**Documentation Preface**

Hi future coops,

The following documentation is written by the Spring 2013 work term coops (May - August), Aaron Chung (Programmer) and Jennica Collette (Planner). The project has been going on for 3 years but there has been inconsistency in documentation procedures between work terms. By the end of the work term, we have been working on the project from backend to frontend, this documentation attempts to summarize our work and, furthermore, provide explanation for the convention we have been created and used during the work term.

This documentation is divided into 2 parts: the Hierarchy documentation and the Scripts-side documentation. This documentation assumes the reader to have basic background knowledge of Unity. The Hierarchy documentation explains how game objects and scripts in Unity interacted with each other in a high level view, whereas the Script-side documentation provides a more detailed explanation on how data is manipulated in compilation and runtime, and how data is being saved and retrieved from backend to frontend, and how does all these affect the way we design the UI (User Interface).

The project code for this project is IRUS, shorthand for Interactive Real-Time Urban Simulator. The main goal for us the Spring 2013 work term coops is to have the functionalities polished and ready to be used by the public. Since IRUS is in a relatively young stage at this moment, we have layout a lot of foundational and sustainable code that hopefully will be reusable for future terms. Before we took the project in May 2013, the basic functionalities of IRUS have already been introduced, however, due to the lack of Quality Assurance and Consideration of User Experience, there are still a bunch of bugs in the system that prevents the project to be published. In addition, our manager Dr. John Lewis has purchased the Unity Pro for a year this term. This helps us to gain access to a lot of advanced functionalities like Profiler and Occlusion Culling.

Throughout the Spring 2013 work term, we have worked on a couple of things listed below:

1. We introduced the save and load functions using text files to allow users to save and undo their changes.
2. We moved the “online database” to a “. Csv” text database under the user Assets/Resources folder so that the user can access the information without connecting to the Internet. Preset and User data are being loaded on runtime to increase flexibility.
3. We redesigned the Right Of Way Editor and the Urban Design Editor so that the user interface is more intuitive and user friendly. We have implemented the save and undo functions with them as well. In addition, we integrated the Edit Object UI with the urban design editor due to the related functionality.
4. We reduced the rendering time dramatically by reducing the vertices, triangles and changing the Mesh Collider to a Box Collider of the buildings and street furniture. The occlusion culling has also helped a lot on that.
5. We introduced a new naming convention for all the buildings, parcels, lots, road segments and intersections so that every object has a unique name that is particularly useful for the save and load functions.
6. We brought the different male, female and senior avatars (pedestrians) into King Street and wrote a simple AI for them.
7. We created a simple Login Scene and Tutorial Scene with animations to give users guidelines.
8. We tried to publish a build version for the first time and resolved some incompatibility issues arise between the editor version and the build version.

Here are a few useful links that gain us useful insight on understanding how Unity works, you may want to bookmark it in your browser:

Unity Scripting Reference

<http://docs.unity3d.com/Documentation/ScriptReference/index.html>

Unity Execution Order of Event

<http://docs.unity3d.com/Documentation/Manual/ExecutionOrder.html>

Unity Script compilation order

<http://docs.unity3d.com/Documentation/Manual/ScriptCompileOrderFolders.html>

In addition, there is an extremely useful Unity Community forum that will usually show up in the first few pages in the Google search result list.

The scripts of this project are written either by UnityScript (javascript) or C#. Most of my code is in UnityScript. People called it UnityScript because of its lack of functionality of the standard javascript.

**Important Concepts In 3-dimensions**

In a general sense of how Unity works, it refers each game entity as a *GameObject*,i.e. even a camera, a game object array are game objects, and each *GameObject* can be attached with scripts, animations, audio listener and more to give different effects in the game environment. When you click on a game object in the *Hierarchy*, you can see the corresponding components attached to the game object in the *Inspector*.

Every game object has a *Transform* component that specifies the position, the rotation and the scale of the game object relative to the “parent” game object (or the World). It is worth to mention that it is the *Transform* component that has children but not the game object itself. From my understanding, it is because the *Transform* is what really gives meaning to the game object in the Unity world. You will gain more knowledge when you dive into the code. So, you should already notice that there are two different “*Transform*”, the global variables that are with respect to the World and the relative variables that are with respect to the parent game object. This will actually be one major issue to tackle with when instantiating objects within a parent. The *Transform* that showing in the *Inspector* is the relative one.

As you can see from above, game object has such a vague definition. The game object that you are expecting are the game objects with *Mesh Filter* and *Mesh Renderer*. These two components are what give visibility to the game object in the Game scene. The *Mesh Filter* provides the shape and geometry of the game object and the *Mesh Renderer* renders those properties. You can add different Materials, Shadows and more on the *Mesh Renderer* to refine the game object. However, only the mesh filter and mesh renderer are not enough to give mass to the game object, the user (or the Character Controller) can still penetrate through it in the Game scene. In order to make a game object un-penetrable, one should add a *Collider* to a game object. (Please note that you can have game object with a Collider but not a Mesh Renderer, they are two separated concepts, but people usually use them at the same time.) There are a variety of colliders provided by Unity: *Box Collider*, *Sphere Collider*, *Mesh Collider*, etc. Different *Colliders* can affect the runtime performance dramatically. For example, a mesh collider may take more time to render than a box collider because there are more triangles on it.

In runtime, game objects are being instantiated and destroyed repeatedly. However, Unity does not destroy game objects immediately, it waits until the next *Update* loop. In addition, there are also nuances between the editor version and the build version in handling garbage collection. As you will see more details in the Scripts-side documentation, this is related to the incompatibility issues mentioned above.