

**Software Design Document for:**

# **OOber Taxi**

**Revolutionising the taxi game!**

“We’re not trying to reinvent the wheel with this one”™

All work Copyright ©2018 by Carré

Written by Jack Matters, Cordell Smith, Michael Bell

Version # 1.00

Friday, April 21, 2018

Table of Contents

[OOber Taxi 1](#_Toc512082982)

[Design History 3](#_Toc512082983)

[Version 1.00 3](#_Toc512082984)

[Game Engine Architecture Overview 4](#_Toc512082985)

[General Overview 4](#_Toc512082986)

[Philosophy 4](#_Toc512082987)

[StarUML 4](#_Toc512082988)

[Third Party API 4](#_Toc512082989)

[OpenGL 4](#_Toc512082990)

[Bullet Physics 4](#_Toc512082991)

[Assimp 4](#_Toc512082992)

[Lua 4](#_Toc512082993)

[Created Classes 4](#_Toc512082994)

[Game Control Engine 4](#_Toc512082995)

[Game Asset Factory 5](#_Toc512082996)

[Game World 5](#_Toc512082997)

[Input Manager 5](#_Toc512082998)

[Physics Engine 5](#_Toc512082999)

[External File Formats 6](#_Toc512083000)

[Scripts 6](#_Toc512083001)

[Script Location 6](#_Toc512083002)

[Script Type 1 6](#_Toc512083003)

[Resources 6](#_Toc512083004)

[Game objects 6](#_Toc512083005)

[Textures 6](#_Toc512083006)

[Terrains 6](#_Toc512083007)

[Shaders 6](#_Toc512083008)

[“StarUML Appendix” 6](#_Toc512083009)

# **Design History**

During the design process of our game engine, the ideas that we intend to bring into fruition will be documented with the intent to allow us to examine the success and failures of our project. Beginning with version 1.00, we will be adding smaller 0.10 increments or even 0.01 indicating minor changes pertinent to the project. When a significant milestone is reached, this will be set during the project planning stage, it will be indicated by intervals of 1.00.

## **Version 1.00**

Version 1.00 is the initial start on the game software design document. An example game document has been used as the foundation for this one, mainly copying the layout for use with this document.

# **Game Engine Architecture Overview**

## **General Overview**

### **Philosophy**

*This section will be completed next assignment.*

### 

### **StarUML**

Please refer to appendix ‘StarUML Appendix’

## **Third Party API**

### **OpenGL**

*This section will be completed next assignment.*

### 

### **Bullet Physics**

Bullet physics is currently not behind a facade. It is coded into the Physics Engine, making it difficult to swap out for another API. In the next assignment, we will aim to facade it in a way that swapping APIs is relatively easy and would require minimal alteration.

### **Assimp**

*This section will be completed next assignment.*

### **Lua**

Lua scripts can manage a lot of the functionality within your engine without having to recompile the entire project. All game logic and configuration files should be exported to scripts and compiled at run time. This also allows for ease of update by non-programmers. Lua scripts are loaded through calling the script manager class, all scripts must be placed in the script resource folder (path found below). A script may be loaded by calling the LoadScript() function and passing in the name of the lua script that you would like to load. This has not been fully implemented for this assignment but a working framework is attached for demonstration.

## **Created Classes**

### **Game Control Engine**

The game control engine is the main hub to which all the components of the game engine communicate with and satisfies the controller component of the MVC design pattern. Within this class, all of the engines assets are initialised within an Initialize() function. Anything you wish to be loaded prior to the creation of a new game world should be loaded in this function first for future access and possible game changes. When all the game engine components have been initialized, the main game loop is started and called using the GameLoop() function. This is where any changes made as a result of user input or game logic such as game states, objects, animation, physics or sound and should be passed into the m\_gameWorld object to be updated and rendered for the user to see. When the user exits the application by closing the window or pressing ‘esc’, the main game loop will end and then code management will proceed to take place calling the Destroy() function. Anything that is created by all of the game components are destroyed and deleted in this function to free up any memory that has been used.

### 

### 

### **Game Asset Factory**

The GameAssetFactory class utilises the factory method design pattern in order to allow the separation between object creation and type. Game assets such as terrains, objects and NPCs can be created and loaded in through the CreateAsset() function that takes the type of asset you wish to create as a first parameter and the file path where the data is contained as the second parameter. The Terrain, NPC and Object class inherit from the parent class IGameObject. The type you specify in the the create asset function can be of type OBJ\_TERRRAIN, OBJ\_NPC and OBJ\_OBJECT and using a switch case, will return a newly created object of that type containing the loaded data.

### **Game World**

The game world class represents all of the data coming together to create an instance of a game world. This includes terrain, objects, characters and any other assets loaded to create a scene.

### **Input Manager**

*This section will be completed next assignment.*

### **Physics Engine**

The Physics Engine is responsible for all physics related simulations, including collision detection. All game object data is passed into this class via the GameWorld class in the Initialize() function in the form of a data structure. Also during this step, ActivateAllObjects() is called to ensure that every rigid body object is active throughout the entire program. During the main game loop, Simulate() is called from the Update() function in the GameWorld class. Simulate() performs all physics calculations during a step simulation. After the simulation, the updated game object rigid body locations are passed back to the GameWorld class to be drawn. This process happens every step until the program is ended.

### 

# 

# **External File Formats**

## **Scripts**

### **Script Location**

All scripts are to be placed in “$(SolutionDir)CarreGameEngine\res\scripts”. All script files are to be placed here, without any subfolders. Recommend naming script files something meaningful.

### **Script Type 1**

*Scripting was not implemented in this assignment. This section will be changed for the next assignment.*

## **Resources**

### **Game objects**

All game objects are to be placed in “$(SolutionDir)CarreGameEngine\res\objects”. All related objects are to be placed in their own subfolder, the name of which is at the user’s discretion. All other objects are not to be placed in a subfolder. Recommend naming game objects something meaningful.

### **Textures**

*Textures was not implemented in this assignment. This section will be changed for the next assignment.*

### **Terrains**

All terrains are to be placed in “$(SolutionDir)CarreGameEngine\res\terrain”. All terrain files are to be placed here, without any subfolders. Recommend naming terrain files something meaningful.

### **Shaders**

All shaders are to placed in “$(SolutionDir)CarreGameEngine\res\shaders”. All shader files are to be placed here, without any subfolders. Recommend naming shader files something meaningful.

# 

# **“StarUML Appendix”**

This appendix provides a StarUML overview of our Game engine.

