unity3d Driving Simulator

2015

Created to show how the program works, decisions made & why

Fontys Automotive

Automotive Campus  
Steenovenweg 1   
5708HN Helmond

## Preface

This document has information about the logic of placing objects on the Grid. It does not talk about physics of the driving simulator nor about the using of buttons to place objects on the Grid. We assume that you have a basic knowledge of the program and know how to work with Unity.

For family and friends

who supported me in following my dream

of destroying hardware’s and creating software’s.

Edson Sint Jago – January 2015

Contents

[Preface 1](#_Toc409095834)

[Introduction 3](#_Toc409095835)

[Before we touched the program 3](#_Toc409095836)

[Decisions 3](#_Toc409095837)

[After we touched the program 3](#_Toc409095838)

[Placing Big Objects 4](#_Toc409095839)

[Removing Big Objects 4](#_Toc409095840)

## Introduction

Before we started working on this project, another team had already worked on it. They created the Grid to successfully place objects on it and after pressing the Enter button, lets the simulation start so the user can run around in the world.

Primary decisions that they made were that one object can be placed on one square on the Grid, “Raycasting” to know where the mouse button position is when clicked and making the Grid only one specific size (or else it will break).

When we started this project we were given the task to create bigger objects, which takes up more than one space on the Grid. With this document we want to show the reader what has been done, where some specific codes are and why the code is made this way.

## Before we touched the program

Where is everything you asked? It is in the class “GridScript.cs”. This class creates the Grid and partially includes the logic to place the objects on the Grid. This class calls the method “PlaceObject” from the class “PlaceItem.cs” which calls a method from the class “Gridcell.cs” to check if the selected “GridCell” object is has a GameObject on it. One cell can have one GameObject, but the object is added into an ArrayList. That is actually weird.

How do you know which object you have selected before placing it on the square? This is also in the “GridScript.cs”. The method is called “setSelectedObject”.

## Decisions

We decided not to recreate the grid. We are doing this because the project consists of two weeks and to recreate the grid we estimate the project to take a month fulltime (every workday from 9 to 5). If the grid is recreated, it can be more extendible and manageable for future teams. We will just add/edit the code to make the functions that is wanted to work.

We edit the code in such a way that objects with more than one square can easily be placed on the grid. We also have to take in account if an object exist with weird shapes, for example an L-Shape building.

## After we touched the program

Since an object can now occupy more than on cell on the grid we need to create a method to check if an array of cells are occupied, to occupy multiple cells and delete multiple cells. To do this we added a new property called “isOccupied”, which is a Boolean and returns if the cell has been set to Occupied.

### Placing Big Objects

We created a method in “GridScript.cs” that checks which type of object the user is going to place on the Grid and if there is enough space for this object. After this is done it places the object on the grid and sets the correct GridCell’s isOccupied property to true. This makes it that a big roads (that have been added in this method) will not be able to overlap with other objects.

### Removing Big Objects

When removing big objects from the grid we have to remember to also toggle the “isOccupied” property of the correct cells. This is done using the method “OpenCells” (in “GridScript.cs”) which checks again what type of object exists on the GridCell and how many GridCell to the right, left, top left-top have to be opened up. When adding more objects it is possible to easily place them in this check.