Virtual Game Simulation:
Create a Virtual Environment

Challenge Problem and Resources

Developed by:
The teachers, students, and mentors in the Gaming Research Integration for Learning Laboratory™ (GRILL™) Summer 2014

# TABLE OF CONTENTS

1. **CHALLENGE PROBLEM: CREATING A VIRTUAL ENVIRONMENT** ........................................... 3  
   1.1. **THE TOOLS** ......................................................................................................................... 3  
   1.2. **THE CHALLENGE** .................................................................................................................. 3  

2. **TUTORIALS** ................................................................................................................................. 4  
   2.1. **DESIGNING AN EFFECTIVE GAME SIMULATION** ................................................................ 4  
      2.1.1. Define Your Game ....................................................................................................... 4  
      2.1.2. Design Documentation ............................................................................................... 5  
      2.1.3. Game Environment ..................................................................................................... 6  
      2.1.4. Camera Placement ...................................................................................................... 8  
      2.1.5. Use of Alternate Devices ............................................................................................. 8  
      2.1.6. Project Breakdown ...................................................................................................... 8  
   2.2. **IMPORTING SKANECK MODEL INTO UNITY** ......................................................................... 8  
      2.2.1. Preparing the Skanect Model ..................................................................................... 9  
      2.2.2. Processing the Model in MeshLab .............................................................................. 9  
      2.2.3. Importing the Model into Unity .................................................................................. 9  
   2.3. **IMPORTING FROM SKETCHUP TO UNITY** ......................................................................... 10  
   2.4. **MAKING A CITY** ................................................................................................................. 14  
   2.5. **APPLYING CODE TO OBJECTS WITHIN UNITY** .............................................................. 19  
      2.5.1. Traffic Light ............................................................................................................... 19  
      2.5.2. Basic Steering ............................................................................................................. 21
1. CHALLENGE PROBLEM: CREATING A VIRTUAL ENVIRONMENT

This challenge problem is focused on the development of high fidelity virtual models of complete environments (e.g. buildings, rooms, or other specific areas) formatted to facilitate import into a game engine. Modeling and Simulation professionals often utilize commercial off-the-shelf (COTS) games to provide a basis for the virtual environment of a training or simulation. Utilizing existing games and gaming tools reduces the economic investment required to provide a customized learning environment and therefore moves the business or industry forward without a large expenditure in the development of the simulation. A game-based simulation provides a fun and creative learning opportunity for users, making learning more enjoyable and increasing user interest. Training in the form of a serious game has the capability to expand a learner’s knowledge and retention of material, as well as increase their confidence for performing a task.

1.1. THE TOOLS

Tools such as Unity, SketchUp, 3ds Max®, and MonoDevelop could be used to create this challenge solution. Many students will have had little to no experience in using these tools, so tutorials are strongly recommended. A very good source to learn coding basics is codecademy.com. Unity script reference will also be an invaluable source while creating the codes for the simulation.

1.2. THE CHALLENGE

Students are challenged to identify a real world task relevant to peers at their own school and design, develop, and test a virtual training or learning environment to address that need. This simulation should be designed to increase knowledge, improve performance, or increase efficiency while engaging users in an interactive environment.
2. TUTORIALS

Wright Scholars, in collaboration with educators and the GRILL™ team, created the tutorials described below as possible solutions to solve the challenge problem. At the time of creation these were working tutorials; however, with software updates and changes in technology, additional steps may be required. Teachers are encouraged to communicate any issues, problems, or suggested changes to these tutorials to ensure the dissemination of helpful materials to support challenge problem implementation.

2.1. DESIGNING AN EFFECTIVE GAME SIMULATION

This tutorial is designed to aid in the creation of a game-based simulation, including planning, designing, implementing, and evaluating. Suggested tools include SketchUp, CityEngine, and Unity. An outline for creating a training simulation is provided with instructions meant to guide the reader through the process. Examples from the project Driving with Sunny provide an illustration of the concept highlighted in that part of the tutorial. Driving with Sunny was a simulation created to serve as an exemplar simulation in which the player can travel through a city by car to practice basic driving skills prior to experience on a real road.

An alternative approach would be to use a model that breaks the process into phases: Analysis, Design, Development, Implementation, and Evaluation (ADDIE model).

This tutorial includes the following topics:

- Define your game (section 2.1.1)
- Design documentation (section 2.1.2)
- Game environment (section 2.1.3)
- Camera placement (section 2.1.4)
- Use of alternate devices (section 2.1.5)
- Project breakdown (section 2.1.6)

2.1.1. DEFINE YOUR GAME

The first step in creating a simulation is to determine the skills on which to focus. The goal of your game will lay the foundation for the remainder of the creation process, so it is essential that sufficient time is spent on this step. The goal of Driving with Sunny is to give an understanding of some driving fundamentals by maneuvering a car in a virtual city. This simulation provides driving experience prior to teenagers first time driving an actual car, a need that was identified through stakeholder interviews.
After identifying several ideas for a game topic, the next step would involve conducting research to identify similar games or simulations that are currently available. Once existing simulations are identified, conduct an analysis of the game(s) to identify positive and negative features and ways to improve these features. If a variety of similar simulations already exist, searching for a more original idea would be recommended.

2.1.2. DESIGN DOCUMENTATION

Once the simulation topic has been identified, you can create an outline, including features such as the main menu, sounds, perspective, view during gameplay, etc. Creating this outline will require consideration of all aspects of the game and create a place that can be referenced throughout the process. A great way to go about creating the outline is to imagine playing the game and note every detail. Changes in the outline throughout the creation process are extremely likely. The next page is a version of the design document created for Driving with Sunny.

Driving with Sunny

1. Overview

This game is designed to be used as an instructing tool to teach teenagers how to drive. As they drive through the simulated city, they will develop a rudimentary understanding of some traffic laws and driving techniques. The inspiration is a mixture of driving lesson games and regular driving games.

2. Game Elements

- **Main menu**
  - Title
  - Audio on/off
  - Start button
  - Weather conditions can be selected.

- **City**
  - Streetlights
  - Stop Signs
  - Stop Lights
  - Street signs

- **Car**
  - The player will be able to drive a car around the city, ideally from a first person view.

3. Game Mechanics

- **Elements in the game** (Each will give unique options that pertain to that situation.)
  - Game mode
  - Vehicle
  - Driving conditions

- **Car**
  - Steering (WILL TAKE lots of code)
+ Using a gaming steering wheel connected to the computer
+ Option for using a keyboard if steering wheel isn’t available
+ When the real steering wheel is turned, the game responds.

- **Stop Lights**
  - To simplify the code, all the traffic lights will run on the same code (meaning all the traffic lights within the city will be the same color at once) but it does not matter as the player is the only car driving and can follow the lights as necessary.
  - The traffic lights will be modeled in SketchUp and then imported into Unity.

- **Street Signs**
  - Stop signs, yield signs, and speed limit signs will also need to be modeled and imported into the game.

4. Aesthetics

- **City Design**
  The city is designed to emulate a large downtown area such as Chicago. The hope is to design it with as much detail as possible so its appearance is realistic. The city is a cluster of large buildings and roadways. There should be basic city attributes, such as trees along side streets, signs, stop lights, and street lights.

- **Car Design**
  The car is designed to look as much like the real thing as possible. The car tutorial from the Unity Assets Store does a very good job of making a realistic car. From the gaming window, you will be able to see the hood of the car and the dash board.

### 2.1.3. GAME ENVIRONMENT

The groundwork of the game environment will be based on the Unity game engine. This game engine was selected because of its user friendliness, importing capabilities, available resources, and user interface.

When creating a game, one of the most important aspects is the environment. The environment can be defined as the setting in which the game takes place. All great games start with a solid environment. When planning the game environment it is important to make a few up front decisions such as:

- **What type or style of environment is appropriate for the game?** For example, if making a driving game, a forest area may not make sense, while a city or suburban area would.
- **Who is the target audience?** This will help discern the level of fidelity required.
- **What is the timeframe for completion?** This will help discern the level of detail that can be accomplished.
- **What are the limitations of the game engine?** This is helpful when considering if objects and textures are going to be imported.
- **Does a similar environment exist and if so, is it available for use?** If the environment is something common, such as a cityscape or forested area, look online for existing objects.
Create the environment using all available resources. The Unity Asset Store offers a variety of objects that can be used, some for free, some for a cost. Objects can also be modeled in 3D modeling software, such as SketchUp, and imported into the game engine. Objects can also be found in SketchUp’s 3D warehouse and online, saving time in the object design process. It is important to know that the more polygons required inside an object, the more processing power is required to display it. Keeping that in mind, relying on low poly objects with high resolution textures is advisable.

These two panels both have the same high quality look; however, the one on the right has a drastically higher polycount.

Another factor to consider when designing the environment would be to avoid tiling your textures. If you choose to tile a texture, use a texture that tiles well. If the seams do not align, the object will look unappealing and have a choppy quality. Check the quality of existing objects if they are being acquired from a repository, such as SketchUp.

To determine the polygon count on an object in SketchUp, click View → Face Style → Wireframe, and the object’s polygons will become viewable.
2.1.4. CAMERA PLACEMENT

The type of simulation being created will determine where to place the camera. In many cases, it is better to have a first person view as opposed to third person. The first person view was optimal for Driving with Sunny.

1. To place the camera, begin by attaching the camera to the player. In Driving with Sunny, the camera was attached to the car so that the camera would travel with the player.
2. To attach the camera, click on the main camera in the hierarchy, and drag it on top of the player until the player is highlighted, then drop it.
3. To correct the placement, find the camera in the hierarchy and then choose the Move option (four arrows pointing up, down, left, and right) in the top left-hand corner.
4. Use the arrows that appear in the scene view to move the camera to the appropriate place. Entering game view shows what the camera will look like if the simulation were actually being played.

2.1.5. USE OF ALTERNATE DEVICES

In many simulations, there are tools that will make it seem more realistic. For example, a pedal and steering wheel apparatus were used in Driving with Sunny to make the player feel as if they were actually driving. The tools you use will depend on the simulation being created as well as the cost and availability of the devices. There are a large number of devices that you could use, so conduct research to determine how it can be implemented. The steering wheel and pedals added an extra script to the Driving with Sunny game.

2.1.6. PROJECT BREAKDOWN

Breaking different aspects of the simulation into separate tasks is recommended. Each member of the group can focus on a different aspect while still being involved in other tasks. For example, tasks could be broken into modeling, coding, and game engine manipulation. Using others’ strengths within the group will allow for maximum efficiency and quality in the creation of the final product.

2.2. IMPORTING SKANECT MODEL INTO UNITY

This tutorial shows how to process and import a 3D model generated by a Kinect into the game engine Unity. Recommended tools include Skanect, MeshLab, and Unity 4.x

This tutorial includes the following topics:

- Preparing the Skanect model (Section 2.2.1)
- Processing the model in MeshLab (Section 2.2.2)
- Importing the model into Unity (Section 2.2.3)
2.2.1. PREPARING THE SKANECT MODEL

Once the Kinect scan is complete in Skanect:

1. Go to Process ➔ Geometry ➔ Fill ➔ Run to fill holes.
2. Go to Process ➔ Geometry ➔ Simplify and reduce the number of faces to 62,000 and press Run.
3. Go to Process ➔ Move & Crop ➔ Rotate and translate the model and press Crop to Ground to get rid of any excess material that may have been scanned.
4. Go to Process ➔ Color ➔ Colorize ➔ Run to add color to the model.
5. Go to Share ➔ Local ➔ Save (save the file somewhere easy to locate, such as a folder on the desktop).
6. Go to Share ➔ Local ➔ Export Model ➔ Export (or with the free version, Share ➔ Local ➔ Export Preview ➔ Export) and export as an .obj file.

2.2.2. PROCESSING THE MODEL IN MESHLAB

In MeshLab:

1. Open Meshlab
2. Go to File > Import Mesh and open the .obj file previously saved.
3. Go to Filters > Texture > Parametrization: Trivial Per-Triangle > Apply. If an error appears, make sure the number of faces is 62,000 or fewer.
4. Go to Filters > Texture > Transfer Vertex Attributes to Texture (between 2 meshes). Make sure the Assign Texture box is checked. This will save a .png file, which you can use in Unity to give the object color, in the same place as the .obj file.
5. Go to File > Export Mesh as.... Change the .ply to a .obj extension and save the mesh. This saved file can replace the previously created .obj file.

2.2.3. IMPORTING THE MODEL INTO UNITY

The picture below provides an overview of the Unity window.
1. Open a project in Unity by going to File > Open Project and either selecting an existing project or moving to the Create New Project tab and selecting Create.

2. Go to Assets > Import New Asset and locate the .obj file. Click Import to import the file.

3. Go to Assets > Import New Asset, locate the .png file that is paired with the .obj file and click Import.

4. In the Project tab, click on the Assets folder, select the imported .obj file, and drag it into the Hierarchy tab. You may need to translate the object or rotate it.

5. In the Hierarchy tab, find the object and click the drop down arrow. Expand the file as much as possible. There should be either multiple default_MeshParts or a single default part.

6. Find the color.png file in the Assets folder. Drag this file into each of the default_MeshParts or the default part to color the object.

7. Double click the object in the Hierarchy tab to find the object in the Scene window.

**2.3. IMPORTING FROM SKETCHUP TO UNITY**

This tutorial describes how to import a model created in SketchUp into the game engine Unity. Recommended tools for this tutorial are Skanect, MeshLab, Unity 4.x.

1. Open Unity, create and save a new project.

2. Create or open a model in SketchUp, select the entire model, right click, and select **Make Component**.
3. Export the model by selecting **File > Export > 3D Model**.

4. Save the model file in the project file that you created for the Unity project in the **Assets** folder. Be sure to make the file name relevant to the model and save the file as a **COLLADA** file (.dae). Click **Export**. Repeat this for each model that you create for the simulation.

   **Note:** Since game engines vary, you may need to look online for the specific import function.
5. Once you have imported all models into the game engine, it is time to design the models. If the models need additional specific components such as rigid bodies and code functions, add the components before placing the models in the environment. Some general guidelines to follow when creating the environment include:
   a. Do not over use a single model. Providing variety in the landscape is visually appealing to the user.
   b. Applying soft shadows can help make the environment visually appealing and realistic.
   c. When applying particle effects, make sure to balance the lifespan of the particle with the amount emitted. If the Lifespan is too long, the processing power will increase which results in a slower load time.
   d. Try to avoid using a large amount of downloaded objects. The quality of objects will vary a lot and the overall quality of the environment will diminish.
   e. Try to keep ambient sounds to a minimum and make them proper to the environment.

6. To place the imported objects into the game, open the project in Unity and locate the model in the Assets folder in the Project window. Select the object and drag it into the empty Scene window. It should display in the hierarchy window.
7. Make the object a prefab by adding physics to it. Depending on the shape of the object, either a Mesh Collider or Box Collider will provide the necessary physics. A box shape to surround the object will work with most objects and is recommended as mesh colliders are more expensive computationally. To add a collider:
   a. Select the model.
   b. Go to Component > Physics > Box Collider or Mesh Collider.

If you choose Mesh Collider, then the object will automatically fit to the shape of the object. If you choose Box Collider you will need to alter the size to fit the object.

**Note:** Wherever a collider is, other objects will not be able to pass through. For example, if a mesh collider is applied to the stop sign, a car will not be able to drive through it. If, instead, a box collider is used, a car could not drive through any part enclosed in the box.

8. Alter the size of the object by using the adjustments for the collider located on the right of the screen when you select the object in the hierarchy. Once you have made the adjustments, the object is considered a prefab.

9. Create a separate folder called Models under Assets where all prefabs are located.

10. Put the models originally imported in a folder called Source Models to help distinguish between models with and without physics. Whenever you need another instance of that model, you can then pull it out of the prefab folder and place it in its desired position without needing to reapply colliders or other physics.
**Note:** When making changes to a prefab, be sure to click Apply in the upper right corner to make sure the changes apply to all the versions of that prefab in the game.

2.4. MAKING A CITY

This tutorial will guide you through the process of creating a city using CityEngine. CityEngine is a procedural model generator, meaning that it will create everything based upon set parameters. This style is good for a large amount of objects that need to be uniform, but at the same time unique. CityEngine comes with some built in examples that you can use to build your city.

1. Open CityEngine. When the program starts up, click the *Launch* button, and you should be presented with a screen similar to the screen below:
2. Click the Next button, and continue to the 3D Navigation Screen. If you are familiar with the controls of any of the software listed, then use those controls, otherwise, take note of the controls provided.

3. Click the Next button, pass the resources and example projects, they are not needed. Name your city.

4. Choose the City Size, I would recommend small, if needed medium. The larger the city size, the more latency will be produced.

5. Click the Next button. On the Terrain screen, you may assign your own height map and textures, if you do not have a personal height map, leave everything alone, and click the Next button.
6. Click the **Next** button. The street styles are inside the red box. Pick the street map you prefer.

7. Click the **Next** button. Choose the style and texture of the buildings you want in your city.
8. Click the *Finish* button. At this point, CityEngine will take some time to generate your city. When it finishes, a screen displays that is similar to the one below. Congratulations, you just created your city!

![Image of CityEngine interface after generation]

9. Next, you will need to smooth out the texture. Using your view controls, zoom in to the city; you may notice that many of the textures look out of place. This can be fixed fairly easily. Select the bad texture, and press *Control + Shift + G*. This will generate a new texture. Keep pressing *Control + Shift + G* until you are satisfied with the texture.

![Image of selecting and adjusting texture in CityEngine]

10. To export the city, click Control-A to select all items in your city, then click *File -> Export Models*. 

---

CREATE A VIRTUAL ENVIRONMENT

Distribution A.
11. Click the *Collada File* type, click *Next*, and click *Finish*.
12. After you finish exporting, open Unity and start a project. Drag your city’s model folder into your assets.
2.5. APPLYING CODE TO OBJECTS WITHIN UNITY

In Unity, code can be compiled into Scripts located under Favorites in the project hierarchy. The possibilities with coding in Unity are seemingly endless. There is an abundant amount of help available online for coding, as well. Some of these examples are outlined in the resources section. This tutorial will guide you through the process of applying code to a traffic light and a car in Unity. Recommended software includes SketchUp and Unity 4.x.

This tutorial includes the following topics:

- Traffic light (Section 2.5.1)
- Basic steering (Section 2.5.2)

### 2.5.1. TRAFFIC LIGHT

In Driving with Sunny, a traffic light was created using SketchUp. It was imported into Unity. Colored lights, called point lights, were added where each light in the traffic signal is located (red on top, yellow in the middle, green on bottom).

The three point lights should alternate being on rather than all being on at once. For this to work, you will need to create a script. The next page contains code (C-sharp, C#) in red and comments in black.

1. To copy and paste the code into a script and attach it to the light:
2. Right-click in the Assets pane at the bottom and select *Create -> C# Script.*
3. Assign each point light in the Inspector window by dragging its name from the hierarchy to the spot for point lights under the Script.

```csharp
using UnityEngine;
using System.Collections;
public class TrafficLightControl : MonoBehaviour{
    public Light[] lights;
    public enum LightState {
        green=0,
        yellow=1,  //THIS PART CREATES THE SPACE FOR THE LIGHTS TO BE ASSIGNED
        red=2,
    }
    public float timer = 0;
    LightState state;
    // Use this for initialization
    void Start () {
        timer=0;
        TrafficLight ();
        state= LightState.green;
    }
    // Update is called once per frame
    void Update () {
        TrafficLight ();
        timer+=Time.deltaTime;
        // THIS SETS GREEN TIME TO 15s
        if (state == LightState.green && timer>=15) {
            timer = 0;
            state = LightState.yellow;
        }
        // THIS SETS YELLOW TIME TO 5s
        if (state == LightState.yellow && timer>= 5){
            timer = 0;
            state = LightState.red;
        }
        // THIS SETS RED TIME TO 15s
        if (state == LightState.red && timer >=15){
            timer= 0;
            state = LightState.green;
        }
    }
    void TrafficLight () {
        switch (state){
            case LightState.green:
                lights[0].enabled=true;  //R and Y off, G on
                lights[1].enabled=false;
                lights[2].enabled=false;
                break;
            case LightState.yellow:
                lights[0].enabled=false; //G and R off, Y on
                lights[1].enabled=true;
                lights[2].enabled=false;
                break;
            case LightState.red:
                lights[0].enabled=false; //G and Y off, R on
                lights[1].enabled=false;
                lights[2].enabled=true;
                break;
        }
    }
}
```
The traffic light will alternate on a cycle of 15 seconds of green, 5 seconds of yellow, and 15 seconds of red. You can alter those values by changing the appropriate values in the code shown above.

### 2.5.2. BASIC STEERING

The following tutorial provides a simplified way to create a car within a game engine with realistic physical forces.

1. Create a cube under *GameObject* within Unity:
2. Create a box collider that fits the cube as shown earlier in the tutorial with the stop sign. Also, make the cube a RigidBody by clicking on `Rigidbody` under `Component -> Physics RigidBody`.

3. Create four wheel colliders (`Component -> Physics -> Wheel Collider`). Name them as follows: WheelFL, WheelFR, WheelRL, and WheelRR (signifies front left, front right, rear left and rear right). Position the colliders around the box where wheels would be located.

Above you see wheel colliders placed on a box collider. The body of the car is a physical representation. The circles are the wheel colliders placed where wheels are located.

4. The following script (C#) enables the car to steer. You can directly copy the code in red into Unity, comments are in black.

```csharp
using UnityEngine;
using System.Collections;
public class Car : MonoBehaviour {
    public WheelCollider WheelFL;     //THESE LINES CREATE VARIABLES
    public WheelCollider WheelFR;     //FOR THE WHEEL COLLIDERS THAT
    public WheelCollider WheelRL;     //WILL BE ASSIGNED IN THE
    public WheelCollider WheelRR;     //INSPECTOR WINDOW
    float Speed = 10;   //THESE VALUES CAN BE ADJUSTED AS NECESSARY
    float Braking = 20;
    float Turning = 20;
    // Update is called once per frame
    void Update () {
        //Makes car go forward
        WheelRR.motorTorque = Input.GetAxis("Vertical") * Speed;
        WheelRL.motorTorque = Input.GetAxis("Vertical") * Speed;
        WheelRL.brakeTorque = 0;
        WheelRR.brakeTorque = 0;
        //Makes car turn
        WheelFR.steerAngle = -1 * Input.GetAxis("Horizontal") * Turning;
        WheelFL.steerAngle = -1 * Input.GetAxis("Horizontal") * Turning;
        //Makes car brake
        if (Input.GetKey(KeyCode.Space)){
        }
    }
}
```
After you attach the script to the cube or body of the car, the wheel colliders will be assigned in the Inspector window (see image below).

5. You can drive the cube around as if it were a car with wheels. You can also get the body of the car, similar to the one shown above, from the SketchUp warehouse and import it into Unity.