CS 491 Senior Design
High-Level Design 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)

December 31, 2018

Table Of Contents

Table Of Contents **1**

1. Introduction **3**

1.1 Purpose of the System 3

1.2 Design Goals and Trade-offs 4

1.2.1 Design Goals 4

1.2.1.1 Ease of Usability 4

1.2.1.2 Optimization 4

1.2.1.3 Privacy and Security 5

1.2.1.4 Readability of the Software 5

1.2.1.5 Understanding How ML Works 5

1.2.1.6 Efficiency 6

1.2.1.7 Reliability 6

1.2.2 Trade-offs 6

1.2.2.1 Optimization vs Portability 6

1.2.2.2 Cost vs Reusability 7

1.3 Definitions, Acronyms, and Abbreviations 7

1.4 Overview 8

2. Current System Architecture **9**

3. Proposed Software Architecture **9**

3.1 Overview 9

3.2 Subsystem Decomposition 10

3.3 Hardware/Software Mapping 16

3.4 Persistent Data Management 17

3.5 Access Control and Security 18

3.6 Global Software Control 19

3.7 Boundary Conditions 20

Boundary conditions are the start-up, shutdown and failure conditions of the system. 20

4. Subsystem Services **21**

4.1 Entity - Component Subsystem 21

4.2 User Interface Subsystem 23

4.3 Neural Network Subsystem 24

4.4 GameManagement Subsystem 24

5. Conclusion **25**

6. References **26**

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, cars and 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 system. We will provide our design goals, trade-offs and detailed subsystem decomposition of our project.

1.1 Purpose of the System

Even though there are countless innovative ideas that are being used in games, there are not many games that uses machine learning as a gameplay mechanic. Our aim is by introducing this new concept in our game to create a unique gameplay experience for the player in which the players can see how different designs try to overcome the challenges by evolving with respect to them.

1.2 Design Goals and Trade-offs

By analyzing the design goals and the trade-offs of our system, we define critical aspects of our project. Then by these definitions, we can prioritize and improve some aspects of the project more than others in return this makes the end product more on par with the expectations.

1.2.1 Design Goals

1.2.1.1 Ease of Usability

Ease of usability is one of the most important design goals. A game must be easily understable and playable in order to be be enjoyable because if a player cannot understand the game, s/he cannot meet the main requirements of the game and therefore, get bored. Since the main goal of creating game is to entertain the players this would make all the efforts we put in the game meaningless. The ease of usability can be achieved by designing a clean user-interface, making the controls easy to use and carefully explaining the game mechanics that might need explaining.

1.2.1.2 Optimization

By optimizing our game we are not only making our game reachable to the users with low system specs but also making sure that our game runs smoothly in all systems. This would allow the players to enjoy the game without having performance issues.

1.2.1.3 Privacy and Security

Our game should not have access to anything that can be viewed as intrusion to privacy of the users. It can only make changes inside the folder that it is allowed to do so (i.e. save folder). We also plan to include an online scoring feature where users can see the some of the highlighted scores of other players. We have to make sure that no unnecessary data is stored online, and the data that is stored is safe from unwanted accesses unauthorized people.

1.2.1.4 Readability of the Software

The importance of readability is apparent in projects that are being worked with groups. The readability ensures that the different team members can easily understand the code and its purpose without wasting time. It is also important that if in future a new member comes, s/he can get up to speed with the project much more easily and quickly.

1.2.1.5 Understanding How ML Works

The purpose of our game is not just for it to be fun but also give insight about machine learning to its players. We believe that the best way to understand how something works is to observe it while it is working, so we plan to show the interactions between the neurons of the neural network in real time to the users. This way the users can see why their design acts that way.

1.2.1.6 Efficiency

The main design goal of the system is efficiency, to achieve that, the system must be able to work in high performance. Since, a smooth gameplay is one of the most important feature which increases the player's urge to play the game, we are going to minimize the memory, CPU and GPU usage. In order to achieve that, first, we are going to implement the code in the most efficient way possible. Also, we are going to design the system so that the workload of the objects is going to be nearly balanced.

1.2.1.7 Reliability

Our system will be reliable in terms of being consistent with the boundary conditions. The system should not respond with any unexpected results -like bugs, crashes- which are not specified in the boundary conditions. In order to provide that, we are going to test the system in all possible ways during and after the development stage. Also, the boundary conditions will be selected carefully and with caution so that there won't be a case with which puts the system in an unexpected situation. This will provide the system to foresee the possible fatal failures which will be dealt with.

1.2.2 Trade-offs

1.2.2.1 Optimization vs Portability

We can choose to deploy our product on multiple different operating systems but this would cost us the time we can optimize our game to run smoothly. We decided that

we are going to deploy only on Windows systems. This will allow us to focus on one system only and in return a more optimized end product.

1.2.2.2 Cost vs Reusability

Currently, we do not have any plans to improve our project so that we can build it on another system. Our plan is to improve our project with the existing system. Therefore, in the future stages of development, we can aim to create a maintainable, low-cost product and sacrifice reusability while doing so.

1.3 Definitions, Acronyms, and Abbreviations

Unreal Engine: Unreal Engine 4 is a complete suite of development tools made for anyone working with real-time technology. From enterprise applications and cinematic experiences to high-quality games across PC, console, mobile, VR and AR [1].

Entity Component System (ECS): ECS is an architectural pattern that is mostly used in video game development. ECS allows a greater flexibility in defining entities where each object in a game's scene is an entity (e.g. enemies, npcs, bullets, vehicles, etc.). Each entity is consists of components which adds behaviour or functionality. This makes it easier to change the behaviour of an entity at runtime by just adding or removing components. This is especially very useful in deep and wide inheritance hierarchies because it eliminates the ambiguity problems in just systems. [2]

Source Control Management (SCM): Source control (or version control) is a system that records to changes to files so that the user can recall to the specific versions of that file(s) later [3]. This management system also makes it very easy for developers to work on same project simultaneously since they can easily upload and merge their work with these tools (e.g. Git, Mercurial).

1.4 Overview

DriveSim is meant to be a informative, entertaining and challenging game with a unique gameplay mechanic to simulate driverless cars. The core feature of the game is making a neural network to learn how to drive while not hitting static and dynamic objects such as walls, cars, pedestrians and meet the goal of different game modes. To do that, the players will design a vehicle model, which includes different body parts with different shapes and various types of sensors. These sensors will be used to detect objects and make a decision which will make the vehicle adjust its behavior.

Our intention to make the game modes such that they will have different objectives to achieve. By doing so, we allow the users to experiment with different vehicle designs for different gameplay modes and observe that a vehicle design that may be successful for a specific game mode may not be successful in other game modes. Therefore, the player should consider all the aspects of the game mode while designing the car.

Also, we intend to develop a maintainable game so that the players will not experience bugs or glitches.

2. Current System Architecture

Neural networks and genetic algorithms have been applied to games in order to make them learn how to play. Also, 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 [4] 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 [5].

3. Proposed Software Architecture

In this chapter, we discuss what our architecture will look like and what kind of subsystems / sub structures we will implement to optimize our work in terms of reliability, performance, extensibility and maintainability.

3.1 Overview

Our high level design includes 5 different subsystems, such as the engine, the game manager, neural network functions, UI, entity component system. For our game to function nicely all these 5 subsystems should work without causing any significant drawback and they should communicate with each other efficiently. Our main approach

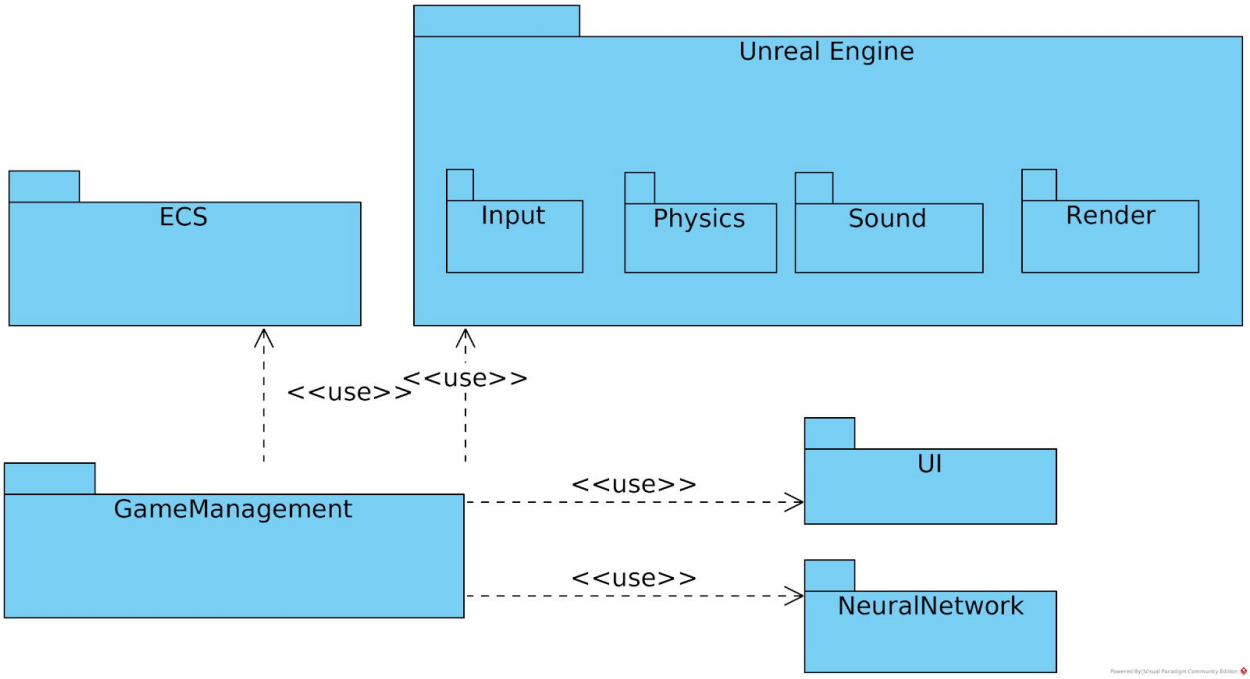
to structure the project is using the data oriented design patterns. We believe that creating too many abstraction layers where one sits on top of the other distances us from the actual hardware which in return causes in performance penalties, such as frequent cache misses caused by virtual function calls. To avoid such issues, we will try to mostly use flattened data structures such as aligned vectors or arrays and create functions that operate on batches of data rather than a single unit. In fact the ECS or the entity component system design pattern just does this. Separating components from the entities and bundling them together so that we can process them in batches.

Secondly we will focus on separating the UI, input and the game logic. The game manager will deal with the game logic and the game loop while communicating with the UI and the input layer through an abstraction layer we will write to make our game logic engine and platform agnostic. This will allow us to break the dependency between these three layers so that we can implement, improve and update these three layers to our liking whenever we want. We will not strictly use MVC (model view controller) approach since it sometimes be too limiting especially if we decide to include IMGUI (immediate mode gui) to our game.

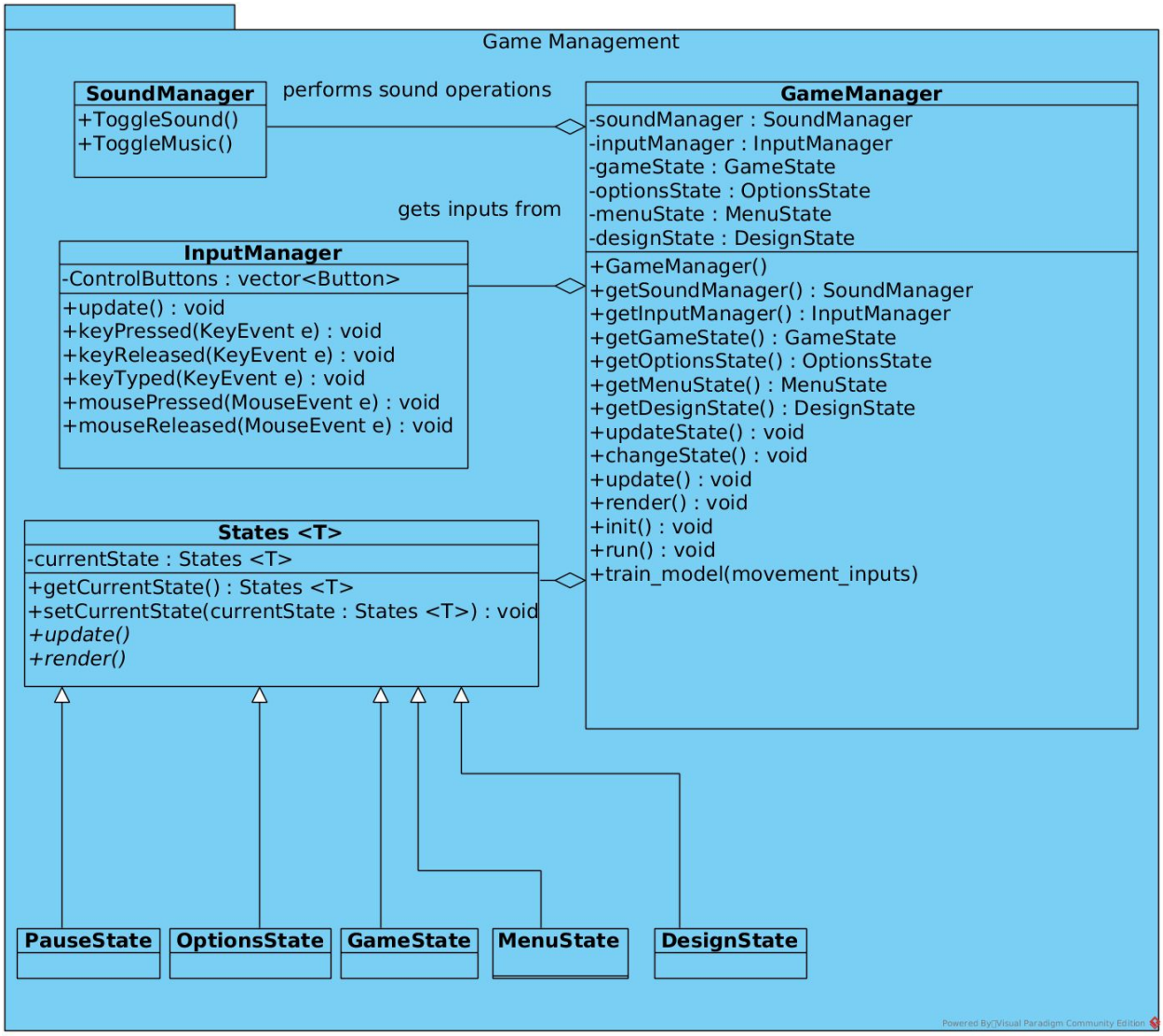
3.2 Subsystem Decomposition

Subsystem decomposition is about structuring the software into subunits based on their function. This method allows us to show these sub structures graphically. This graphical representation is important since it helps us discover unnecessary

dependency issues before hand so that we can create more concrete and whole subsystems. Our current high level subsystem decomposition is given below:

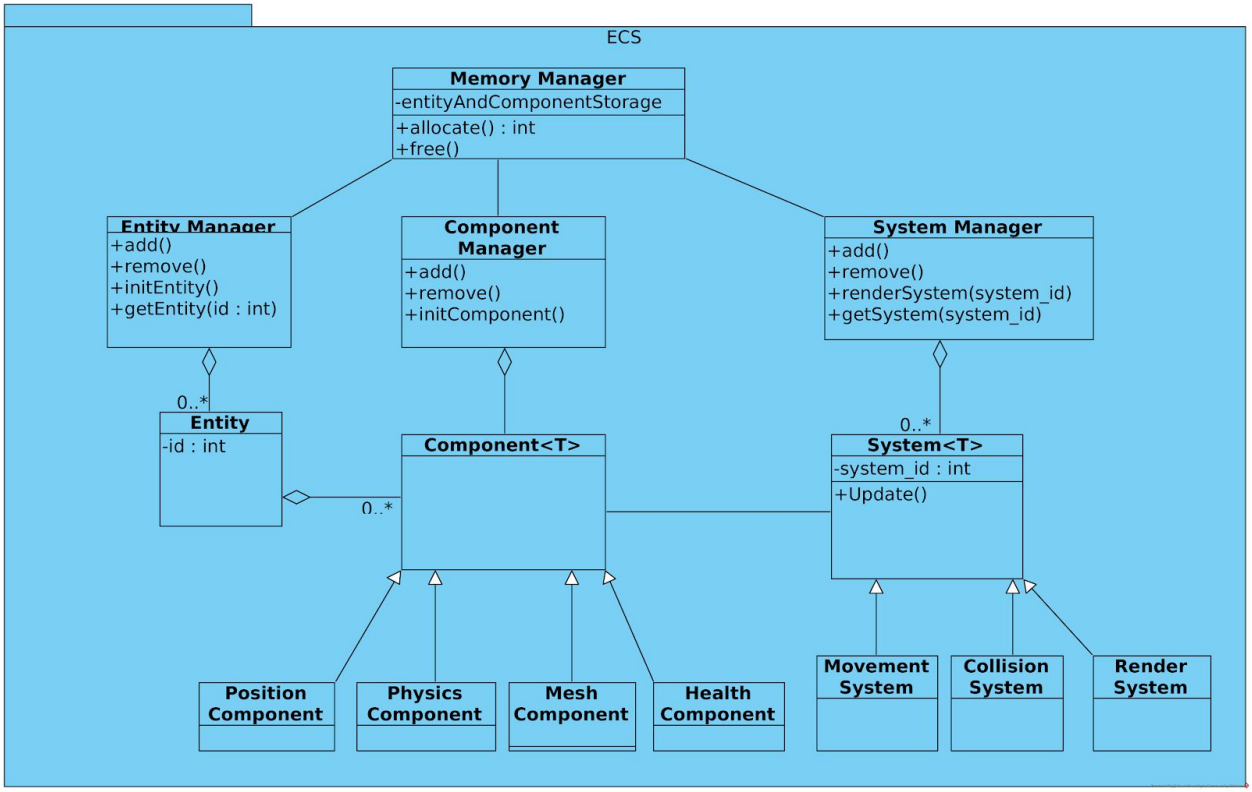


Game management subsystem will deal with the core game logic and the game loop. It will process the input, update the entities and hold the current and the past necessary states.

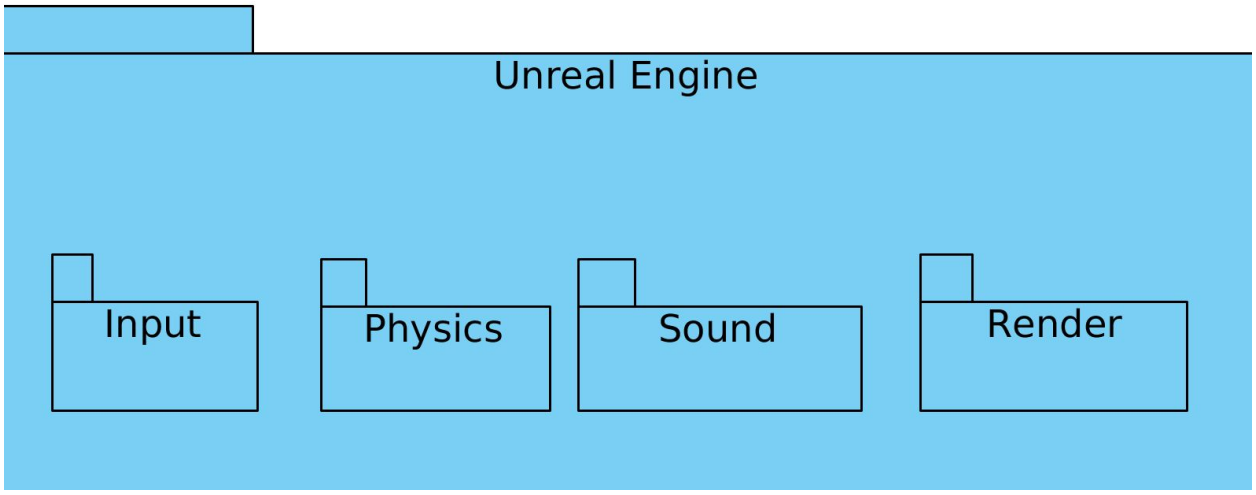


Entity Component System (ECS) will deal with, as the name suggest, entities and the components related to the game entite. It will include a job system and a component system. Component system's main function will be attaching or detaching components from or to entities. The job system will include what is called "job functions", type of "batch" functions that we have mentioned above. These job functions will work on a couple of components and update them in batches. This system will increase the overall

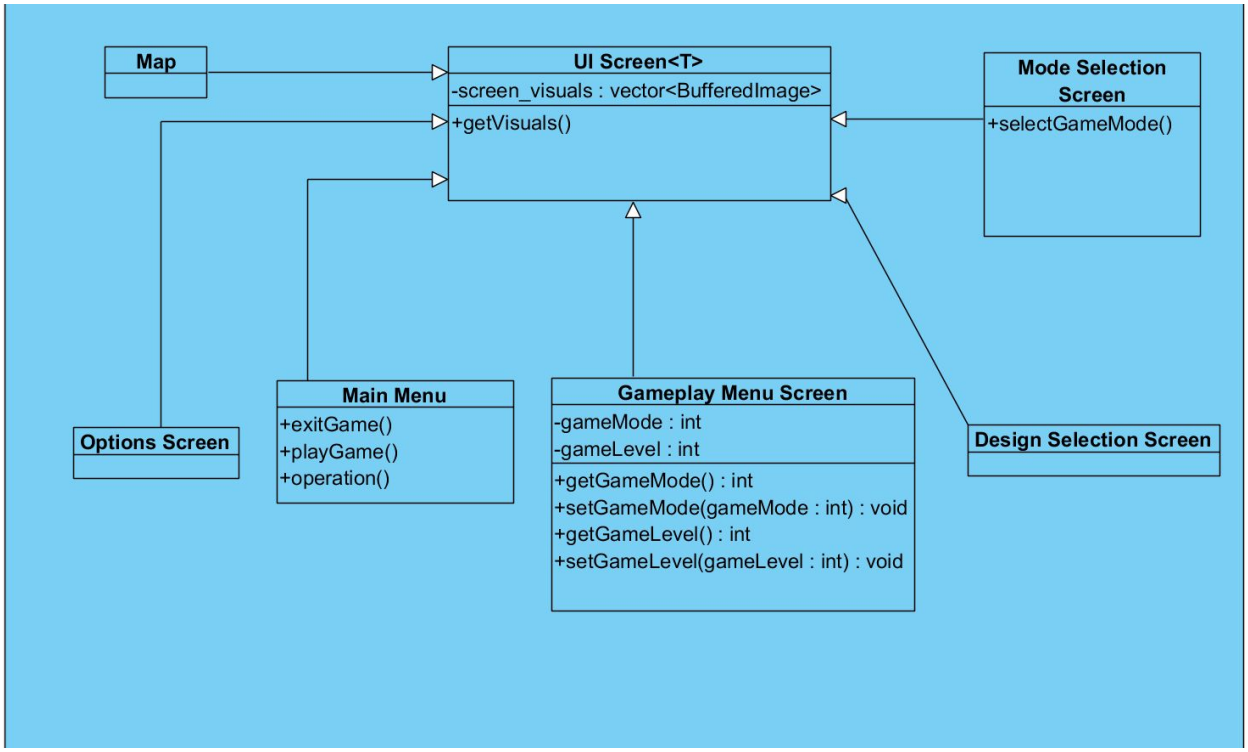
performance of our system and will create a nice decoupling between the entities and their components.



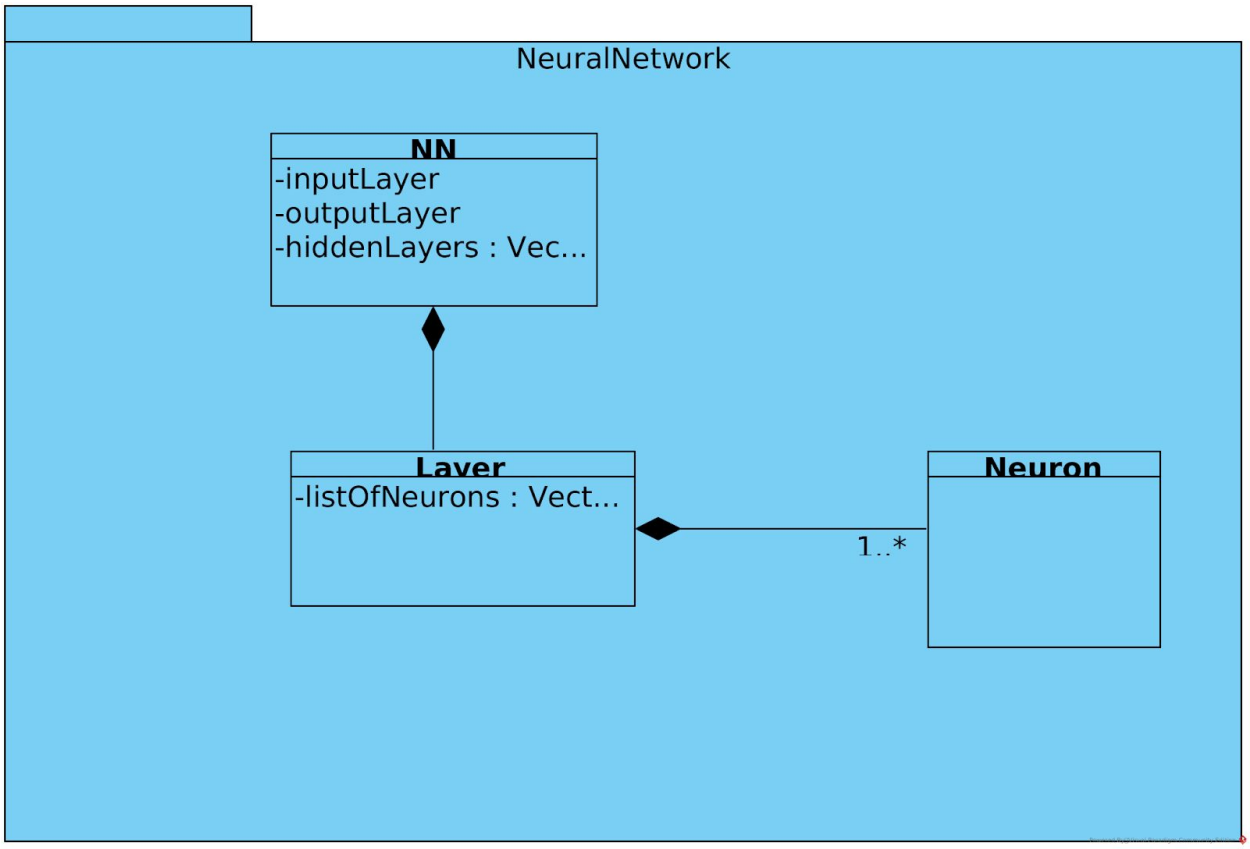
We will use UE4 as our game engine. It will deal with the rendering, physics, mathematics, sound and input. We will create some coroutines around these systems so that we can easily make the engine talk with our game logic.



As mentioned above, we will try to separate the UI from the game logic as much as we can. UI subsystem will deal with the user input, create appropriate data structures and pass appropriate messages to the other systems that need to act on.



Neural network subsystem will encapsulate all functioning related with the neural network such as training agents, cars, updating node values, node places etc.

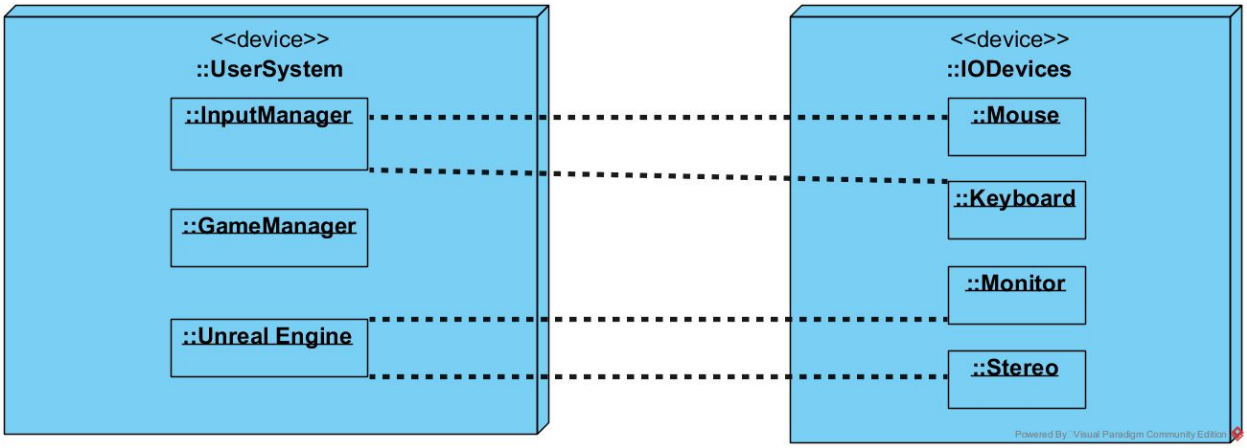


3.3 Hardware/Software Mapping

Unreal engine provides us with powerful and gigantic IO libraries. We will create a layer or a wrapper around the part of the library that we actually need so that we can isolate the parts we need and specifically work on those. This will create simplicity and will help us adjust the input system as we like.

Since our software is a game, we will need hardware accelerated graphics i.e a dedicated graphic processor unit (GPU). Fortunately unreal engine can easily handle the rendering side of our game. It allows us to write configurable shaders, and do gpu

calculations. Our game can, potentially, be cross-platform since UE4 will handle all platform specific details.



3.4 Persistent Data Management

Currently we are not planning to implement online features. As mentioned above, in the late stages of the development, if we were to implement the aforementioned offline features without facing much difficulties, we are considering to implement and online highlighted scores feature for which the users will be able to observe the scores of the other players. So, we might be able to keep an online database for scores. As for the rest of the game, there will be a local database to store game related data.

3.5 Access Control and Security

In this part we will talk about security levels of DriveSim and its access matrix. Security is one of the important aspect of a game. A game that has low security checks can ruin the experience of other people. For this reason, it’s important to make sure user can not modify its scores globally. Every user will have its own game replay. They will be able to share these replays. But inside of these, DriveSim will not save the game mechanics data, instead it will save only the position of the car. For this reason, users who owns the replay will be only “render” the replay and this will guarantee that they will not change the inside parameters.

Additionally, we want to save game settings in a serialised way, this will protect the DriveSim from users with bad intentions. In other words, we want to serialise and protect DriveSim’s I/O files so that they won't be reachable from outside. This also includes the game communication packets in network protocols.

Finally since users will write their personal informations such as age, profession etc. while registering, we want to make sure these data are safe and protected well. Due to technical concerns, we will not share these informations to outside and will be only used to improve DriveSim’s user experience.

3.6 Global Software Control

This section will give overall information about the system within DriveSim and their synchronous working principle.

DriveSim will use Unreal Engine’s powerful engine. This will allow us to capture inputs easily, handle the assets such as sound etc. without fragmenting the memory and play in a stereo mode and finally render the graphical visuals in efficient way.

We will have our own Entity-Component System(ECS). This will allow us to control and update entities in a efficient, fast and reliable way. ECS will also provide us nice decoupling between components such that dependency between components will be minimum.

DriveSim’s game manager will be “controller” of the game state, settings and inputs. This will be main controller that will control the current state of the game. Lastly, It’s important to note that our Neural Network will be independent of the game.

These different system will allow DriveSim to work on different levels such that they won't depend on each other. This will provide fast and elegant system that is needed for heavy tasks. During these tasks, because of the multi-level system, they will work in synchronous way.

Finally, we will separate our UI from our game logic. It’s important to have such system that can work without UI and vice verse because during the implementation or later phases one of them may require functionality change and dependency between them would make it cost more.

3.7 Boundary Conditions

Boundary conditions are the start-up, shutdown and failure conditions of the system.

Start-up: The executable file of the game will be .exe file so it will not require an install. At startup time, the GameManagement subsystem will be triggered. The GameManagement subsystem will access the data which is required for the user interface, and the sound effects.

Successful Termination: User can follow GUI to close the program anytime. Based on user's decision any successful termination will either update the game state and the save data or it will simply exit.

Pause: If user switches to using another application, our game window will lose focus this then will cause the game to transition into a safe pause state. After the transition if user refocuses to our game window, game can continue without any problems.

Failure or Panic: Panic is a type of exception that is released if the game gets into a state that it cannot recover from. On such cases, we will try to make sure that we have a not corrupted save file, on the which we can save the current state and terminate the program. In the rare case where we discover that the save file is actually corrupted, we can try to recover it or create a new save using the current game state.

4. Subsystem Services

4.1 Entity - Component Subsystem

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.

This systems captures the entities and components of the software. Entities and Systems are shown as parts of managers however aggregations are only to show relations and actual data are stored in arrays managed by the memory manager.

- Memory Manager

  Manages memory and related requests from other systems such as Entity or System Manager. Memory manager stores array which information about entities reside. It interacts with these and other systems when necessary.

- System Manager

  Knows about System<T> types of Systems such as Movement System or Component System. Updates other systems to make them update components which they are responsible.

- Entity Manager

Knows about entities, handles operations related to entities such as add and remove.

- Position Component

  Gives capabilities related to positioning to the entity which it is added.

- Mesh Component

  Gives capabilities related to drawing to the entity which it is added. Planned to control drawing of the entity.

- Physics Component

  Gives capabilities related to physics, such as collisions to the entity which it is added.

- Health Component

  Gives capabilities related to health to the entity which it is added. Entities with this component will have health value which will be updated by the relevant system in batches.

- Movement System

  Updates positions components which effects the positions of the entity. Also gets information related to entities which is planned to be useful for extracting related data.

- Collision System

  Updates physics related components and used in physics related calculations such as collision detection.

- Render System

  Updates mesh components to control display of entities which have related components.

4.2 User Interface Subsystem

This subsystem contains the graphical menu elements of that the players will see.

- UI Screen <T>

  Will provide the features that are shared by all of the screens. This will provide an inheritance relationship, and common adjustments will be done only once.

- Main Menu

  Consists the elements that will be shown in the main menu.

- Options Screen

  Consists the elements that will be shown in the options menu.

- Mode Selection Screen

  Consists the game play modes and the selection of the gameplay mode will be done in this screen.

- Design Selection Screen

  Provides the screen in which the player will choose the vehicle that s/he designed earlier or it will also have an option to create another vehicle design.

- Map

  Keeps the elements of the map that the game will be played on.

- Gameplay Menu Screen

  Provides the elements related to the in-game menu elements.

4.3 Neural Network Subsystem

This system consists of classes needed for the neural network to be used.

- NN

  The object which will store information about the neural network.

- Layer

  Object that will store information about the nodes of a layer of the network.

- Neuron

  The basic node for the neural network.

4.4 GameManagement Subsystem

The GameManagement subsystem will deal with the communication between the different subsystems, and coordinate them. This coordination will be based on the game states and the other manager objects provided above.

- GameManager

  GameManager class is the facade class of the system, the remaining control objects provide inputs for the GameManager and Game Manager decides which action will be taken.

- InputManager

  Provides user input specifications and deals with them.

- SoundManager

  SoundManager provides control over the sound options.

- States classes

  States classes are for differentiating the states of the game, they will do similar tasks with different specifications.

5. Conclusion

Throughout this report, technical informations about the DriveSim is explained. High-level designs, our current and planned system is clarified and our subsystem and their working details are related. These specifications gives detailed answer how and why DriveSim will be different compared to other similar games. DriveSim is aimed to be implemented in correlation with these factors.

DriveSim's unique and simple UI will allow users to understand the basic concepts of the game easily and clear the confusion that new players face. It will be simple but elegant enough give unique view.

DriveSim will provide unique way to challenge player's creativity and teach them how neural network thinks based on their choices. This created opportunity to users to challenge each other and create a competition. We believe that this competition will improve driverless car technology. Experiences learned through the game's life can be transferred real-life situation, if needed.

6. References

[1] Unrealengine.com. (2018). *Unreal Engine 4*. [online] Available at: https://www.unrealengine.com/en-US/features [Accessed 30 Dec. 2018].

[2] En.wikipedia.org. (2018). Entity–component–system. [online] Available at: https://en.wikipedia.org/wiki/Entity%E2%80%93component%E2%80%93system [Accessed 30 Dec. 2018].

[3] Git-scm.com. (2018). Git - About Version Control. [online] Available at: https://git-scm.com/book/en/v1/Getting-Started-About-Version-Control [Accessed 30 Dec. 2018].

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

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