

DESIGNING AND BUILDING A THREE-DIMENSIONAL  
ENVIRONMENT USING BLENDER 3D AND UNITY GAME  
ENGINE

Sheshina Evgeniya

Thesis  
Lapland University of Applied Sciences  
Degree Programme In Information Technology  
Bachelor of Engineering

2016

Abstract of Thesis

---

|                        |   |      |      |
|------------------------|---|------|------|
| <b>Author</b>          | Evgeniya Sheshina   | Year | 2016 |
| <b>Supervisor</b>      | Veikko Keränen  |      |      |
| <b>Commissioned by</b> | pLAB  |      |      |
| <b>Title of Thesis</b> | Designing and Building a Three-dimensional Environment Using Blender 3D and Unity Game Engine |      |      |
| <b>Number of pages</b> | 39 + 6  |      |      |

---

This thesis was divided into two parts. The first part introduced and explained 3D modelling terminology. Moreover, the software that was used to build the environment was covered. The second part focused on the implementation of the project.

First a floor plan of the environment was drawn, and a list of models was created. Then the models were made using Blender 3D. The process of making two of the models was documented. After the models were created, they were imported into Unity for testing, which was an iterative process.

After the planning, modelling and testing were completed, the environment was built using the previously created and tested models. A small amount of coding was done to implement animation of several models, and interaction that activated the animation. Lighting was added in Unity as well.

Finally, the finished environment was delivered to the commissioner. As a backup, the Unity project was brought as well. Moreover, all the models and textures created were delivered as well.

# CONTENTS

|       |  |    |
|-------|--|----|
| 1     | INTRODUCTION.....  | 5  |
| 2     | 3D MODELING CONCEPTS OVERVIEW.....                             | 6  |
| 2.1   | 3D Modeling.....   | 6  |
| 2.2   | 3D Models and Modeling Techniques.....                         | 6  |
| 2.2.1 | Elements of a Polygonal 3D Model.....                          | 7  |
| 2.2.2 | Shaders, Normals and Maps.....                                 | 8  |
| 2.2.3 | Rendering.....   | 10 |
| 2.2.4 | Rigging and Animation.....                                     | 11 |
| 2.3   | Game Engines.....  | 11 |
| 3     | SOFTWARE USED IN THE PROJECT.....                              | 13 |
| 3.1   | Blender 3D.....  | 13 |
| 3.1.1 | Blender UI and Tools.....                                      | 14 |
| 3.1.2 | Blender Modes.....   | 15 |
| 3.2   | Unity 3D.....  | 16 |
| 3.2.1 | Unity Terms.....   | 16 |
| 3.2.2 | Unity Interface.....   | 17 |
| 3.2.3 | Unity Physics Engine.....                                      | 18 |
| 3.2.4 | Animation in Unity .....                                       | 19 |
| 3.2.5 | Differences Between Blender and Unity Coordinate Systems.....  | 20 |
| 3.3   | Adobe Photoshop CS6.....                                       | 20 |
| 3.4   | Room Sketcher.....   | 21 |
| 3.4.1 | ENVI Platform.....   | 21 |
| 4     | BUILDING AN APARTMENT IN BLENDER 3D AND UNITY.....             | 23 |
| 4.1   | Creating a Floor Plan.....                                     | 23 |
| 4.2   | Model Creation.....  | 24 |
| 4.2.1 | Model Creation: Television Stand.....                          | 25 |
| 4.2.2 | Model Creation: Walls and Floors.....                          | 28 |
| 4.3   | Testing in Unity.....  | 29 |
| 4.3.1 | Preparing a Model for Export to a Unity-Compatible Format..... | 30 |
| 4.3.2 | Importing Separate Models into Unity.....                      | 30 |
| 4.4   | Building the Apartment.....                                    | 31 |
| 4.4.1 | First Person Controller, Animation and Interaction.....        | 31 |
| 4.4.2 | Physics and Lighting.....                                      | 32 |
| 4.4.3 | Delivery of the Finished Environment.....                      | 33 |

|  |    |
|--|----|
| 5 CONCLUSIONS.....                           | 34 |
| 6 DISCUSSION.....                            | 35 |
| 6.1 Reliability, Usefulness and Purpose..... | 35 |

## 1 INTRODUCTION

3D modeling is extremely spread. Therefore, skills needed to create and modify 3D models are in demand in multiple industries, such as game development and 3D printing. As a result, 3D modeling skills open many opportunities. That is why a 3D modeling topic was chosen.

This work consists of two parts: the theory part and the implementation part. An overview of 3D modeling concepts is provided in the first part, as well as descriptions of the programs and tools that are used throughout this thesis. The second part follows the planning and creation of the apartment. Due to a high number of 3D models required, not every single one of them is followed through its creation.

The result of the thesis is a 3D environment built in Unity using 3D models created for it. The 3D environment is delivered to pLAB at Lapland University of Applied Sciences, where it is used as a part of education process of the students.

## 2 3D MODELING CONCEPTS OVERVIEW

### 2.1 3D Modeling

3D modeling is the process of creating digital models of objects using specialized software. It is extensively employed in a multitude of areas ranging from entertainment, such as games and films, to education, art, construction and medicine. Moreover, new uses of 3D modeling are being discovered.

Game and film industries are probably the most common and well-known examples of 3D modeling use. The majority of games released nowadays are 3D, featuring 3D characters and environments. Those models have become more realistic, complex, and detailed over time. Film industry takes the use of 3D modeling a step further, using modeling to add special effects or create environments. (Jonpolygon 2016.)

Computer Aided Design software (CAD) was used by architects for decades, making it another common application of 3D modeling. It has assisted greatly in creating floor plans, drafts, and models of finished products, as well as presenting them. Moreover, CAD software is precise and therefore lowers the risk of human errors and the time required to fix them. (Rasmussen 2013.)

Lastly, 3D printing is an emerging and rapidly growing industry with 3D modeling at its core. Uses of 3D printing are extremely diverse. Aircraft and spacecraft industries make use of 3D printing to accelerate and improve the process of building of spacecraft and aircraft parts. Bio printing is an area of medicine where 3D printing is harnessed to create replacement organs and tissue (3D Printing.com 2016.). Moreover, 3D printing is also used privately to print decorative items or replacement parts for household items. More uses of 3D printing will likely be discovered in the future.

### 2.2 3D Models and Modeling Techniques

A 3D model is a digital representation of a three-dimensional object. A 3D model can represent any type of object, real or imagined. 3D models are

created, viewed and modified using specialized software referred to as 3D modeling software.

There are three most common types of 3D models: polygon models, NURB curves and surfaces, and models created via digital sculpting. Polygon models and NURB curves are the most prominent types of models. Digital sculpting is an emerging type of 3D models with its own modeling techniques. However, it is usually tied with polygonal modeling.

Polygon or mesh models are the most common type of 3D models, widely used in entertainment such as games and animation. Polygon models consist of a large number of flat surfaces, called faces or polygons, that are connected at an angle to form the desired shape. The more surfaces there is in a model, the more detailed and smooth it will appear. This type of models is what will be created and discussed during the implementation stage.

Non-Uniform Rational B-Splines (NURB or NURBS) curves are mathematically defined and have several advantages over regular polygon models. First, they can be tailored to represent almost any shape from simple lines to complex curves (Schneider 1996.). Second, as NURB curves are determined mathematically, they can be scaled without the shape becoming distorted. Third, NURB curves allow for more precision than polygon modeling. This makes them useful in industries like architecture, design, and construction.

An emerging technique, digital sculpting, as argued by Slick (2014) is a disruptive technology meaning it has drastically changed how some aspects of 3D modeling are approached. Digital sculpting is a process of creating a 3D model that is similar to working with a real world medium like clay.

### 2.2.1 Elements of a Polygonal 3D Model

A polygonal 3D model consists of three key elements: vertices, edges and faces. Vertices are simply points in 3D space that possess three coordinates: height, width and depth, that can be manipulated to place the vertex at the desired location and, consequently, alter the model's shape. Edges are lines

that connect two vertices in 3D space. Three or more vertices that are connected by edges form a face or a polygon. Together, those elements form the surface of the model, also known as the mesh, or geometry.

While a face can have three or more sides, three and four-sided faces are most common. Although three-sided polygons or triangles can be a cause of problems in modeling (Digital-tutors 2014a.), most 3D modeling programs have a function to convert four-sided faces to three-sided faces. Four-sided faces or squares are the standard in 3D modeling and are the default polygon size in 3D modeling applications. Squares are the simplest to work with when considering future animation and texturing.

Polycount refers to the amount of polygons that build the surface of the model. A high number of polygons will provide smooth surfaces, high level of detail, and ensure that any subsequent animation will look as natural as possible and no distortions occur. However, a model with a high polycount usually takes more time to render. Moreover, high polycount models are difficult to modify. As a result, polycount is an important factor to consider in game or other interactive media development. (Silverman 2013.)

### 2.2.2 Shaders, Normals and Maps

Shaders, or materials, as they are called inside Blender software, are sets of parameters that affect the appearance of the model's surface. Examples of those parameters include diffuse, shading, specular, reflectiveness, and transparency. Diffuse is the base color of the material. Shading is how the material will interact with the light cast on it and how the shadow will be rendered on it. Specular determines the presence and amount of shine on the material. Reflectiveness is responsible for whether the material will reflect the light and surroundings. Lastly, transparency determines whether the light will pass through the material. All these settings can be configured and combined to simulate the look of various materials, such as metal, glass or clay.

Normals are imaginary lines perpendicular to a surface of a polygon. They determine which side of the polygon is the front side (Digital-tutors 2014b.). The



position of the normals is an important detail to consider, since a polygon will usually be invisible if viewed or rendered from the back side.

Maps are image files applied to the model to further change its appearance. There are various types of maps used in 3D modeling: UV or texture maps, normal maps, transparency maps, displacement maps, emission maps, and specular maps. Models created throughout the thesis will use only two kinds of maps: UV maps, and normal maps.

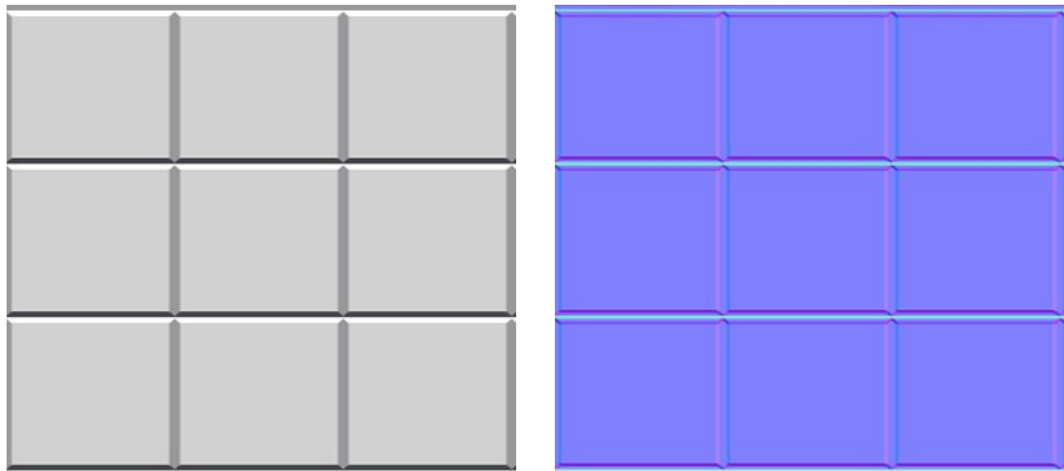


Figure 1. UV Map (Left) And A Corresponding Normal Map (Right)

UV maps or textures (Figure 1, left.) are images that alter the coloring of the model, similar to how paint is applied to a wall. The process of mapping a 2D image to a 3D model is called texture or UV mapping. When mapping a model, edges play an important role as they can be used to delimit the seams on the UV map. Seams tell the 3D modeling software where to “cut” the model when projecting it onto a 2D plane.

Normal and bump maps are images that, when applied to a 3D model, simulate relief. In polygonal modeling they are widely used to add detail to a model without affecting its polycount. Bump maps use grayscale colors to simulate the changes of height of a surface. Normal maps use different values of green, blue and red that translate to coordinates  $x$ ,  $y$ , and  $z$ , affecting the surface's appearance when viewed from different angles (Hajjoannou 2013.). As a result,

a surface with a normal map applied will appear detailed, even though it is actually flat (Figure 1, right). Normal maps provide greater detail than bump maps.

Transparency or alpha maps determine the opacity of a model by interpreting black and white values of an image. Black usually corresponds to complete transparency while white typically makes the surface opaque. A typical example of transparency map use are things like grass, fur, and leaves. The use of an alpha map in such cases saves a significant amount of time and resources.

Other kinds of maps include displacement maps, emission maps, and specular maps. Displacement maps work in a similar fashion to normal maps. However, they alter the model's geometry, while normal maps do not. Emission maps are image files that control glow on the model's surface. Specular maps control shine.

Several kinds of maps can be applied to a model simultaneously. For example, a combination of UV, normal and transparency maps can give a model consisting of a single plane an appearance of a carpet. There is a variety of other ways to combine maps to give the model the desired appearance.

### 2.2.3 Rendering

Rendering is the process of creating a 2D representation of a 3D model. 3D modeling application render and display models on the screen as they are modified or viewed. Games render the game world and the characters in it in real time. Additionally, rendering often refers to images of 3D models created by 3D modeling programs or specialized rendering software.

Rendering time is an important factor in 3D graphics. On the one hand, rendering time eventually does not affect the viewers of films and animation incorporating 3D models because they are viewing a recording. On the other hand, rendering time should be kept as low as possible in games and other interactive media where the models are rendered in real time. (Silverman 2013.)

#### 2.2.4 Rigging and Animation

Creating a 3D model is often merely a first step in the process of building a game or a film. If 3D modeling is used to create a film, the models are animated and then recorded. In games, characters and objects are animated as well, but the animation generally plays in response to the player's actions.

Before a model is animated, a structure similar to a skeleton called a rig is created. A rig is a construct similar to a human skeleton that consists of joints and bones. Sometimes the terms "joints" and "bones" are used interchangeably. The rig is placed inside the model, with joints corresponding to parts of the model that are bent. The rig is invisible when the model and its animation are viewed inside a game or film. (Masters 2013.)

Joints act in a similar way to joints in the human body, allowing the model to be bent at the points where the joints are located. However, to make the joints affect the 3D model, the rig must be bound to it. The process of connecting the model to its bones is referred to as skinning. A part of the skinning process is determining how much influence a joint has over the model's vertices which is referred to as weight assigning. For a model to look natural when it is animated, weight should be smoothly distributed between joints. (Masters 2013.)

#### 2.3 Game Engines

A game engine is an environment where a game is built. Often the software is tested inside game engine before it is complete enough to undergo independent testing by external users or other developers. Game engines consist of several groups of tools for editing various aspects of a game. Those groups of tools are generally the renderer, physics, scripts and AI.

The renderer is responsible for the graphic output. In other words, the renderer displays all the objects inside the game so that the developer or user is able to see them (ExtremeTech 2002.). Different game engines employ different technologies to provide graphical output. As a result, each game engine has its own visual style, and several games created using the same engine often share

that style despite being otherwise different, which, according to Enger (2013), can be viewed as a drawback.

Physics in game engines simulate physics of the real world. Physics engines use mathematical representation of laws of physics to ensure that objects in the game behave realistically. However, according to Enger (2013), physics are not a part of the game world from the beginning, and have to be added manually by the developer.

Scripts are sets of code snippets that store common behaviors for game objects, such as player input detection and animation. Scripts are also referred to as behaviors or modules, depending on the game engine (Enger 2013.). They greatly simplify the coding process, providing basic building blocks from which a developer can construct complex behavior. However, the drawback is that it is difficult to extend the predetermined behaviors in case a more specific approach is needed.

### 3 SOFTWARE USED IN THE PROJECT

#### 3.1 Blender 3D

The thesis project will employ several programs to create and test models: Blender 3D, Unity, Adobe Photoshop, and Room Sketcher. Blender 3D was used for modeling. Unity game engine was used for testing and actual implementation. Room Sketcher was used to create the plan of the apartment that was built. Adobe Photoshop was employed to create and modify textures.

Blender 3D is an open source 3D modeling and animation software package. Its initial development started in 1995 as a company's own 3D toolset. Moreover, initially Blender was supposed to have both free and commercial versions. However, because of low sales, Not a Number, the company behind the development, was shut down in 2002 (Blender Foundation 2016c.). In an effort to keep the project alive, the community that had formed around Blender pushed for it to become open source, and in October 2002 the first open source version of Blender was released.

Blender's development still continues years later, with new features being added and existing ones being refined. As a result, Blender supports a wide range of tools from basic polygon modeling to advanced rendering to animation. Moreover, experienced users can write their own extensions in Python programming language, as well as download and integrate plug-ins made by other users. (Blender Foundation 2016a.)

Being open source and free, as well as including a variety of powerful tools for a wide range of 3D modeling purposes, is Blender's main strength (Blender Foundation 2016a.). It has two main drawbacks: the fact that Blender is very reliant on the keyboard shortcuts which may be challenging to memorize, and the fact that Blender may be overwhelming for the beginners to master. However, there is a multitude of tutorials and a large quantity of learning material that can be found on the internet and can teach the new users sufficient skills to start using the software.(Tech World 2011.)

### 3.1.1 Blender UI and Tools

Blender's default interface consists of a series of windows with headers, that in turn comprise several key areas: the 3D viewport, the outliner, the properties panel and the timeline (Figure 1). Despite the name, the header can be located at the bottom of the window, as is the case with the viewport and timeline windows. (Blender Manual 2016a.)

The viewport is the primary window in Blender. By default it is located in the center of the screen and is where the model is displayed (Figure 2). There are two toolbars on the sides of the viewport, that can be toggled on or off. The content of the toolbars depends on which mode Blender is in.

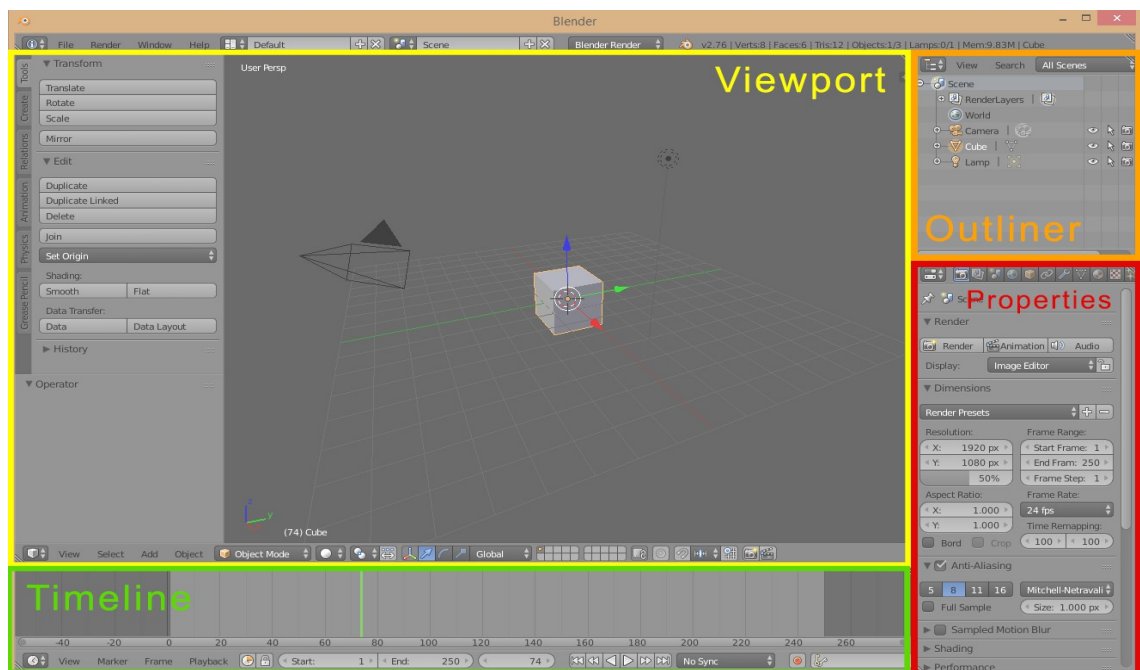


Figure 2. Blender UI

The outliner contains a hierarchical view of everything in the current Blender file. The outliner's purpose is to provide an overview of the file's contents. Moreover, models and other objects can be organized via this window. Operations like deletion, renaming, and selection can all be performed there.

Properties panel is a group of panels that contain various settings. The settings can be general, applied to the file as a whole, like rendering or view options.

Otherwise, the settings are applied to the selected object, like material and texture settings, modifiers, and constrains.

Blender's UI is also extremely flexible. It is possible to switch between displaying every kind of view or editor in any of the available panels, as well as create new windows or panels inside the UI. Moreover, Blender features several layouts for different purposes, such as animation or texture editing. New layouts can be created and saved by the users as well. Additionally, experienced users can use Python programming language to customize the interface and the rest of Blender's functionality (Blender Foundation 2016b.).

### 3.1.2 Blender Modes

Blender features a number of modeling modes. What modes are available at any given point depends on the type of object selected. Different modes enable different tools to edit the models, although some tools are available in several modes (Blender Manual 2016.). The most often used modes are object mode, which is the default mode, and edit mode, where the modeling is done.

Object mode is a type of a management mode, where the user is able to move, rotate and scale entire objects, create, change and delete material settings, as well as create and delete entire objects. Object mode is the only mode where different objects can be selected (BlenderWiki 2016.). In other modes, it is impossible to select any other object except the one currently selected.

Edit mode is where the modeling takes place. Options to select single vertices, edges or faces are present in edit mode, as well as various tools to change the model's shape. Materials can be applied to an entire model or parts of it while in edit mode.

Sculpt mode is slightly similar to edit mode as both are used to change the model's shape. However, sculpt mode enables a different set of tools and, as its name suggests, allows the user to work with a 3D model as if it was a real world medium such as clay. This mode is useful when organic or uneven shape is needed.

## 3.2 Unity 3D

Unity 3D is a popular 2D and 3D game engine used to create games and other interactive media. The development of Unity started in 2002, and the first version was released in 2005. Originally Unity was only available for Mac OS devices. Since then, Unity has undergone numerous updates that introduced new features, expanded the array of operation systems Unity can be installed on, as well as platforms programs made with Unity can be installed on. (Haas 2012.)

With an abundance of learning material both official and produced by its large community, Unity is relatively easy to learn. As a result of that and Unity being cross-platform, it is the most widespread game engine. According to the official website, about one third of mobile games created in the first quarter of 2016 were developed with Unity (Unity Technologies 2016b.).

Unity has several access levels. Personal access is free, and the free version possesses all the features needed to sufficiently develop and deploy games. However, there are some restrictions regarding the revenue generated by a game created using the free version. Moreover, the paid versions include collaboration tools and several other features, such as statistics tools and advertisement integration. (Unity Technologies 2016a.)

### 3.2.1 Unity Terms

A scene is the first thing created in a Unity project. It is required to have at least one scene in which assets are placed. A single Unity project can have several scenes that could correspond to different locations or levels in a game, for instance. Individual scenes can be viewed and edited separately within a project. (Unity Manual 2016e.)

A Game Object is a term referring to basically every item within a scene. On their own, Game Objects do not possess any functionality. However, they act as containers for Components, which are various properties that define what type



of Game Object is created. There are several types of Game Objects with predetermined sets of components to ease the creation of often used items such as lights, sounds, cameras, 3D primitives, and imported 3D models.(Unity Manual 2016c.)

### 3.2.2 Unity Interface

By default, Unity's UI is divided into 5 main areas: The Scene View, the Hierarchy, the Game View, the Project panel and the Inspector panel. However, the areas can be rearranged and scaled to fit the user's preferences. Additional panels and windows can also be opened via the Window menu. (Unity Technologies 2016c.)

In the upper left part of the screen (Figure 3) is the Scene View. Scene View is essentially where the game is built. 3D or 2D assets can be placed and manipulated in this view, and there are a number of controls that can simplify navigation within the scene.

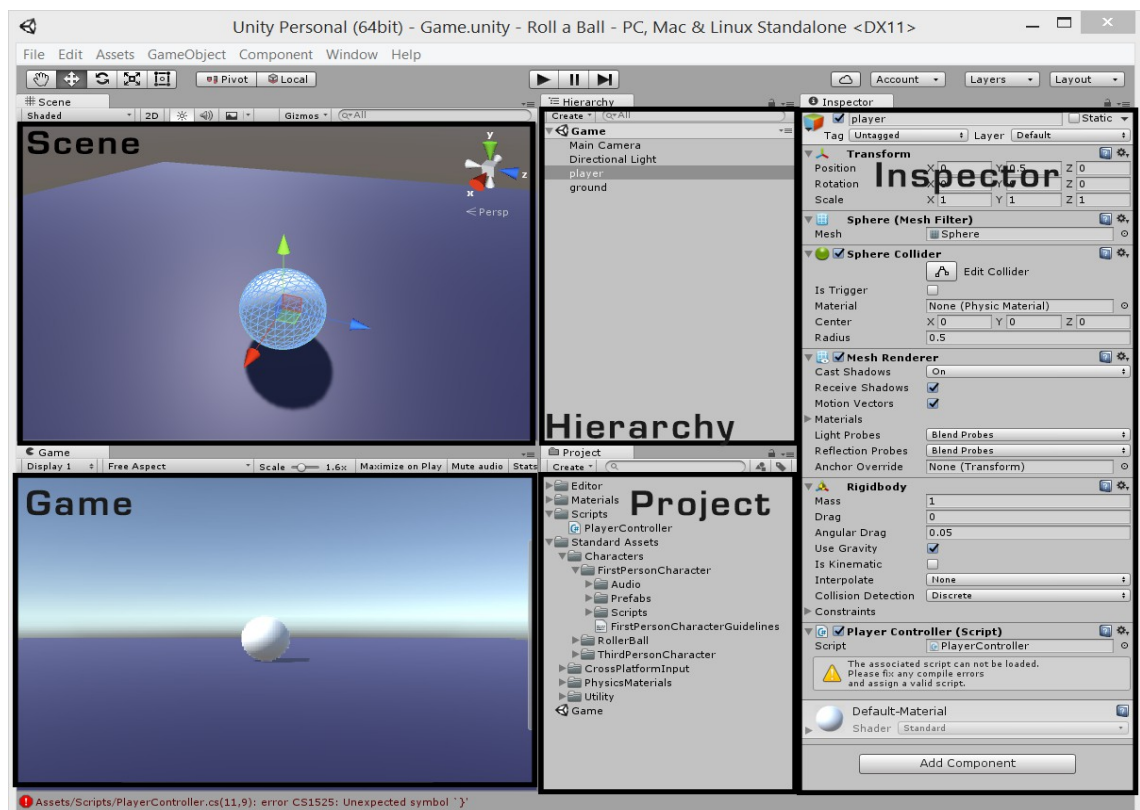


Figure 3. Unity's UI

Game View or play mode is usually located below the Scene View. It displays the contents of the scene from the camera's point of view. It is activated by pressing the play button and is where the game can be played and tested at any point of the development process.

The Hierarchy View lists all the objects in the scene, in alphabetical and hierarchical order. It is located to the right of the Scene View, though its location may vary depending on the version of Unity. Aside from listing all the object, the hierarchy is one of the means that can be used to create new objects. Moreover, objects can be rearranged and sorted using this view.

The Project Panel lists all the assets that were imported into the project, allowing for quick access to them. By default, it is situated to the right of the Game View. Similarly to the Hierarchy View, assets can be rearranged and sorted into folders if necessary. Project Panel will also automatically display the contents of the current project's folder and update them if a file was added or removed.

The Inspector Panel is located to the right of the screen. It is context-sensitive, meaning it displays the properties of an object currently selected. Inspector panel allows adding properties to objects. It is also where the existing properties are edited, including public variables of any attached script. (Unity Technologies 2016c.)

### 3.2.3 Unity Physics Engine

There are tools for physics simulation built into Unity, ensuring that the objects inside the game behave in a believable fashion. Two physics features will be employed in this project: colliders and Rigidbody components. There are other physics-related tools inside Unity, but the rigidbodies and colliders are the basic ones.

A Rigidbody is the main Unity object component that enables physics behavior. An object with a Rigidbody attached is subject to gravity(Unity Manual 2016d.). However, a Rigidbody alone is not enough to achieve the full spectrum of

physics simulation, so a collider is usually necessary to give the object an ability to interact with other objects in the scene.

A collider is a component that defines the shape of an object for the purposes of physics. A collider is not necessarily the exact same shape as the object's mesh (Unity Manual 2016b.). There are several types of primitives that the colliders can take shape of: box, sphere and capsule. However, it is possible to have a collider to be set to have the same shape as a 3D model. This type of collider is referred to as a mesh collider.

One sub-type of a collider is a static collider. A static collider is an object that has a collider component, but not a rigidbody component. Static colliders are usually objects in the game or other 3D environment that do not require to be moved, such as walls and floors. (Unity Manual 2016b.)

#### 3.2.4 Animation in Unity

Unity offers tools for animation among its features. It is possible to animate models within Unity itself, or import animation clips made in other software (Unity Manual 2016a.). However, Unity's animation capabilities are currently limited to playing imported animation and creating simple animation, like rotation, scale, or movement. More complex animation must be created in other software.

With Unity's scripting capabilities, it is possible to link the animations to various events and triggers within the project. For example, doors opening when the player character approaches them, or character running when the player uses the movement controls. Additionally, simple animation such as rotation can be made using code alone, as an object's properties such as scale or, in this case, rotation, can be accessed inside the code.

### 3.2.5 Differences Between Blender and Unity Coordinate Systems

All 3D modeling programs rely on coordinate systems to determine the location of each vertex of the 3D model. However, coordinate systems sometimes differ between programs. This is the case with Blender and Unity 3D.

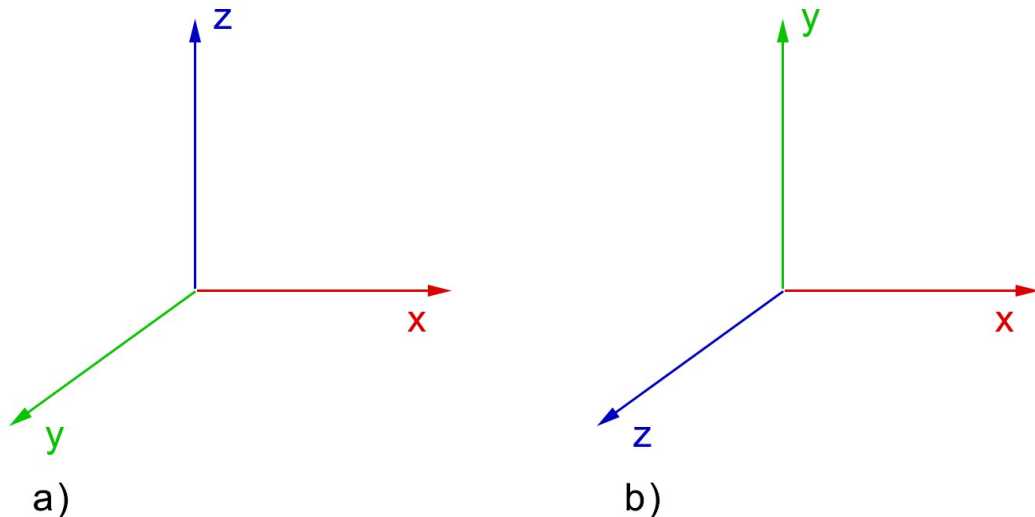


Figure 4. Blender Coordinate System (a) and Unity Coordinate System (b)

A three-dimensional coordinate system has three axes, commonly labeled as X, Y, and Z. The main difference between Blender and Unity's coordinate systems is which axis points up and denotes height. In Blender, the Z axis represents height, while X and Y lie on a horizontal plane (Figure 4, a), while in Unity, the Y axis points up (Figure 4, b). This causes issues when a model created in Blender is imported into Unity without preparation.

### 3.3 Adobe Photoshop CS6

Adobe Photoshop is a powerful graphic design program developed by Adobe Inc. Its development started in 1988 and the program was first released in 1990 for Macintosh computers and in 1993 for Windows. Since version 1.0 Photoshop has gone through a multitude of revisions and updates that added new features and ensured that the program utilizes its full potential. (Brandrick & Burns 2016.)

Adobe Photoshop is a very versatile program. While Photoshop's primary purpose is photo editing, it features a wide array of tools for other purposes. For example, there are tools for digital painting and vector graphics. Moreover, some versions of the software have video editing capabilities. As a result, Photoshop is extremely multi-purpose.

Additionally, Photoshop permits several levels of customization. Beginner users have an ability to create and save custom brushes and patterns. It is also possible to save sequences of operations as actions that can replay those operations, eliminating the need to perform the operations manually which in turn makes the workflow more efficient. Additionally, users experienced in programming are able to write plug-ins and share them online.

Photoshop is extremely powerful when it comes to editing photographs or otherwise creating images. However, it has several drawbacks. One is high memory usage. Another is a steep learning curve due to a massive quantity of features it possesses.

### 3.4 Room Sketcher

Room Sketcher is an online interior design application. It possesses a host of features. The features include: floor plan drawing, floor plan import, and an ability to place furniture. Additionally, the Floor plan can be visualized as a 3D environment, or viewed as a 2D image. (Roomsketcher AS 2016.)

Room Sketcher allows its users to create plans of apartments or homes with accurate measurements. Moreover the users can add windows and doors to the plans, as well as change the appearance of walls and floors. Moreover, Room Sketcher possesses a library of furniture items that can be placed onto the floor plan to create interior designs for various purposes.

#### 3.4.1 ENVI Platform

ENVI (Ensihoidon virtuaalinen oppimisympäristö or Virtual learning environment for the social and health care students) (Männikkö 2016b.) is a virtual learning

environment designed for healthcare students at University of Lapland and hosted there. It consists of a server and several client computers. Students can participate in simulated scenarios or watch other students. The participants are usually in teams of three with a leader and two assistants. Communication and teamwork are vital in successful completion of the tasks posed by the environment.(Männikkö 2016a.)

Currently there are three different scenarios that can be simulated by the program. First is a road accident where the students have to assess a pedestrian who had been hit by a car and the car driver and give them medical assistance. Secondly, there is a cottage fire scenario with several injured where the team has to decide which patient has to be hospitalized in which order based on a number of factors such as person's injuries and the weather. Lastly, there is a person's apartment where the students have to locate things that are not right, such as missing food. (Männikkö 2016a.)

While the application is not used directly throughout this thesis project, the 3D models produced in the process will be used to create another simulation scenario for ENVI. The majority of coding was out of scope of this project. However, the basic programming that enabled animation of some models and user interaction was done.

## 4 BUILDING AN APARTMENT IN BLENDER 3D AND UNITY

### 4.1 Creating a Floor Plan

Using the RoomSketcher software, a basic floor plan for the apartment was created. A large rectangle-shaped space served as a basis for the apartment, and was then divided into rooms that have had their sizes adjusted. The apartment was planned feature two bedrooms, a bathroom and a kitchen\living room area joined with the entry area. After the walls were placed, they were automatically added to the plan by the program. The image of the floor plan was saved for future use and reference (Figure 5).

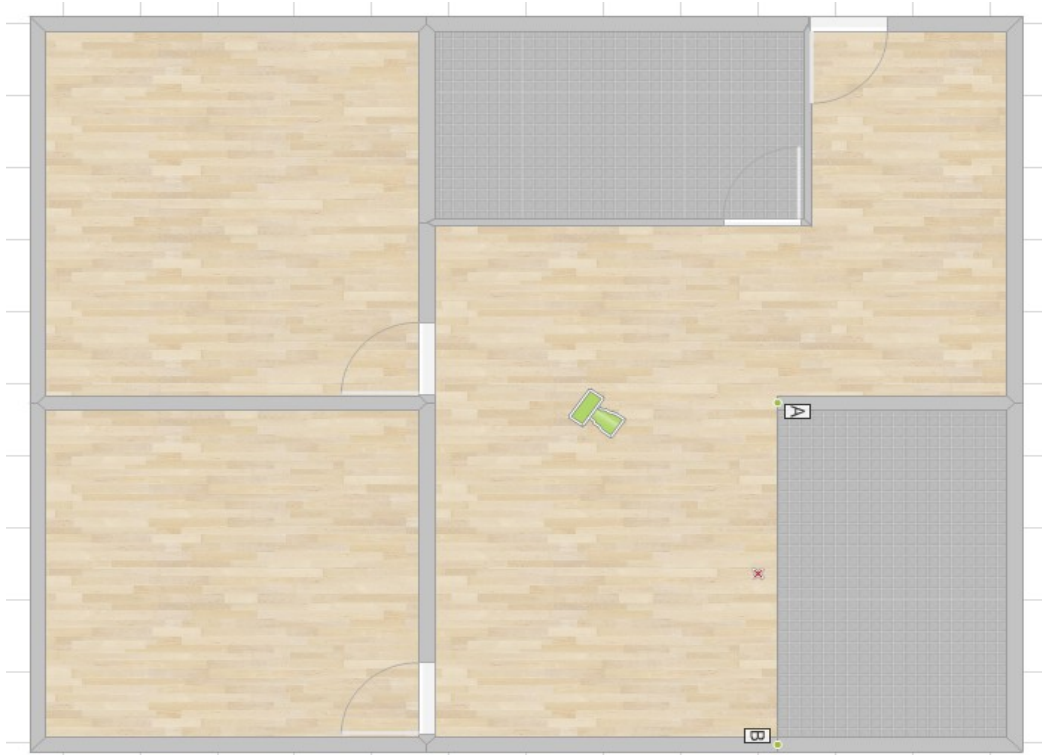


Figure 5. Basic Floor Plan

RoomSketcher allows the users to add furniture to their floor plans. For the purposes of this project, this feature was used to mark the positions the items were supposed to be placed in (Figure 6.). This way, it was possible to determine which models were essential and create a list of those models for each area of the apartment.



Figure 6. Floor Plan with Furniture Placed to Mark the Approximate Positions of 3d Models

## 4.2 Model Creation

After a list of models was established, it was possible to actually start creating the models. The general process was similar across all the models created for this thesis. First, a reference image was found on the Internet to get a general idea of what the chosen model will resemble. Then, in Blender, a suitable shape was chosen as a starting point, and from it, a model was created. After the model achieved the desired look, it was UV mapped and textured.



#### 4.2.1 Model Creation: Television Stand

As there was a large number of models needed for the project, it was inefficient to follow the creation of every single one of them. Therefore, only a few models were described in detail. One of the models was a television stand, and the other was the shell, or the walls and the floor, of the apartment.

The starting point of the model was a cube which was scaled along the Z axis (vertically) to create a flat panel. Then one of the panel's sides that will be in the front was selected and scaled horizontally. As a result, the panel attained a trapezoid shape.

Next the panel was UV mapped. UV mapping refers to the process of projecting a model onto a flat surface so that this surface could be used to create a texture. Edges are important in the process, as they can be marked as seams along with Blender “cuts” the model when projecting it.

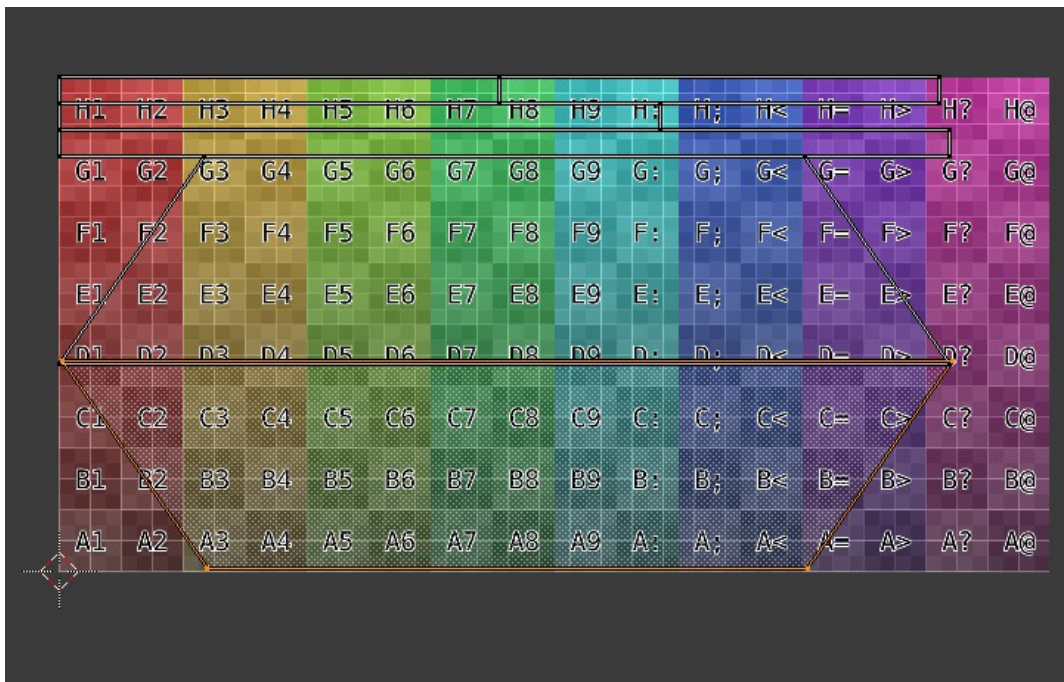


Figure 7. Uv Map of the Television Stand Shelf

To have a texture added to it, the model required a material assigned to it. By default, Blender has a neutral light gray material created for all models. It was possible to use this material to add a texture later, or to change its settings to look differently. It was also possible to add a new material to use, which is what

was done. A new material named “Wood” was added. Naming the materials was optional, but it made distinguishing between materials easier, especially if a model used multiple materials, or was to be used in Unity later.

Then an empty texture was added under the same name. By default, Blender adds one of its generated textures. However, an external image was used, so in a drop-down menu titled “Type” the “image or movie” option was selected. At first, the texture was displayed as pure black because there was no image file attached or no generated texture specified (Figure 8).

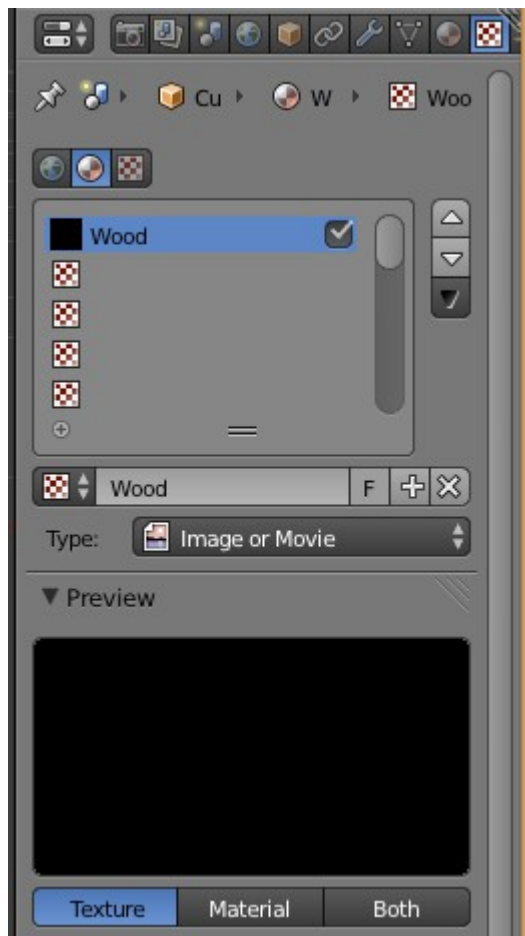


Figure 8. Material's Texture Settings Before a Texture Was Added

Next step was to create a texture image. Blender can generate several types of textures: a blank image, a color grid and an UV grid. The latter two are used to test the mapping to see if there is distortion and to position parts of the UV map better. A color grid is created just for that purpose, and the the UV map is

unwrapped onto it.

The UV map was adjusted by moving portions of it to better utilize the space provided by the image. Then the UV map was exported to be used in texture creation. By default Blender saves the file in the PNG format in the location where the 3D model itself is stored.

It is possible to paint textures directly inside Blender, or generate uniform patterns to serve the purpose. However, usually textures are created in external graphics editing programs, as the textures made that way are higher quality. Moreover, textures created in other software can be tailored to fit the UV map precisely, and are less limiting than generated textures. Therefore, a texture was created in Adobe Photoshop using the UV map as a reference and starting point. An image of a wooden surface was pasted over the UV map to create a texture that gave the model an illusion of being made of wood. Once the texture was ready, it was imported back into Blender and assigned to the model.

Next, a cylinder that would represent one of the legs of the stand was added. The cylinder did not require any texturing, and its material was set to resemble metal. A copy of the cylinder was created by using the “duplicate” tool, then the “legs” were positioned along the corners on one side of the panel. When viewed from the front, the “legs” created previously were placed along one side of the stand. They were copied and via the “mirror” tool, were flipped and placed on the opposite side.

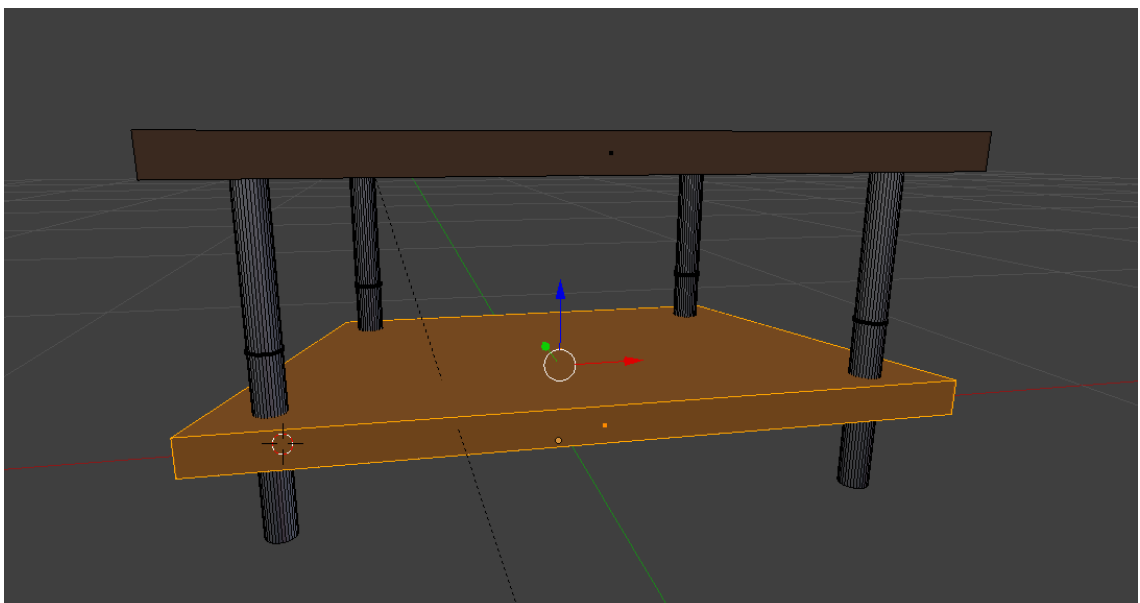


Figure 9. The Stand with Legs and Lower Shelf Added

A shelf was added by duplicating the trapezoid that had been created previously and moving it down along the Z axis so that it was placed slightly above the ground level (Figure 9.). Since the shelf was copied from a previously created object, it had exact same material and texture settings, and uses the same texture image.

#### 4.2.2 Model Creation: Walls and Floors

The apartment itself was modeled in Blender as a series of empty cubes. The cubes were carefully scaled and placed according to the floor plan, with spaces for doors and windows added. The top faces of the cubes were removed, and the normals were set to point inward, making the inside of the cubes visible (Figure 10.). This ensured that, when the model is opened in Unity, the walls are visible when viewed from the inside.

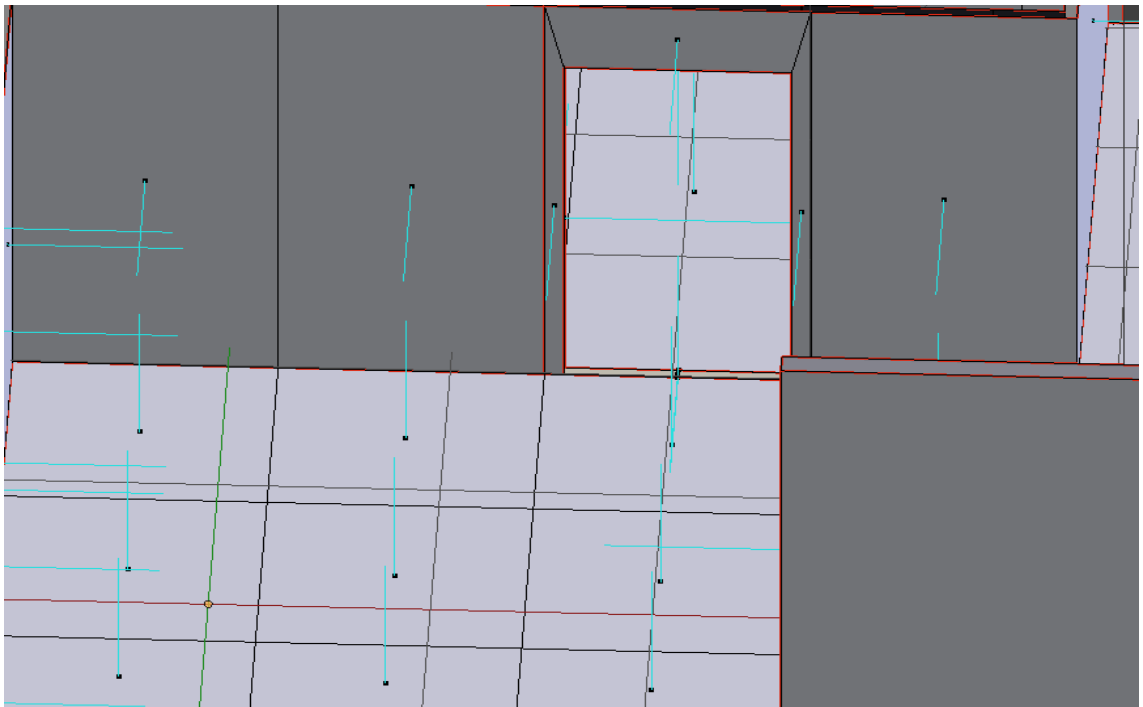


Figure 10. Normals (Cyan Lines)

The cubes were UV-mapped, and several materials were created to represent different floor and wall covering types: tile, wood, linoleum, and paint. The UV maps were exported as image files, and then used to create textures and normal maps. Then, the textures were applied to the model, giving the walls and floors the intended appearance.

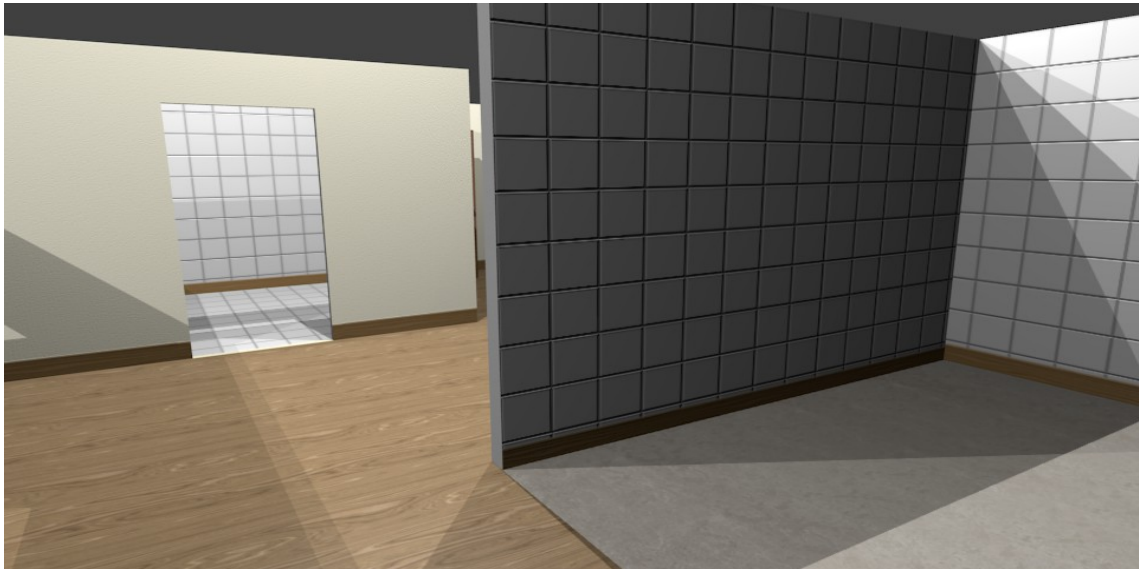


Figure 11. The Apartment Shell with Textures and Normal Maps Applied

Since doors were to be animated, they were created as separate objects each, and added to the apartment later on in Unity. Windows were created as separate objects as well. Both doors and windows were made to fit the openings left in the walls.

#### 4.3 Testing in Unity

The testing process was iterative and was done in a Unity project separate from the project in which the final environment was assembled. It was divided into smaller phases that were repeated as needed. The models were imported into the Unity project and placed onto the scene and checked for errors. If errors were found, the models were opened with Blender and the errors were fixed. After that the models were imported again. Physics, interaction, and animation were implemented and tested in the test project as well.

#### 4.3.1 Preparing a Model for Export to a Unity-Compatible Format.

Before a model was imported into Unity, it needed to be prepared. A number of things were checked before saving the model as a FBX file. First, the model's size was inspected. By default, Blender starts with a large object and uses its own measurement units, so before scaling the model down, the measurement system was switched to metric. Next, the model's origin, or the point that is considered when Unity calculates an object's coordinates, is placed at the bottom of the model (Figure 13). That way, the model is placed properly and is the intended size when loaded into Unity.

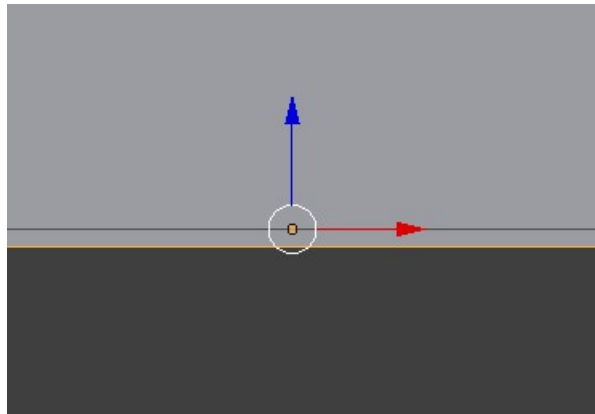


Figure 12: Origin Point in Blender 3D,  
Marked by an Orange Circle

Additionally, the materials created for every model were checked and given more descriptive names. This was done so that later, when several models were imported into Unity, the process of finding a specific material would be easier. The texture and normal map files were renamed in a similar fashion.

#### 4.3.2 Importing Separate Models into Unity

For model testing purposes, a separate empty Unity project was created. This is where all the models were tested to check that they work as intended. Moreover, the test project was used to create and test the animations and camera controls.

While Unity recognizes the file format Blender models are saved in by default (.blend), and models can be imported that way, they are not displayed properly due to differences between Blender and Unity's coordinate systems. Moreover, the shaders and shader settings used by Unity are different from the materials used by Blender, and as a result, some materials required a few adjustments when imported into Unity.

The models were imported into the Unity project and checked. If there were any errors, those errors were fixed and the models were checked again. That was repeated until they were functioning as intended. To summarize, the testing was an iterative process.

#### 4.4 Building the Apartment

The apartment was assembled in a Unity project separate from the one used for testing purposes. All the models were imported and then placed approximately according to the floor plan created earlier. Aside from the models, the code that had been tested, was imported as well. Physics and lighting were implemented.

##### 4.4.1 First Person Controller, Animation and Interaction

A First Person Controller is an object that was created inside Unity and was linked with the camera object. Two C# scripts enabled the controller to respond to keyboard commands and the camera to mouse movements. Consequently, in Play mode, it was now possible to walk inside the apartment, and the camera attached to the object allowed the user to look around.

A C# script was written to create simple rotation animation for various doors. Since almost all the doors rotate in the same way, it was possible to use the same script with very small adjustments to make the doors respond to interaction from the first person controller. In turn, the ability to interact with the surroundings was also added via a C# script called "Interact" (Appendix 1).



Figure 13. The Cursor

For convenience, a cursor was implemented using a plane placed in front of the camera. A transparent texture with an image was applied to it (Figure 13). The plane was attached to the camera object so that it would always stay in the center of the screen, making interaction with the objects inside the apartment more simple.

#### 4.4.2 Physics and Lighting

Physics were implemented via the use of collider and Rigidbody components. Colliders were added to the apartment walls and all the furniture items within. A Rigidbody was added to the First Person Controller. Collider components were also vital in adding interaction.

Lights were created using a combination of 3D models of several types of lighting fixtures and light sources provided by Unity. The light sources were attached to the 3D models of lamps to emulate light coming from them. The lamps were then placed on ceilings, walls, and tables.

Finally, a lightmap was created. A lightmap is a texture generated by Unity that takes into account all the light sources present, the models' shape, texture and normal maps. This texture is then overlaid on top of the model to simulate light hitting its surface. The results are more versatile and realistic than using Unity's light objects alone.



#### 4.4.3 Delivery of the Finished Environment

The finished Unity project was exported as a Unity package file. Unity Package files store all the information needed for a project. They are independent of the model and other asset files, and can be opened within a Unity project. This makes Unity packages a good way to transport a Unity project between developers.

Despite the Unity package being independent of the 3D model files and other assets, those were delivered as well. This was done as a backup measure. The Unity project itself was delivered as additional backup. Moreover, access to the model files will allow the project and its parts to be used again in the future.

## 5 CONCLUSIONS

Building a 3D environment using Blender 3D and Unity, especially as a beginner 3D modeler and Unity user, was a challenging process. Over the course of the project work, new skills were learned in order to overcome challenges that arose. The work required a lot of planning: making a map or a floor plan, making a list of models to create, organizing models inside Unity.

Planning was a more difficult part of the project. On the one hand, it was important to follow the plan through and deviate from it as little as possible. On the other hand, no plan takes into account everything, so sometimes alterations to the plan were necessary. Creating 3D models also required several iterations of testing to ensure that they function as intended. As a result, it required a fair amount of time to complete the project.

The modeling process itself was less difficult than anticipated. It was challenging in the beginning, but became easier over time. 3D modeling skills were learned and improved.

## 6 DISCUSSION

### 6.1 Reliability, Usefulness and Purpose

The models were checked and tested inside Unity several times to assure they work as intended. Similarly, the few code scripts were tested in a similar fashion, to check if the interaction and the objects they were attached to behave as planned. However, since Unity undergoes a constant update process, some features might not function properly or work differently if accessed using an older or newer version of Unity. It is recommended to use the latest version for development, maintenance and updating.

When assets are created for video games, they are often re-used in several locations throughout the game. This tactic saves time and resources that otherwise would have been spent on creating unique models for each distinct part of the game. The models created during the work on this project are rather general purpose, so they can be re-used for a variety of other projects, including other environments and scenarios within ENVI.

This project and parts of it will be used as one of the virtual locations in an education program. Assets created for this project may be used in other parts of ENVI platform. As a result, the 3D environment will be a part of the education process for healthcare students at Lapland University of Applied Sciences.

## BIBLIOGRAPHY

3D Printing.com. 2016. What is 3D printing? Accessed 07 November 2016.  
<http://3dprinting.com/what-is-3d-printing/>

BlenderWiki 2016. Doc:2.4/Manual/Interface/Modes. Accessed 17 November 2016.  
<https://wiki.blender.org/index.php/Doc:2.4/Manual/Interface/Modes>

Blender Foundation 2016a. About Blender. Accessed 15 March 2016  
<https://www.blender.org/about/>

Blender Foundation 2016b. Features. Accessed 15 March 2016  
<https://www.blender.org/features/>

Blender Foundation 2016c. History. Accessed 15 March 2016  
<https://www.blender.org/foundation/history/>

Blender Manual 2016. Introduction. Modeling Modes. Accessed 17 November 2016.  
<https://www.blender.org/manual/modeling/meshes/introduction.html>

Blender Reference Manual 2016. Interface. Accessed 23 April 2016  
<https://www.blender.org/manual/interface/introduction.html>

Brandrick C., Burns M. 2016. Adobe Photoshop turns 26: chart its history from version 1.0 to CC 2015. Accessed 17 November 2016.  
<http://www.digitalartsonline.co.uk/news/creative-software/adobe-photoshop-celebrates-26th-birthday/>

Digital-tutors 2014a. Key 3D modeling terminology you need to master. Accessed 15 March 2016  
<http://blog.digitaltutors.com/basic-3d-modeling-terminology/>

Digital-tutors 2014b. 3D texturing terminology. Accessed 15 March 2016  
<http://blog.digitaltutors.com/cover-bases-common-3d-texturing-terminology/>

Enger M. 2013. Game Engines: How do they work?. Accessed 09 October 2016  
<http://www.giantbomb.com/profile/michaelenger/blog/game-engines-how-do-they-work/101529/>

ExtremeTech 2002. Game Engine anatomy 101, part I. Accessed 17 November 2016.  
<https://www.extremetech.com/computing/50944-game-engine-anatomy-101-part-ii>

Haas J. 2012. A History of the Unity Game Engine. An Interactive Qualifying Project. Worcester Polytechnic Institute.

Hajioannou Y. 2013. Gamedev Glossary: What Is a "Normal Map"? Accessed 15 March 2016

<http://gamedevelopment.tutsplus.com/articles/quick-tip-what-is-a-normal-map--gamedev-3893>

Jonpolygon 2016. Industries Utilizing 3D Models. Accessed 17 November 2016.  
<https://jonpolygon.com/who-uses-3d-models/>

Männikkö E. 2016a. ENVI and thesis question. Email 28 January 2016.  
evgeniya.sheshina@edu.lapinamk.fi Printed out 10 May 2016.

Männikkö E. 2016b. Re: Some more questions about ENVI and 3D modeling.  
Email 05 September 2016. evgeniya.sheshina@edu.lapinamk.fi Printed out 09 October 2016.

Masters M. 2013. Key 3D Rigging Terms to Get You Moving. Accessed 04 November 2016.  
<http://blog.digitaltutors.com/key-rigging-terms-get-moving/>

Rasmussen A. 2013. How 3D Modeling Changed Architectural Presentations.  
3D Printing.com. Accessed 07 November 2016.  
<http://3dprinting.com/products/architecture/how-3d-modeling-changed-architectural-presentations/>

Roomsketcher AS. 2016. RoomSketcher Home Designer. Accessed 08 November 2016.  
<http://www.roomsketcher.com/features/home-designer/>

Schneider P. J. 1996. NURB Curves: A Guide for the Uninitiated. MacTech Develop March 1996. Accessed 05 September 2016.  
[http://www.mactech.com/articles/develop/issue\\_25/schneider.html](http://www.mactech.com/articles/develop/issue_25/schneider.html).

Silverman D. 2013. 3D Primer for Game Developers: An Overview of 3D Modeling in Games. Accessed 15 March 2016  
<http://gamedevelopment.tutsplus.com/articles/3d-primer-for-game-developers-an-overview-of-3d-modeling-in-games--gamedev-5704>

Slick J. 2014. 7 Common Modeling Techniques for Film and Games.  
Lifewire.com. Accessed 31 March 2016  
<https://www.lifewire.com/common-modeling-techniques-for-film-1953>

Tech World 2011. Blender Review. Accessed 23 April 2016  
<http://www.techworld.com/review/graphic-design-and-publishing-software/blender-review-3320178/>

Unity Manual 2016a. Animation system overview. Accessed 17 November 2016.  
<https://docs.unity3d.com/Manual/AnimationOverview.html>

Unity Manual 2016b. Colliders. Accessed 06 November 2016  
<https://docs.unity3d.com/Manual/CollidersOverview.html>

Unity Manual 2016c. GameObjects. Accessed 06 November 2016  
<https://docs.unity3d.com/Manual/GameObjects.html>

Unity Manual 2016d. Rigidbody overview. Accessed 06 November 2016  
<https://docs.unity3d.com/Manual/RigidbodiesOverview.html>

Unity Manual 2016e. Scenes. Accessed 17 November 2016.  
<https://docs.unity3d.com/Manual/CreatingScenes.html>

Unity Technologies 2016a. Unity – game engine, tools and multiplatform.  
Accessed 15 March 2016  
<http://unity3d.com/unity>

Unity Technologies 2016b. Company facts. Accessed 15 March 2016  
<http://unity3d.com/public-relations>

Unity Technologies 2016c. Interface overview. Accessed 05 September 2016  
<https://unity3d.com/learn/tutorials/topics/interface-essentials/interface-overview>

## APPENDICES

Appendix 1.                   C# scripts used in the project

Below the code of the scripts used in the project is presented. There was a total of four C# scripts.

The CamMouseLook script was responsible for receiving input from the mouse and relaying it to Unity's camera object. As a result, the user is able to use the mouse to look around the environment.

```
using UnityEngine;
```

```
using System.Collections;
```

```
public class camMouseLook : MonoBehaviour {
```

```
    Vector2 mouseLook;
```

```
    Vector2 smoothV;
```

```
    public float sensitivity = 5.0f;
```

```
    public float smoothing = 2.0f;
```

```
    GameObject character;
```

```
    // Use this for initialization
```



```
void Start () {  
  
    character = this.transform.parent.gameObject;  
  
}  
  
// Update is called once per frame  
  
void Update () {  
  
    var md = new Vector2 (Input.GetAxisRaw ("Mouse X"),  
Input.GetAxisRaw ("Mouse Y"));  
  
    md = Vector2.Scale (md, new Vector2 (sensitivity * smoothing,  
sensitivity * smoothing));  
  
    smoothV.x = Mathf.Lerp (smoothV.x, md.x, 1f / smoothing);  
  
    smoothV.y = Mathf.Lerp (smoothV.y, md.y, 1f / smoothing);  
  
    mouseLook += smoothV;  
  
    transform.localRotation = Quaternion.AngleAxis (-mouseLook.y,  
Vector3.right);  
  
    character.transform.localRotation = Quaternion.AngleAxis  
(mouseLook.x, character.transform.up);  
  
}  
  
}
```

The CharacterController script lets the user control an object with the camera attached to it using the keyboard.

```
using UnityEngine;
```

```
using System.Collections;
```

```
public class CharacterController : MonoBehaviour {
```

```
    public float speed = 10.0F;
```

```
    // Update is called once per frame
```

```
    void Update () {
```

```
        float translation = Input.GetAxis ("Vertical") * speed;
```

```
        float straffe = Input.GetAxis ("Horizontal") * speed;
```

```
        straffe *= Time.deltaTime;
```

```
        translation *= Time.deltaTime;
```

```
        transform.Translate (straffe, 0, translation);
```

```
    }
```

```
}
```

The DoorOpenClose script was what added animation to the doors inside the apartment. The animation was done via code, and was adjusted for each door object as needed.

```
using UnityEngine;
```

```
using System.Collections;
```

```
public class DoorOpenClose : MonoBehaviour {
```

```
    public bool open = false;
```

```
    public float doorOpenAngle = 90f;
```

```
    public float doorClosedAngle = 0f;
```

```
    public float smooth = 2f;
```

```
    // Use this for initialization
```

```
    public void ChangeDoorState () {
```

```
        open = !open;
```

```
    }
```

```
    // Update is called once per frame
```

```
    void Update () {
```

```

        if (open)
        {
            Quaternion targetRotation = Quaternion.Euler (0, doorOpenAngle, 0);

            transform.localRotation = Quaternion.Slerp (transform.localRotation,
targetRotation, smooth * Time.deltaTime);

        }

        else

        {

            Quaternion targetRotation2 = Quaternion.Euler (0, doorClosedAngle, 0);

            transform.localRotation = Quaternion.Slerp (transform.localRotation,
targetRotation2, smooth * Time.deltaTime);

        }

    }
}

```

The Interact script enabled the user to interact with the environment. For the purposes of this project, the interaction was limited to opening and closing doors, but it is possible to extend the interaction in a variety of ways.

A Raycast is a feature that emits an invisible ray from the game object, which has properties such as distance. If the ray collides with an object with a specific “Tag” property, for example, one tagged as a door, and left mouse button is pressed, the script attached to the door object will be called.

```

using UnityEngine;
using System.Collections;

```

```

public class Interact : MonoBehaviour {

    public float interactDistance = 9f;
    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

        if (Input.GetKeyDown (KeyCode.Mouse0))
        {
            Ray ray = new Ray (transform.position,
transform.forward);
            RaycastHit hit;
            if (Physics.Raycast (ray, out hit, interactDistance))
            {
                if(hit.collider.CompareTag("Door"))
                {

                    hit.collider.transform.parent.GetComponent<DoorOpenClose>
().ChangeDoorState ();
                }
            }
        }
    }
}

```