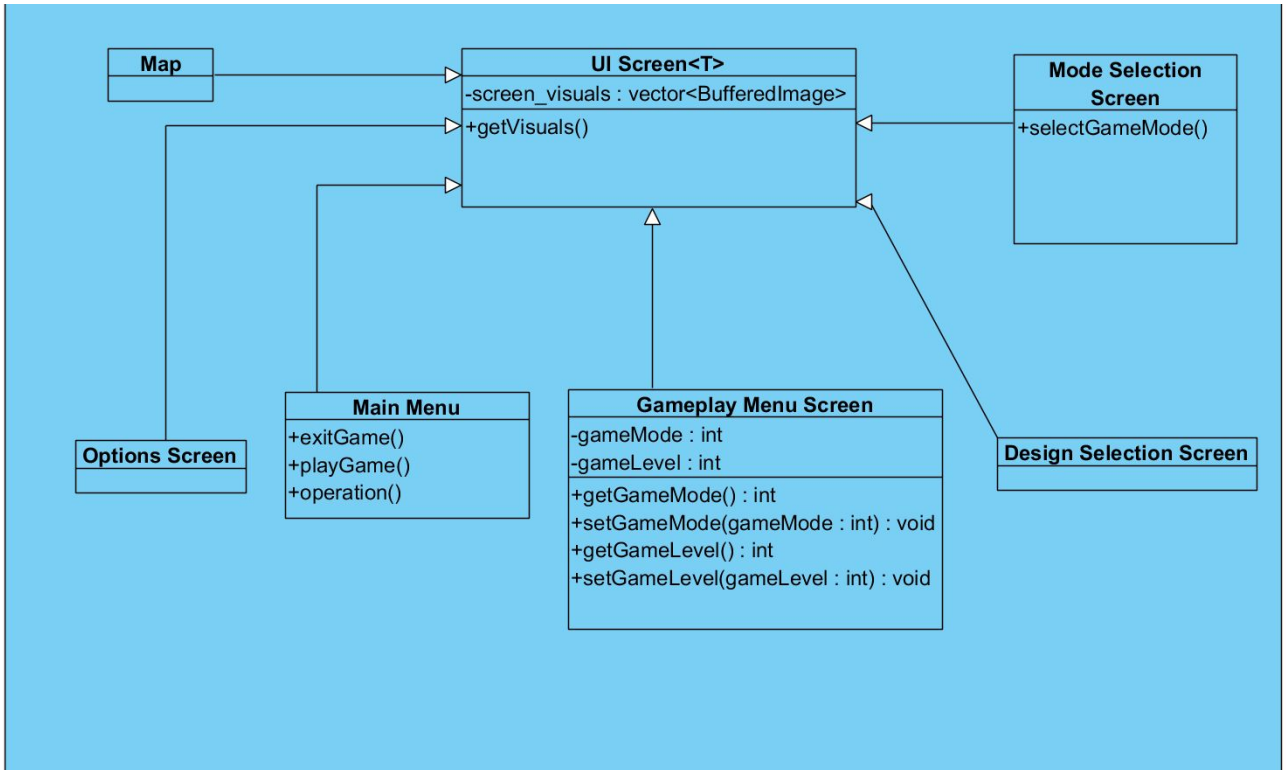
### 3.3.3 Object and Class Model

The following diagram is the general overview of the system. The object model is divided into five parts which are Game Management, Entity Component System, User-Interface, Neural Network and Unreal Engine. The Entity Component System will follow the Entity-Component design pattern and it will be used for the game objects. Entity is a general purpose object, component is the parameters for one facet of the object, and the system is something that runs continuously and performs actions on every entity that has a component of the same facet. It is expected that this design pattern will make development and maintenance easier and improve the performance of the system [5].




Game Management
SoundMan...
+ToggleSoun...
+ToggleMusi...
performes sound ...
gets inputs ...
GameManager
-soundManager : SoundManager
-inputManager : InputManager
-gameState : GameState
-optionsState : OptionsState
-menuState : MenuState
-designState : DesignState
+GameManager()
+getSoundManager() : SoundManager
+getInputManager() : InputManager
+getGameState() : GameState
+getOptionsState() : OptionsState
+getMenuState() : MenuState
+getDesignState() : DesignState
+updateState() : void
+changeState() : void
+update() : void
+render() : void
+init() : void
+run() : void
+train_model(movement_inputs)
InputManager
-ControlButtons : vector<Button>
+update() : void
+keyPressed(KeyEvent e) : void
+keyReleased(KeyEvent e) : void
+keyTyped(KeyEvent e) : void
+mousePressed(MouseEvent e) : ...
+mouseReleased(MouseEvent e) ...
States <T>
-currentState : States <T>
+getCurrentState() : States <T>
+setCurrentState(currentState : States <T>...
+update()
+render()
PauseS...
OptionsS...
GameS...
MenuS...
DesignS...
<<use>>
Unreal Engine
Input
Physics
Sound
Render
UI
Map
UI Screen<T>
-map : BufferedImage
-screen_visuals : vector<BufferedI...
+getVisuals()
Mode Selection Screen
+selectGameMode()
Options ...
Main Menu
+exitGame()
+playGame()
+operation()
Gameplay Menu Screen
-gameMode : int
-gameLevel : int
+getGameMode() : int
+setGameMode(gameMode : int)...
+getGameLevel() : int
+setGameLevel(gameLevel : int)...
Design Selection ...
NeuralNetwork
NN
-inputLayer
-outputLayer
-hiddenLayers : Vector<Layer>
Layer
-listOfNeurons : Vector<Neur...
Neuron
1..*
ECS
Memory Manager
-entityAndComponentStorage
+allocate() : int
+free()
Entity Manager
+add()
+remove()
+initEntity()
+getEntity(id : int)
Component Manager
+add()
+remove()
+initComponent()
System Manager
+add()
+remove()
+renderSystem(system_id)
+getSystem(system_id)
0..*
Entity
-id : int
Component<T>
System<T>
-system_id : int
+Update()
Position Component
Physics Component
Mesh Component
Health Component
Movement System
Collision System
Render System


ECS
Memory Manager
-entityAndComponentStorage
+allocate() : int
+free()
Entity Manager
+add()
+remove()
+initEntity()
+getEntity(id : int)
Component Manager
+add()
+remove()
+initComponent()
System Manager
+add()
+remove()
+renderSystem(system_id)
+getSystem(system_id)
0..*
Entity
-id : int
Component<T>
0..*
0..*
System<T>
-system_id : int
+Update()
Position Component
Physics Component
Mesh Component
Health Component
Movement System
Collision System
Render System


Map
UI Screen<T>
-screen_visuals : vector<BufferedImage>
+getVisuals()
Mode Selection Screen
+selectGameMode()
Options Screen
Main Menu
+exitGame()
+playGame()
+operation()
Gameplay Menu Screen
-gameMode : int
-gameLevel : int
+getGameMode() : int
+setGameMode(gameMode : int) : void
+getGameLevel() : int
+setGameLevel(gameLevel : int) : void
Design Selection Screen

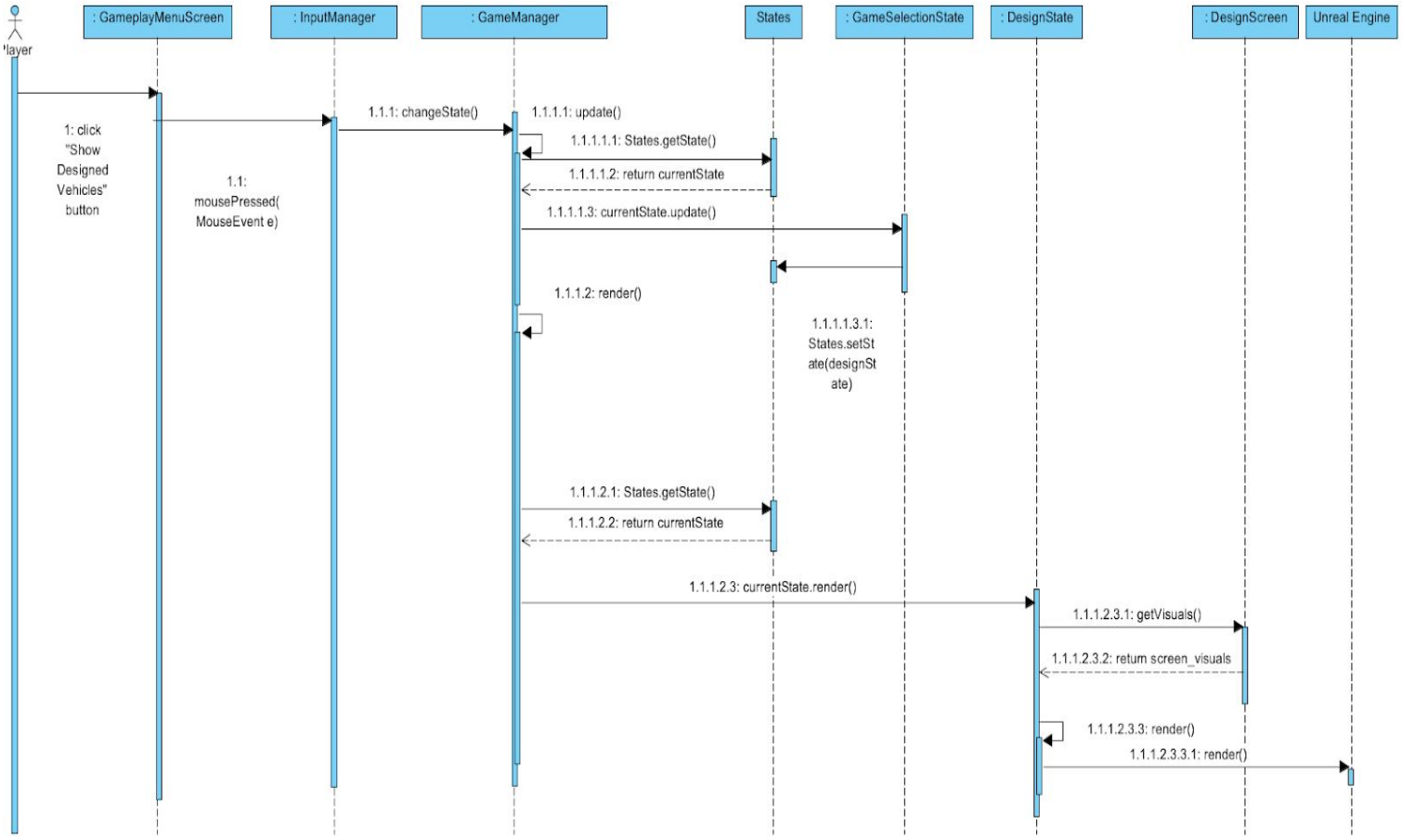
### 3.3.4 Dynamic Models

The following diagrams describe the dynamic behavior of the components of the system. In order to explain the dynamic workflow of our system with the respect to the player interactions we have used sequence diagrams and activity diagram.

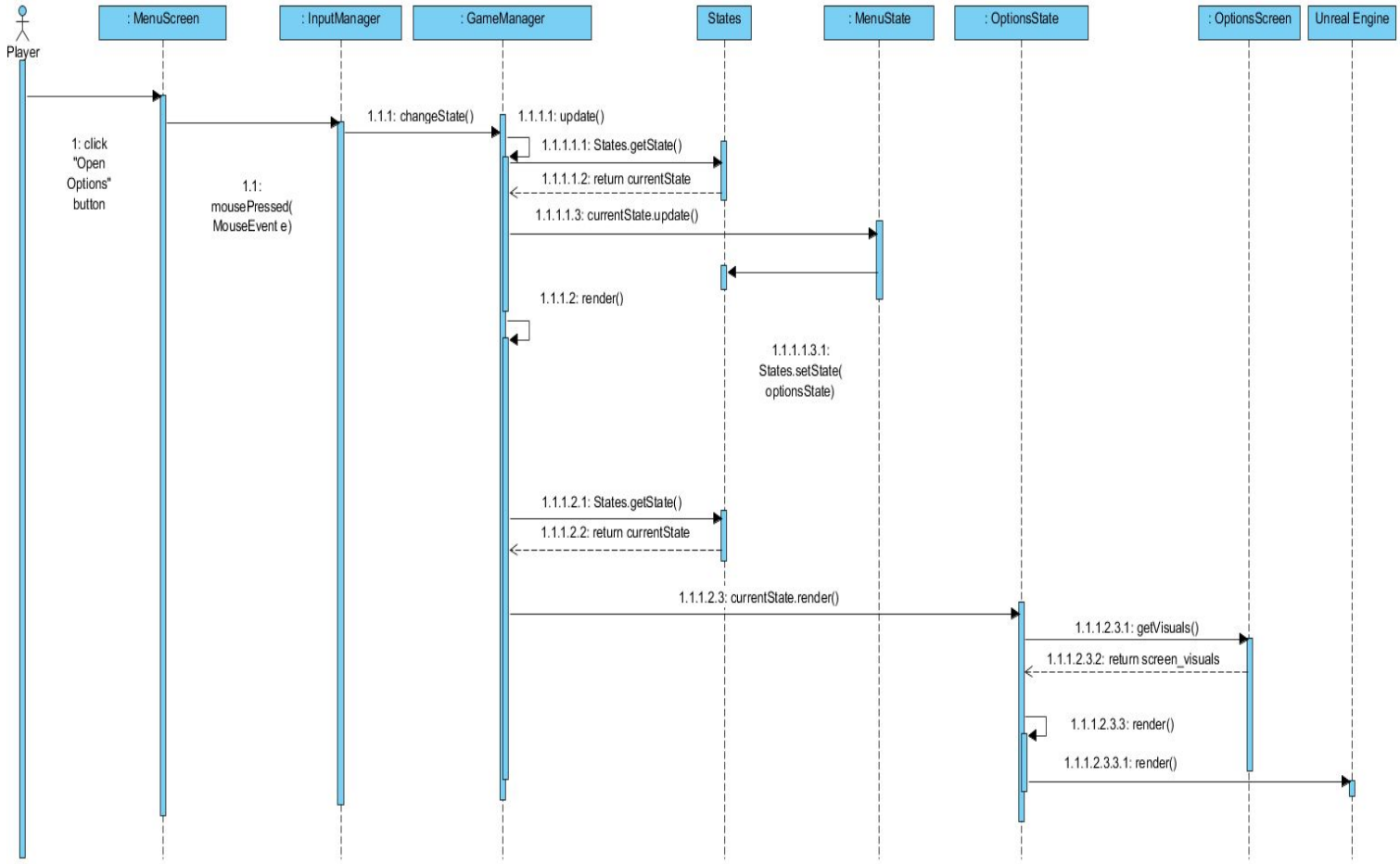
#### 3.3.4.1 Sequence Diagrams

The sequence diagrams in this section represents the parts of the activities given in the scenarios 1, 3 and 4. Unfortunately most of the given scenarios cannot be covered with the sequence diagrams, therefore we only covered the part of them.

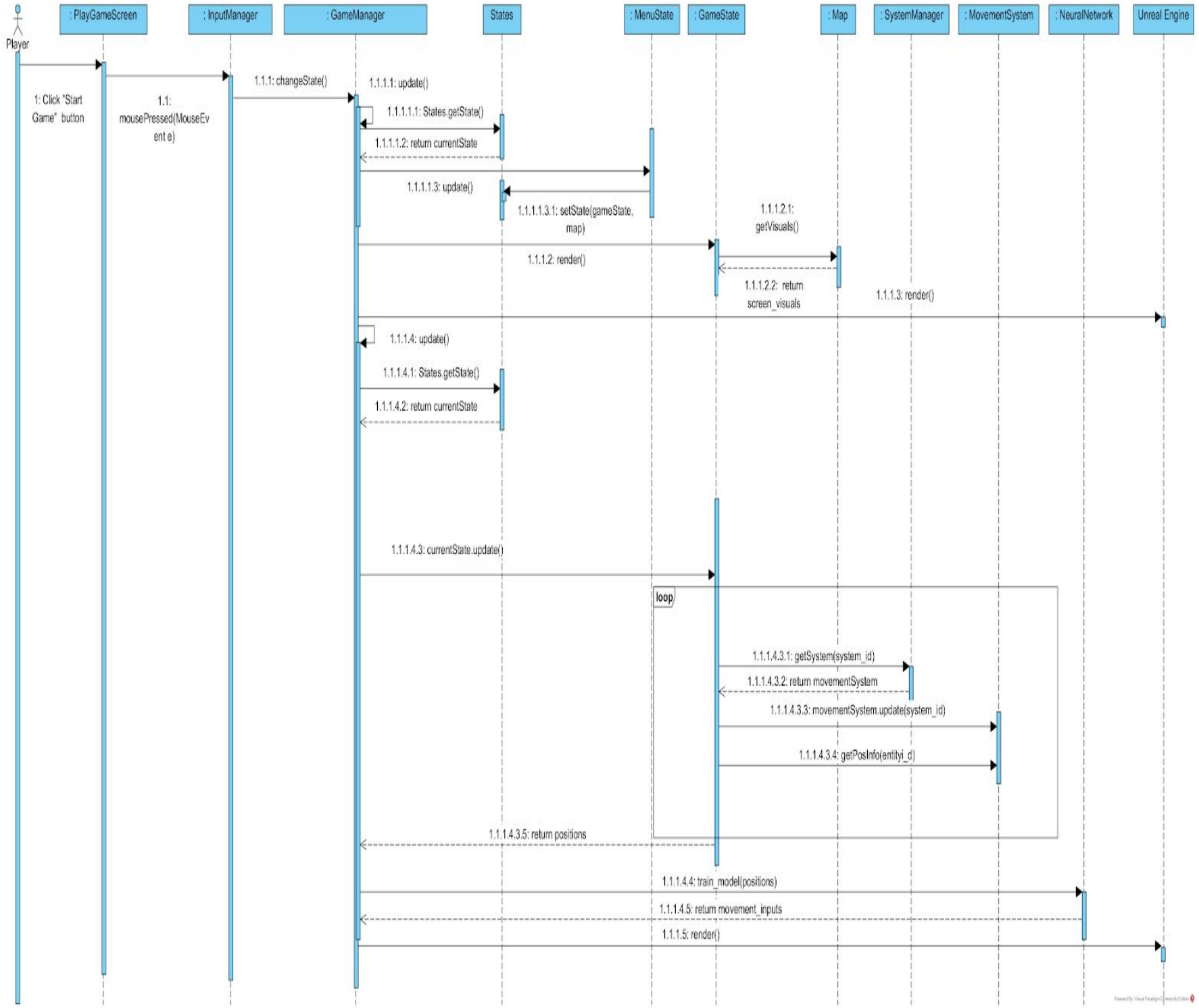
##### 3.3.4.1.1 Open Options Sequence Diagram



### 3.3.4.1.2 Show Designed Vehicles Sequence Diagram

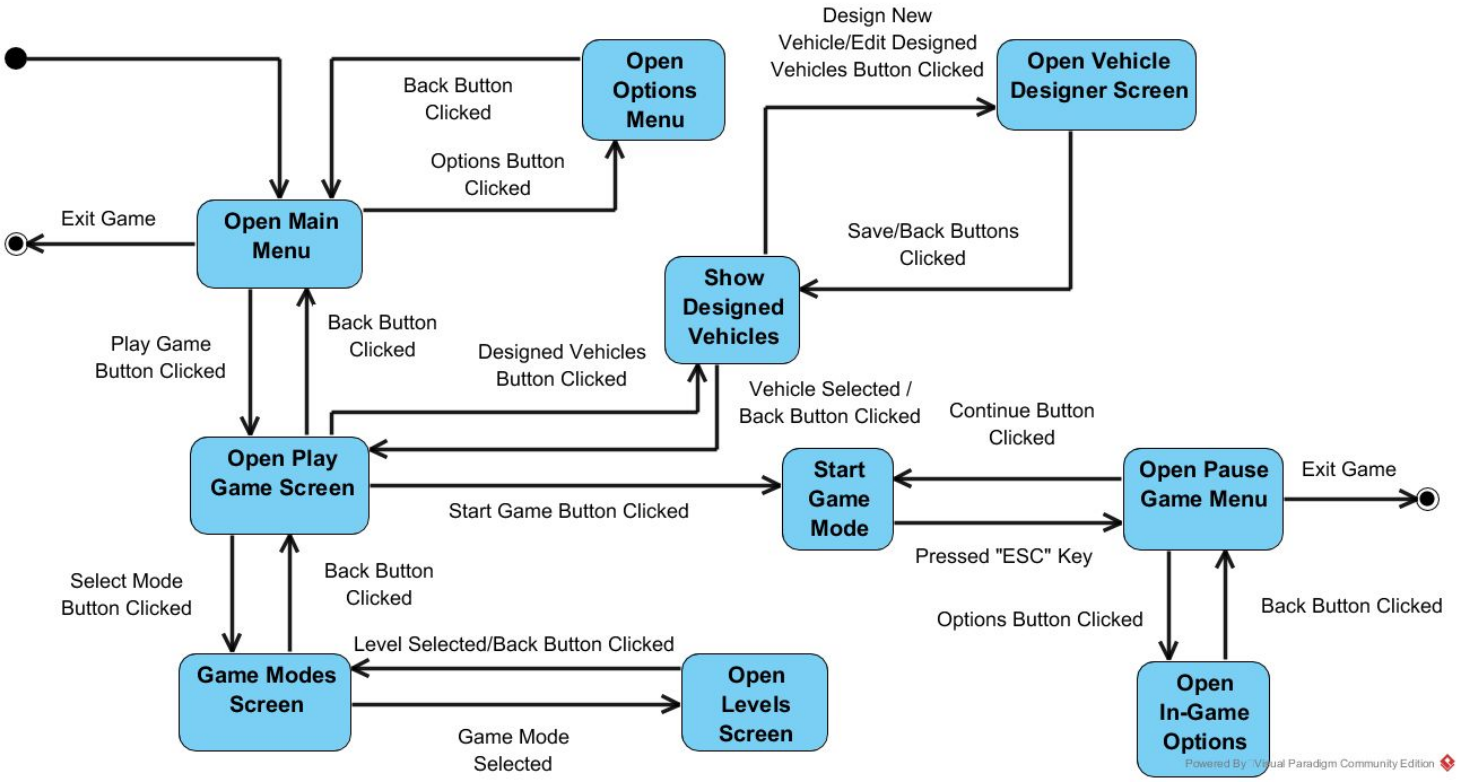



### 3.3.4.1.3 Start Game Sequence Diagram


Player
: PlayGameScreen
: InputManager
: GameManager
States
: MenuState
: GameState
: Map
: SystemManager
: MovementSystem
: NeuralNetwork
Unreal Engine
1: Click "Start Game" button
1.1: mousePressed(MouseEvent e)
1.1.1: changeState()
1.1.1.1: update()
1.1.1.1.1: States.getState()
1.1.1.1.2: return currentState
1.1.1.1.3: update()
1.1.1.1.3.1: setState(gameState, map)
1.1.1.2.1: getVisuals()
1.1.1.2: render()
1.1.1.2.2: return screen_visuals
1.1.1.3: render()
1.1.1.4: update()
1.1.1.4.1: States.getState()
1.1.1.4.2: return currentState
1.1.1.4.3: currentState.update()
loop
1.1.1.4.3.1: getSystem(system_id)
1.1.1.4.3.2: return movementSystem
1.1.1.4.3.3: movementSystem.update(system_id)
1.1.1.4.3.4: getPosInfo(entityi_d)
1.1.1.4.3.5: return positions
1.1.1.4.4: train_model(positions)
1.1.1.4.5: return movement_inputs
1.1.1.5: render()

### 3.3.4.2 Activity Diagram

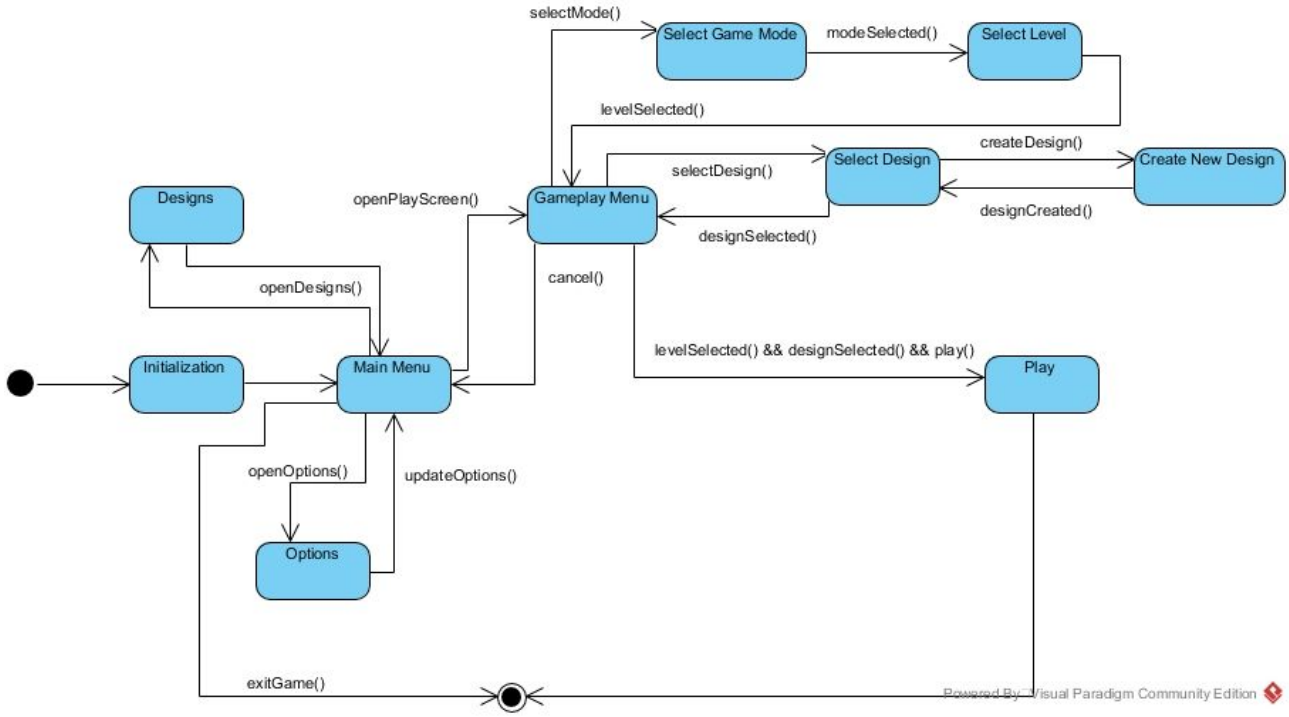
The activity diagram is used to cover the operations that our system does. In the following model, starting from the launch of the game, it is possible to view all possible steps that a player can take.



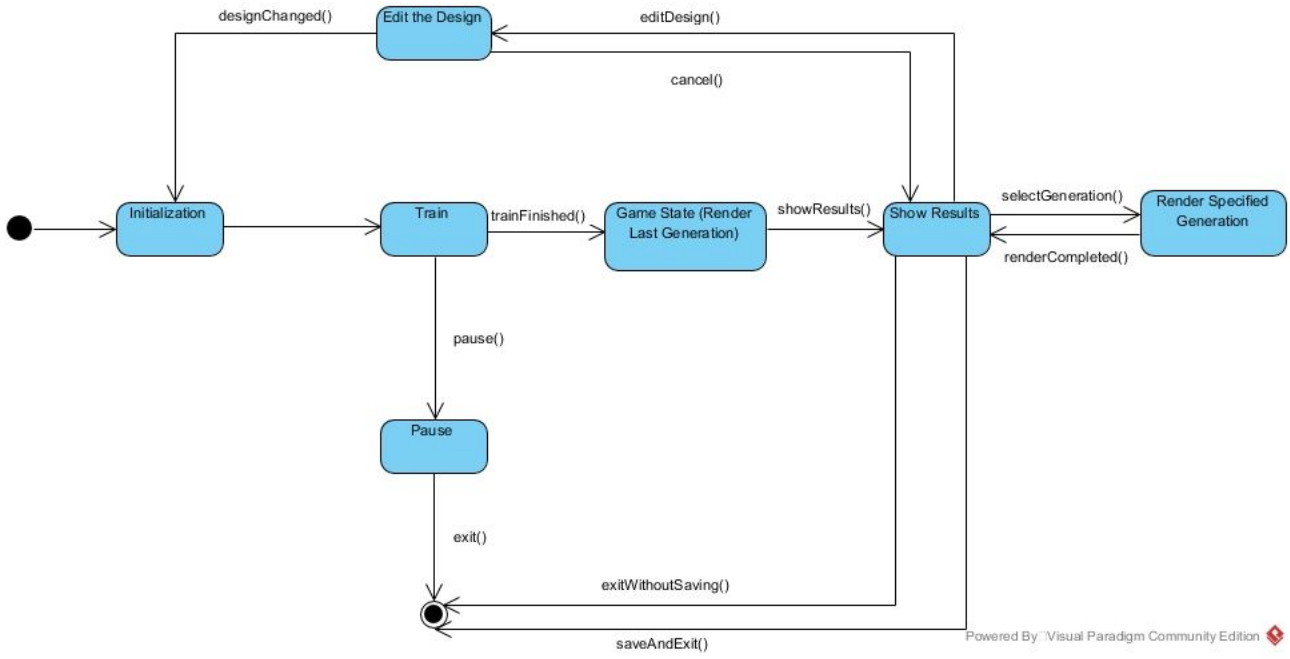

### 3.3.4.3 State Diagrams

The following diagrams represent states of certain classes. According to their conditions, these classes alter their behavior.

### 3.3.4.3.1 State Diagram for the States




### 3.3.4.3.1 State Diagram for the Game Flow



### 3.3.5 User Interface

Once user starts the game, he or she will be welcomed with main menu screen. Prototype of main menu screen is followed:

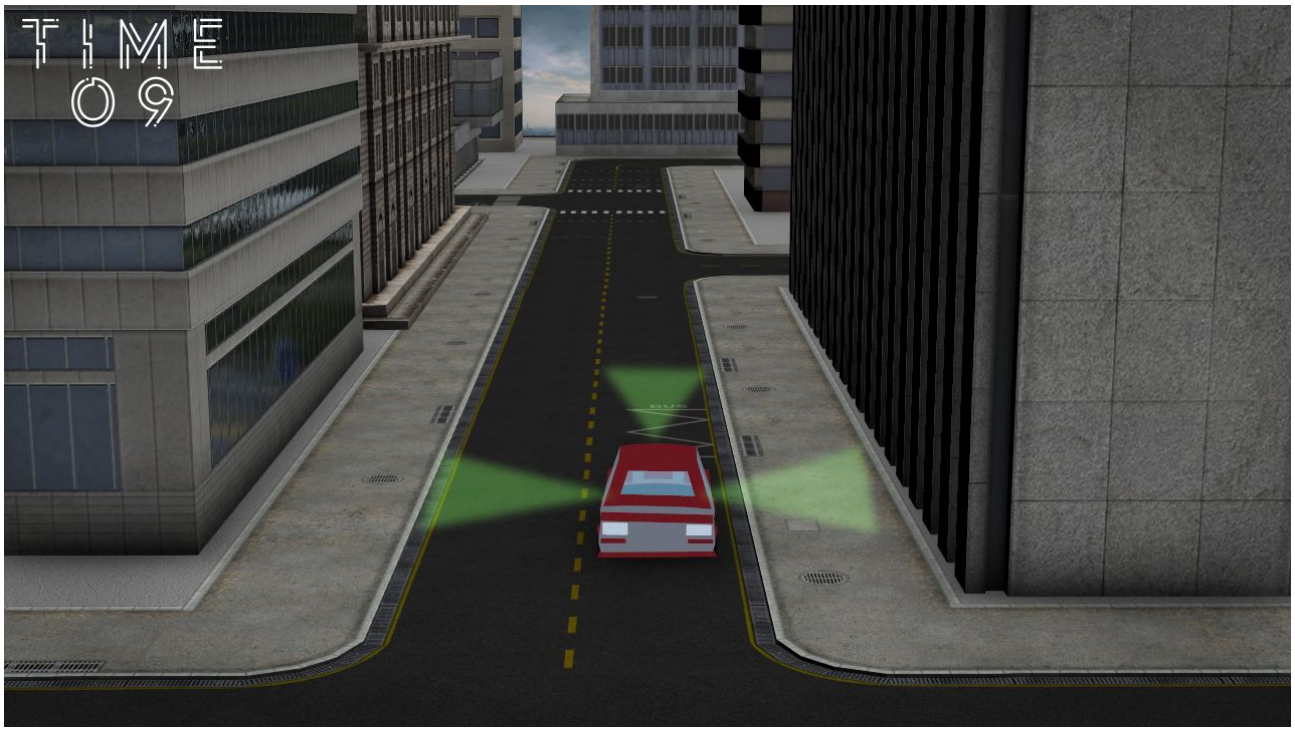

Here user has 3 options: Play, Options, Exit. It's clear that what "Options" and "Exit" buttons do, once user clicks on "Play" he or she needs to select what kind of mode he/she wants.


Details of these modes are already covered. After user selects the prefered mode, he/she needs to select a vehicle they already saved or need to create a new one.


Here user can see already created and saved vehicles, if they want to create a new one, it will be available to do so through “New” button.


In this screen, user can create a new vehicle through selected components. There are “Bricks” for the body part of the vehicle. “Engines” as the name suggest offers different kind of engine to run the vehicle. “Sensors” offers different kind of sensors in order car to see its surroundings. In “Tools”, user can select different kind of tools to help car to stabilize itself and achieve its goal in more efficient way.

After selecting the car, simulation will be run. After simulation ends, they will be welcomed with the game screen in which they will see how their vehicle did the job. Preview of the in game screen will be like this:




# 4. Conclusion

This report covered some information about the game's goal and its mechanics. Additionally, core system design of our game has been explained and possible scenarios that user might encounter are stated. Basic system overview as well as system model is created. Finally, mock-up images that allows us to show a brief look about the game interface are explained in detail.

In conclusion, the intention of DriveSim is to allow user to think creatively to solve a problem. With that, players will understand how a artificial intelligence thinks and create a scheme to allow a vehicle to drive without a driver. Additionally, there will be some elements that allow each user to compete with each other so that there will be competitive side of the of the game. Finally, graphical part of the game will be good enough to appear nice to the user's eyes. In order to create a game like this, game requirements have been analyzed and suitable system model has been provided.

# 5. References


[1] "Dream Car Builder". *Steam*. 2018. [Online]. Available: https://store.steampowered.com/app/488550/Dream_Car_Builder/. [Accessed: 11 Nov. 2018].

[2] Klande, David. "Cars Incorporated". *Wildduckgames*. 2018. [Online]. Available: http://www.wildduckgames.com/index.php/carsinc. [Accessed: 11 Nov. 2018].

[3] T. Coventry, "Requirements management – planning for success!", *Pmi.org*, 2018. [Online]. Available: https://www.pmi.org/learning/library/requirements-management-planning-for-success-9669. [Accessed: 11 Nov. 2018].

[4] J. Johnson, "What is a Change Request and How to Manage It", *Tallyfy*. 2018. [Online]. Available: https://tallyfy.com/change-request. [Accessed: 11 Nov. 2018].

[5] Nugent, Tim et al. "Entity-Component-Systems And You: Not Just For Game Developers - O'reilly Software Architecture Conference In New York 2019". *OreillyConferences*. 2018. [Online]. Available: https://conferences.oreilly.com/software-architecture/sa-ny/public/schedule/detail/71964. [Accessed: 11 Nov. 2018.]