Stylized Medieval Village

Modular 3D Asset Package

Bachelor's Degree Thesis

Degree in Video Games Design and Development

Author: Joan Barduena Reyes

Director: Marc Ripoll Tarré

Year: 2022

To Anna Torrano, Josep Lleal, Ana Maria Reyes, and Marc Ripoll

Thank you

Index

Index of Figures

Index of Tables

Summary

This document describes and explains the process of developing a three-dimensional stylized medieval village asset package. The pack is modular, so lots of different combinations are possible. Meshes, textures, materials, and scenes are all created from scratch.

The project results into an asset package with more than 140 unique meshes, several materials and material instances that allow to change several parameters, some prefabs, as well as two example scenes, one to show the deployed props and the other with a ready-to-play scene with already constructed buildings and street decorations placed.

Though this document, the objectives, research, some analysis, along with the management and planning of the project, are explained.

Then, there is the development process, where the techniques and procedures used for the package are described. This step goes from the modeling of the first meshes, through creating procedural materials, the adaptation of the tool into Unreal Engine 4 and the different problems and difficulties encountered during the whole development process.

Once the project was finished, it was concluded that it could be good to adapt the package into Unity Engine, in conjunction with a wider variety of props and materials as well as resolving bugs, errors, and add possible improvements.

Keywords

Asset package, Stylized, Modular, Environment, Unreal Engine, Materials, Props, Blender, Substance Designer.

Links

YouTube video:

https://youtu.be/HXrqzWvX4gs

Unreal Engine Marketplace Asset Package:

https://www.mediafire.com/file/f6cfz98fkgn8tbs/BarduenaJoan_MedievalAssetPack.zip/file

Glossary

Asset: Shorthand for anything that goes into a video game – characters, objects, sound effects, maps, environments, etc.

Baking: A method of preprocessing performed on game assets and data to ensure they load and perform well in real-time and do not slow down gameplay due to requiring a lot of processor or GPU capacity.

Blueprint: Blueprint Visual Scripting system in Unreal Engine is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within Unreal Editor.

Bug: Any development issue that makes a game unenjoyable, unstable, or unplayable in its current state.

Collision: The action of two objects coming together and touching/striking one another in-game.

Demo: A proof-of-concept version of a game, typically released to the public for promotional and/or feedback purposes.

Developer: A person who helps to transform games from a concept to a playable reality. They do this by coding visual elements, programming features, and testing iterations.

Edge: The connection between two vertices of an angle.

Game engine: Software that offers a suite of tools and features to game developers to build their games professionally and efficiently.

Indie: A video game typically created by individuals or smaller development teams without the financial and technical support of a large game publisher.

Mesh: A collection of vertices, edges, and faces that act as the foundation of a model in a video game.

Model: A fully 3D asset in a video game that is created by adding textures and other features to a mesh.

PBR: Stands for **P**hysically **B**ased **R**endering and means that the material describes the visual properties of a surface in a physically plausible way, such that realistic results are possible under all lighting conditions.

Polygon: A computer-programmed series of lines that form a three-dimensional (3D) object.

Prop: Interactive objects in a game.

Prototyping: Creating different early versions of a game to explore different mechanics and features to decide which will be best for the full game.

Render: The act of continuously generating and refreshing a 2D or 3D image through computer processing.

Shaders: Small programs within larger game development processes are typically used to control lighting and shadow effects.

Sprite: Bitmap images, often used as 2D Game Objects. In 3D, sprites generally function as textures.

Terrain: Anything that creates the environment in a video game.

Texture: A visual wrapping placed around Game Objects, such as the skin on a character.

Texture mapping: The process of applying textures to Game Objects.

Tile: An image that is used to create other, bigger images (such as a platform) in a 2D game.

Tilemap: A system that stores and handles tile assets for creating 2D levels.

Trim sheet: Several textures laid out side-by-side inside one bitmap.

UV: Texture coordinates of an object.

Vertex: A point in 2D or 3D space. Joining two vertices together forms an edge.

1. Introduction

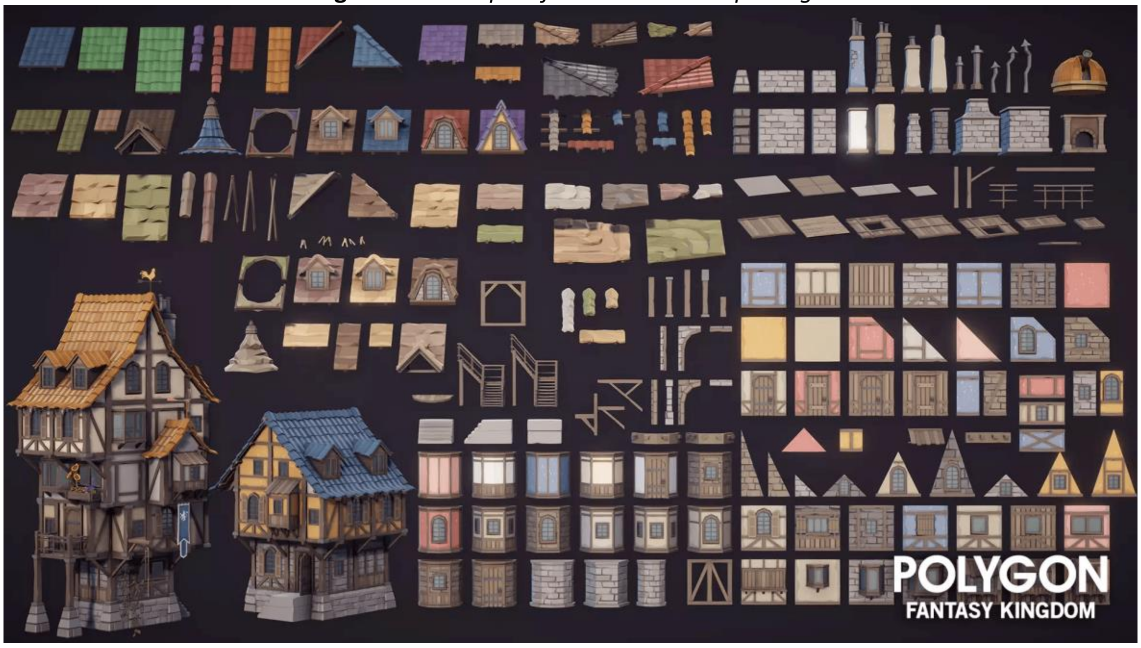
In this document, the author will explain his research about asset packages, how they are developed and the modularity they can have. This 3D asset package will be made with a set of tools and will be uploaded to Unreal Engine Marketplace.

The process of creating video games is very slow and complex. Big video game studios have specific art teams to develop the environment of their projects. However, there are lots of small and indie companies that can't afford to have a team like that. Even though every time there are more procedural ways to create environments, these tools are not the standard in the industry yet.

So, this project is about the development of several 3D stylized props with medieval style to allow the creation of a medieval village with a wide variety of combinations, as *Figure 1.1* shows. These assets are modular, so the number of building variations that can be made in a single environment is huge.

Considering that there are entire teams that work to develop these kinds of environments, this project focuses on achieving a small and basic environment but with enough quality to be on the market. For that, only the exterior of the village buildings and some street props will be developed, but not the interiors.

Figure 1.1 *Example of a modular asset package*



Note: Image from Synty Studios of their POLYGON Fantasy Kingdom asset pack

1.1 Motivation

The main motivation behind this project is the need for a good quality stylized asset pack suitable to use in videogames in terms of performance. There are many low poly asset packs as well as realistic ones, but very few stylized. The purpose is to develop a single asset pack with a wide range of different houses and city creations, so everyone that uses it will have the possibility of creating an environment that suits the needs of each.

There is also motivation for this project as it will help to learn Substance Designer intended for creating 2D textures, materials, and filters with a heavy focus on procedural generation, parametrization, and non-destructive workflows. Substance Designer is growing in the industry, and it has become part of the texture creation workflow. It is not easy to use as both logical and artistic knowledge are needed to develop good results.

1.2 Formulation of the Problem

There are hundreds of developers out there trying to develop video games. They want people to play their games and have fun with them. It can't be disagreed that the game design is a key feature to achieving this, but also the game art.

The main problem of video game industry today is Crunch. In 2019, 40 percent of game developers reported a working crunch at least once over the previous year (International Game Developers Association, 2019). For most of these developers, crunch wasn't just a few extra hours or a long weekend, but at least 20 extra hours on top of their standard 40-hour workweek.

Some videogames like House Flipper from Empyrean, which was nominated for Game Simulation at National Academy of Video Game Trade Reviewers (NAVGTR) 2018, uses an external asset pack not developed by them. This asset pack[1] can also be found in games like Exposure by Radmir Kadyrov.

Although this may not solve crunch problem, it allows companies to minimize the level of work and focus more on the game design of the video game instead of having to spend time and money on creating their own assets.

Time is a key point to purchase an asset pack, and so is the knowledge. Some developers, mostly indie developers, may not have the knowledge or the resources to create high-quality game assets.

Having all this into account, asset packs help game developers create detailed and higher quality environments in less time and are already optimized.

[1] Unreal 4: Modular neighborhood pack https://www.youtube.com/watch?v=-UFx0IgsxRo

1.3 General Objectives

The main objective of the project is to create a good-quality modular asset pack of a medieval stylized city. The focus will be on the exterior of the buildings. Materials for walls, roofs, and floors will be developed in Substance Designer, whereas props that need volume, like barrels, stairs, and cages, will be developed following this workflow: Blender, Zbrush, and Substance Painter.

General Objectives can be divided into the following points:

- Make the asset pack ready and easy to use for everyone.
- Develop a study of the market and publish this asset pack at a suitable price.
- Study the desired style of the assets with different references.
- Learn and expand the knowledge on the programs that will be used, to be able to create good materials.
- Document the process to show the study behind the creation of the asset pack.
- Create an Unreal Engine 4 project with all the assets in it.

1.4 Specific Objectives

As mentioned before, the main objective of the project is to create the modular asset pack in Unreal Engine 4. Taking medieval cities as a reference, we will recreate exterior buildings and some assets from the streets. Once the materials and props are completed, the project will be released to Unreal Marketplace for people to use and enjoy. The specific objectives are:

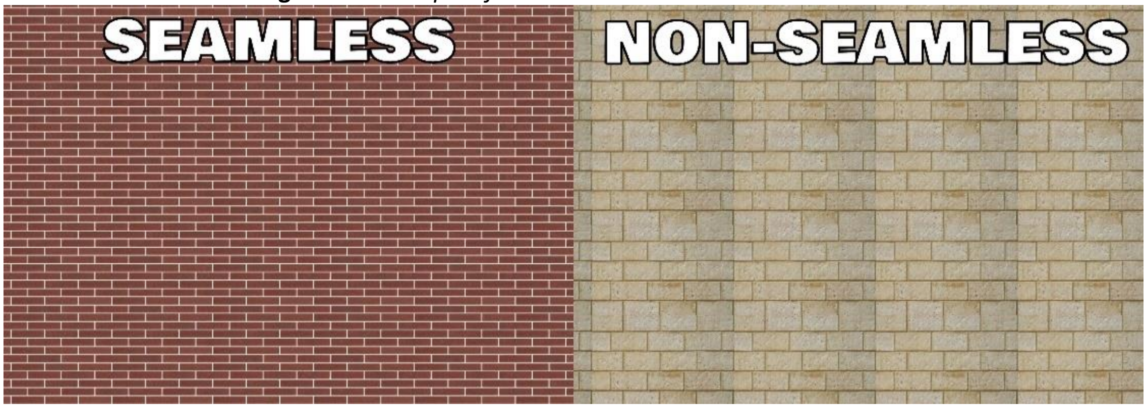
- Make sure there are enough assets, so users can create a wide variety of buildings different from each other.
- Make sure all assets have the same style and look good together.
- Work on performance, so assets need to have the minimum number of polygons but without losing quality. For example, baking the high poly into the low poly model.
- Create a demo project in Unreal Engine 4 showing one city built with the assets.

- Name all the assets and organize them into folders with the corresponding materials into the UE4 project.

- Add different texture sizes (1024, 2048, 4096) so users can choose the one that suits them better.

- Textures need to have modularity and be tileable or seamless. This means a texture image can be placed next to itself without creating obvious seam, join, or boundary between the copies of the image.

Figure 1.2 *Example of seamless and non-seamless texture*



Note: Images extracted from an article on PlusSpec web

1.5 Scope and Beneficiaries

Having all the objectives clear, this project is sure to be long. Apart from doing the project itself, it also must be documented and there are programs like Substance Designer, which knowledge needs to be expanded, to have good results. Before starting to develop the project, it is good to have clear goals for which assets to do, how to do them, and try them out, so time isn't lost in the middle of the development.

Given a time limit, boundaries need to be set on what to develop of the medieval city. Only the exterior of the buildings will be done and some props and materials for decoration of the streets. So first, we will start working on the most basic and necessary assets that are needed to build the city, so we have a solid base that is both functional and well developed. From there, we will focus on developing more assets that will help on having more variety and diversity in the buildings.

Relating to beneficiaries, this project is oriented to people that want or need to build one environment for a project but don't have the time, the knowledge, or the passion to develop it. Maybe because they want to focus more on the designing part of the project rather than the artistic one. So, they can easily design a medieval city in their own way.

2. State of Art

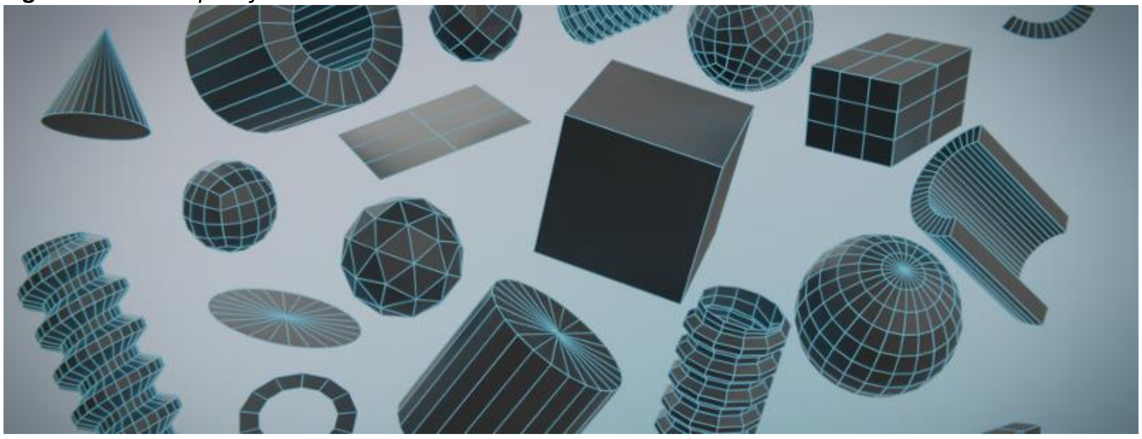
2.1 Processes for the creation of 3D models

2.1.1 3D Modeling

3D modeling is the process to create a three-dimensional representation of an object in a virtual space generated on a computer. These objects are called 3D models and they are represented using a collection of dots/vertices inside a 3D space. Vertices are connected creating geometrical shapes like triangles, quads, curves, lines, etc. And the union of those shapes form what is called a mesh.

The process of modeling consists in the creation and manipulation of these vertexes that create the mesh with a series of tools and parameters.

Figure 2.1.1 *Example of basic 3D models*

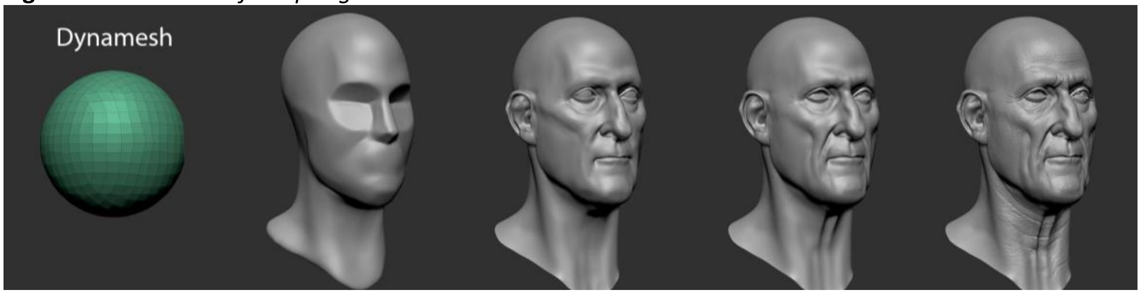


Note: Image from Blender Nation (Add-on: Wonder Mesh)

2.1.2 Digital Sculpting

Digital sculpting is a popular way of representing a 3D model. In this case, instead of gradually moving and creating small blocks of polygons, the process involves manipulating a polygon mesh of an item where the artist can push, pull, smooth, grab, pinch, or otherwise handle the digital object as it was made of real-life substance as clay.

Figure 2.1.2 *Process of sculpting*



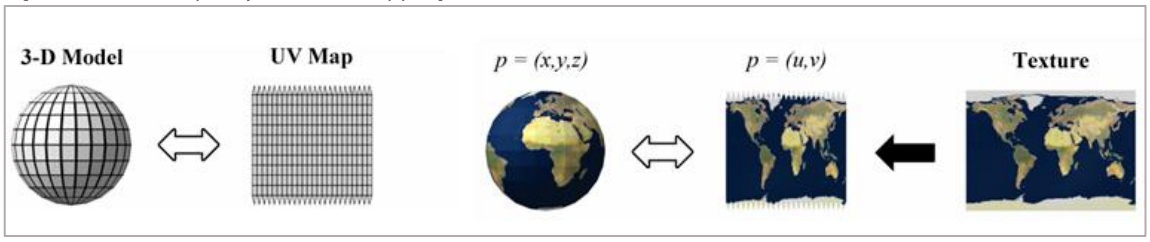
Note: Image from an article on 80.LV (Realistic Facial Features in 3D Character Sculpts, 2019)

2.1.3 UV Mapping

UV Mapping, also known as unwrapping or unfold, is the process of projecting a 2D image onto a 3D model's surface. It is called UV as the "U" and the "V" are denoted for the 2D texture axis "X" and "Y".

To create UVs, the 3D model is unfolded with seams and laid flat on a 2D plane. The pixels on the 2D image are assigned to the surface polygons and the rendering computation uses the UV texture coordinates to determine how to paint the three-dimensional surface.

Figure 2.1.3 *Example of how UV mapping works*

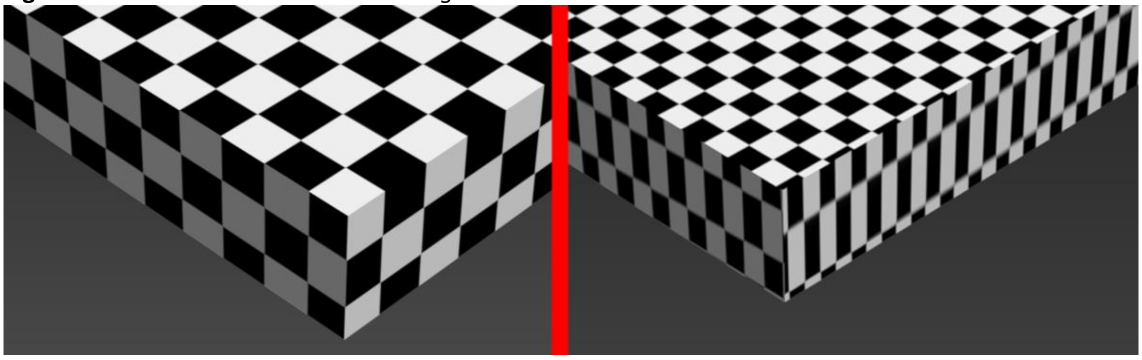


Note: Image from Wikipedia (UV mapping)

UV mapping requires three steps: unwrapping the mesh, creating the texture, and applying this texture to the polygon's face of the model.

To check if UV mapping is performed correctly, a texture mapping grid is commonly used. Squares need to look uniform when projected into the model. If these squares look stretched out in some parts of the model, the final texture will look inadequate, as shown in *Figure 2.1.4*. That's why it is so important to have good UVs.

Figure 2.1.4 *Good vs Bad UVs with a UV grid*



Note: Image from Substance3d forum (topic: smart material stretching, 2016)

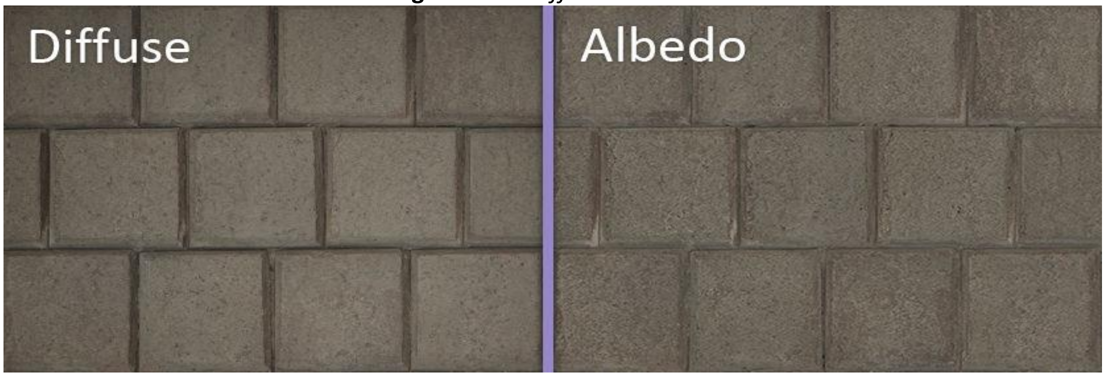
2.1.4 Texturing

Once UV Mapping is correctly set, texturing process takes place. Instead of placing a texture mapping grid, other textures that provide information to the render engine about different aspects of the behavior of the model's surface will be placed.

In this project, some of the PBR (Physically Based Rendering) textures will be used. These textures produce visuals in a way that mimics the flow of light in the real world by simulating how materials absorb and reflect light. There are 10 different texture types which lead to a photorealistic result that mimic the real-world conditions when placed correctly. To create a material, there is a combination of several texture types which has a different function, so when placed together each texture brings a property to the material. However, in most cases, all PBR textures are not combined in one single material, but just a few. In this project, only the following seven out of ten PBR maps are going to be used:

Albedo Map. This map is the base color input that defines the diffuse color without any reflection, shadows, or lighting. It shouldn't be confused with diffuse map since this one does have this kind of information.

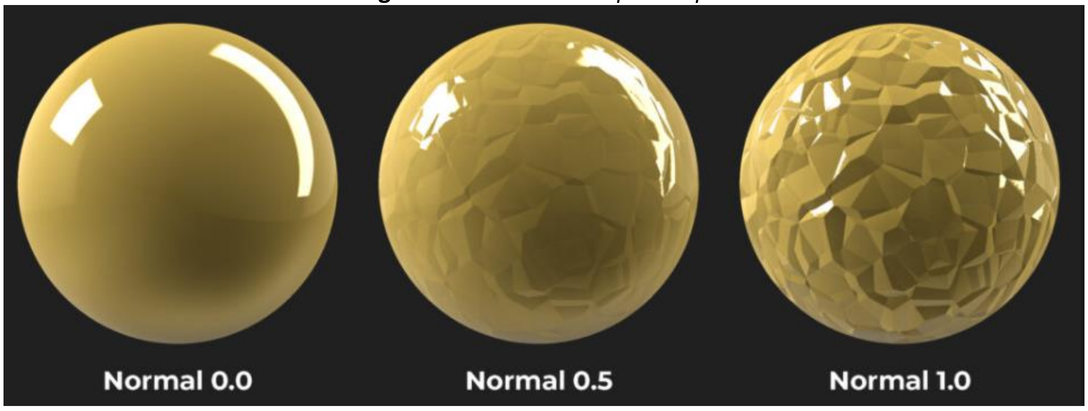
Figure 2.1.5 *Diffuse vs Albedo*



Note: Image from "Texture maps" on Anipro.pro

Normal Map. It provides the texture depth and simulates how the light interacts with the surface without altering the geometry of the model.

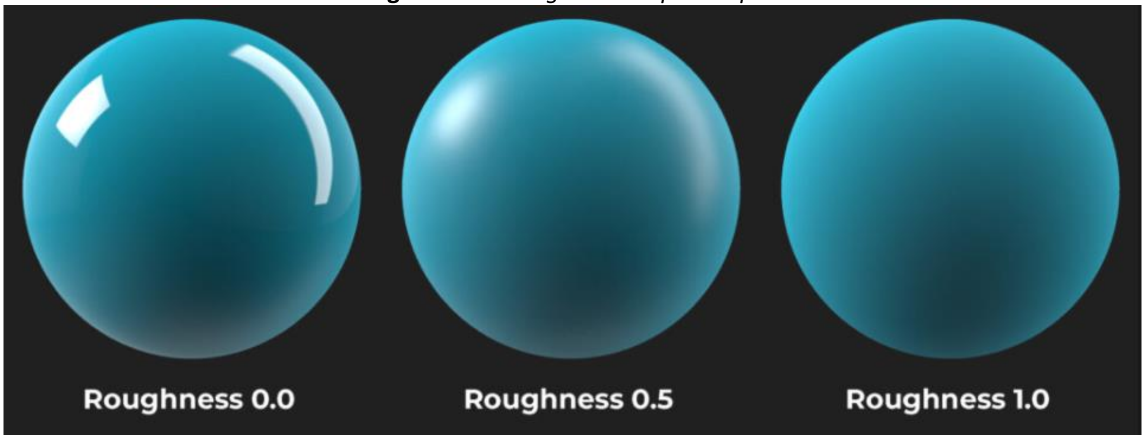
Figure 2.1.6 *Normal map example*



Note: Image from the article" Different maps in PBR textures" on A23D

Roughness Map. Determines how rough or smooth a surface is and how light is spread across the model's surface. The range goes from 0 to 1. When the value is 0, reflections are crisp, and the model doesn't scatter light at all. Reflections get fuzzy when the value is 1 and the light is more dispersed throughout the material.

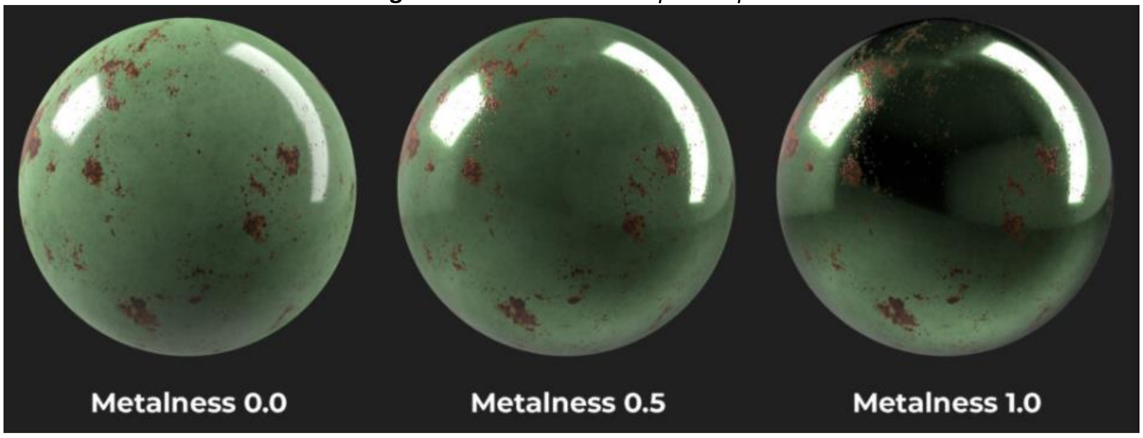
Figure 2.1.7 *Roughness map example*



Note: Image from the article" Different maps in PBR textures" on A23D

Metalness Map. It refers to how much a surface reflects its surroundings. When the value is 0 the albedo color is entirely visible which seems plastic material. However, when the value is 1 the albedo is practically lost, and the environment is fully reflected. When roughness is 0 and metalness is 1, the surface is like a real-world mirror.

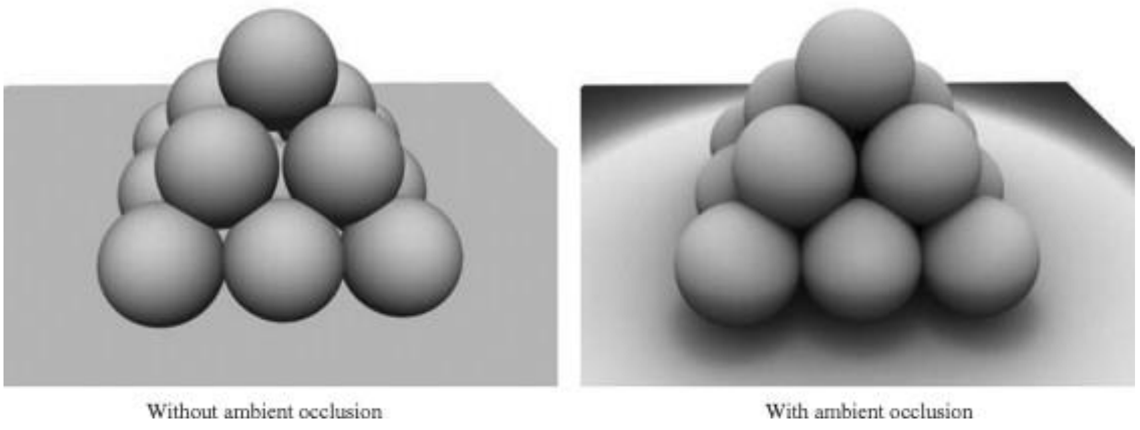
Figure 2.1.8 *Metalness map example*



Note: Image from the article" Different maps in PBR textures" on A23D

Ambient Occlusion. Adds shadows to occluded zones of the model. This map is mixed with albedo to describe how the light would reflect. At value 0 the map is complete darkness and at 1 there aren't occlusions.

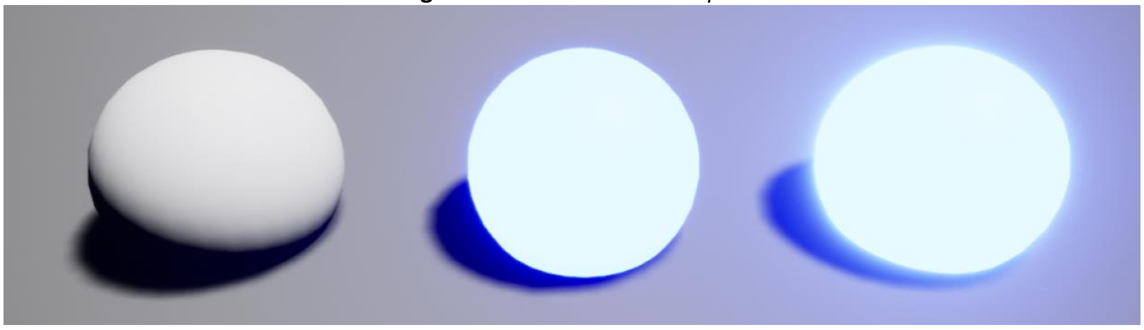
Figure 2.1.9 *Ambien Occlusion example*



Note: Image from the article "Ambient Occlusion" on ScienceDirect

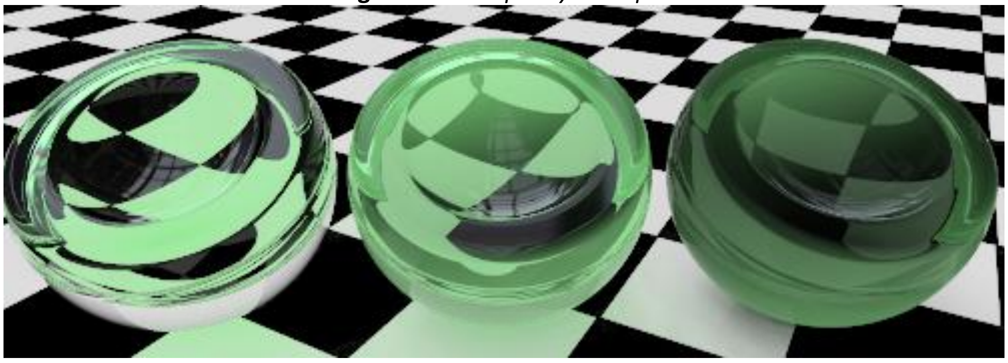
Emissive Map. This map element of the material radiates its own light, so it can be seen in dark places. In this project, it will be useful to create the light inside light posts and windows. It is developed in an RGB map as well as the albedo map, but it is for light.

Figure 2.1.10 *Emissive example*



Opacity Map. This map allows to make areas of the material transparent. In this project it will be crucial for building windows as greyscale opacity levels are available. While white is completely opaque, black is completely transparent.

Figure 2.1.11 *Opacity example*



Note: Image from article "Advanced Material Properties" on Flamingo nXt

All these textures together form a Material that made possible to simulate near-photorealism in real-time. Besides, this vastly reduces the number of polygons and lighting calculations as textures are faking these parameters with much less impact on performance as they aren't calculated in real-time but when the maps are being made.

2.1.5 Animation

Sometimes, 3D models are not just static and that's where 3D animation takes place. The mesh is given an internal digital skeletal structure with a set of controllers that control a group of mesh vertex. This process is called rigging and can be used in conjunction with keyframes to create movement while moving the controllers. Models appear to move through three-dimensional space and can be moved, rotated, and scaled.

At the same time, some techniques like gravity, simulated hair and fire, and water simulations can be applied to 3D models.

It is not planned to be any animations in this project, but things like open doors and windows could be animated.

2.1.6 Rendering

3D rendering is the process to create bi-dimensional images (for example a computer monitor) from a 3D model. These images are generated from a set of data that dictate what textures, materials, and light a certain object has in the image. This image is referred as the render.

Rendering can be a slow and computationally intensive process that has two variants:

- **Pre-rendering:** It is usually generated ahead of time, for example in movie creation, 3D models representations like architecture archive, movie, and television, animation, and commercials, among others. Computing resources are used to calculate visual images of the models according to several settings of lighting, camera, viewport, etc. In this case, all the render process takes place in one step and when it is finished, frames can be seen. Each frame in the pre-rendered scene is present and it takes seconds, minutes or even hours to be rendered. It takes a large amount of GPU and CPU as well as storage resources and energy consumption. That's why in big productions like movies, renders are developed in rendering farms, which provide massively parallel computer clusters.
- **Real-time:** It is used to interactively render a 3D scene, like 3D video games, and generally each frame must be rendered in a few milliseconds. This means the computer is displacing the scene on the screen while computing it. Unreal Engine does this very well, that's why it fits perfectly into this project. This allows a scene than can be controlled and interactive at the same time. Real-time renderings have usually a lower quality than pre-rendering ones as they are limited to one system and are time-constrained. However, as all technologies improve, the speed of real-time calculations is getting faster while providing higher quality images (especially with Ray-tracing).

2.2 Industry pipeline

3D modeling has been in the industry for a long time. The first arrival of the first 3D modeling and rendering software was in the 70's. It was available through personal computers with a program called Sketchpad, but it was too expensive and unavailable to most people in that period. In 1972 Ed Catmull and Fred Parke created the world's first 3D rendered film[2], an animated version of Ed's left hand.

All technologies have evolved, and the 3D modeling pipeline has changed as well as the programs used. Nowadays, the most common pipeline and corresponded software to develop three-dimensional models are:

- **Concept art**. This will help to see the model from different angles and capture little details before starting to model. This step is on pre-production, and it is commonly developed by a concept artist and not a 3D modeler. The most used software is PureRef, Adobe Photoshop, and Painter.
- **3D Modelling**. First, a basic three-dimensional shape is given to the model. After, there are two ways to continue the modeling process:
    - **Sculpting**. A high-poly model is created by sculpting the block-out, and from that model, a low-poly is extracted. Then, the high-poly is used in the low-poly as a texture. This process is useful for organic modeling. Pixologic ZBrush is the standard on the industry.
    - **Modelling**. Here, the block-out primitives are manipulated to create more complex models, but always having control on each vertex. There are many software programs to choose from, but the most used in 2021 are Maya, Blender, Cinema 4D, 3ds Max, and SketchUp.
    - **Retopology**. Is the process of simplifying the topology of a mesh into a lower resolution and simplified mesh, so it is easy to be animated, takes less memory and it is easy to calculate in real-time renders.
- **UV Mapping.** Model's surfaces are pulled apart to have it as a 2D image.
- **Texturing.** Color and other textures are applied to the model. Substance Painter Substance Designer and Quixel Mixer have become the number one programs when talking about designing textures.
- **Rigging and animation.** This step is only followed if the model has some movement. A 3D skeleton is set to the model and later it is animated with a group of controllers during a time-lapse.
- **Lighting.** Some lights are placed around the model on the scene to show better and emphasize some parts of the model.
- **Rendering.** When lights are placed, it's time to render an image to show the full potential and work of the model. Marmoset Toolbag, Keyshot, Blender,

[2] First ever 3D animation film https://www.youtube.com/watch?v=T5seU-5U0ms

Substance Painter, Maya, and V-Ray are frequently used lighting and rendering programs.

- **Compositing.** This step goes hand by hand with rendering and is usually done to show visual effects.

Note: Concepts explained before are much more developed in *2.1 Processes for the creation of 3D models.*

2.2.1 This project workflow

Having explained the industry pipeline for developing three-dimensional models, some of the most employed industry software is going to be used. First, because the author has some previous knowledge on some of the programs, so time isn't lost on learning new software as the time is limited. Secondly, as the author is a student, showing his skills in software the industry uses, will help him to have more job opportunities in the future. And finally, most of the industry software is not free and is expensive. The university gives free access to some software in this project workflow. These are the programs that are going to be used to develop 3D models:

PureRef. This program helps to see reference images in one place. It is free. It was useful to decide which modular pack environment to choose.

Blender. It is a free and open-source 3D computer graphics software. It is used for many three-dimensional things such as texturing, rigging, particle simulation, or sculpting. But in this project, it is going to be used for modeling and UV unwrapping.

Pixologic ZBrush. It sets the industry standard for digital sculpting. It enables customizable brushes to shape, texture, and paint virtual clay in a real-time environment. For some props like barrels and streetlights, this software will be used.

Adobe Substance Designer. Is an application intended for creating 2D textures, materials, filters, and 3D models in a node-based interface with a heavy focus on procedural generation, parametrization, and non-destructive workflows (*deeply explained in 2.2.2 – Node-based*).

Adobe Substance Painter. It's a tool to texture 3D assets with simple brushes to smart materials that adapt automatically to the model. Those materials can apply realistic details working on a non-destructive environment. It's the industry standard when talking of 3D texturing.

Marmoset Toolbag 3. In some cases, is good to present the developed assets to its maximum potential. It is a real-time rendering, texturing, and baking tool. When uploading the project to Unreal Engine Marketplace, it is good to have high-quality renders of the assets developed so the users can see the best results possible.

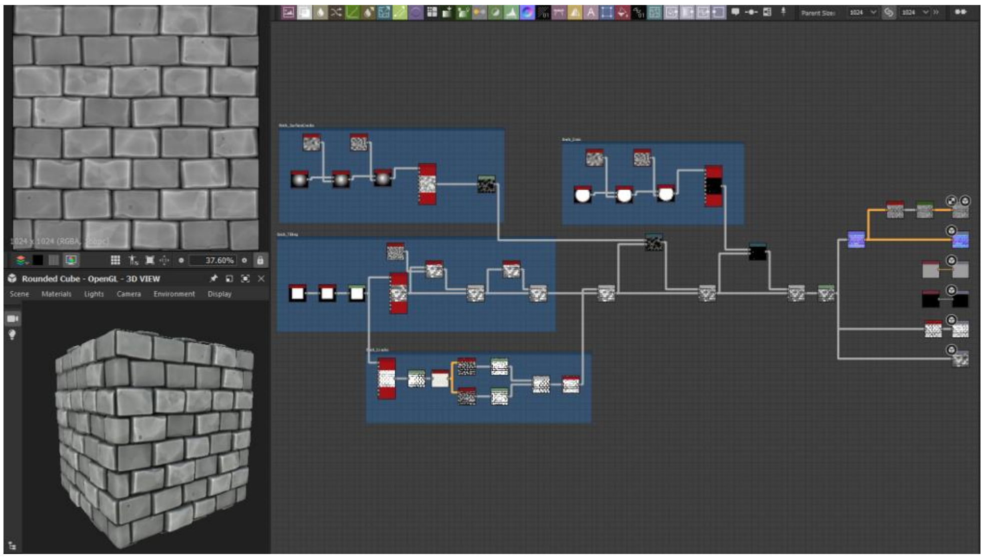
Unreal Engine 4. It is a very powerful game engine that supports demanding performance, developed by Epic Games. The asset pack will be optimized. However, it is going to be used as it is free, its marketplace hasn't got so much of stylized asset packs and it is becoming more and more popular.

2.2.2 Node-Based

As previously explained, most of the asset materials are going to be developed with Adobe Substance Designer. These materials are developed on a node-based interface.

On a nodal system, each element and effect is called a "node". These nodes are linked and represent one progression from the origin to the exit. What's good about this system is the non-destructive workflow and the capacity for great flexibility, including the ability to modify the parameters of each node.

Figure 2.2.1 *Project Stone Material developed with Substance Designer nodes*


Rounded Cube - OpenGL - 3D VIEW
Scene Materials Lights Camera Environment Display
37.60%

Substance Designer nodes can be divided into two main groups:

- **Atomic nodes.** These nodes are the lower-level most basic building blocks of Substance and if a graph is broken down to the most core operations, it would consist of these atomic nodes. The most used and useful ones are Blend, Uniform Color, Bitmap, and Levels.

- **Graph instances.** It is a graph that has been packed together and turned into a reusable, standalone resource. This is very useful as it saves a big amount of time and allows to work more efficiently. The bad part is the impossibility to inspect and change some parameters of these graphs.

2.3 Optimization

Optimization is key when developing anything for a computer. When running video games, there are lots of assets being computed at the same time. Assets need to be developed to be run on most personal computers. If those assets aren't optimized, games can have lag spikes, frame drops, freezes or it shut down in the worst scenario. Even though Unreal Engine 4 is a very powerful game engine, performance and optimization will be improved on every game asset trying not to lose quality.

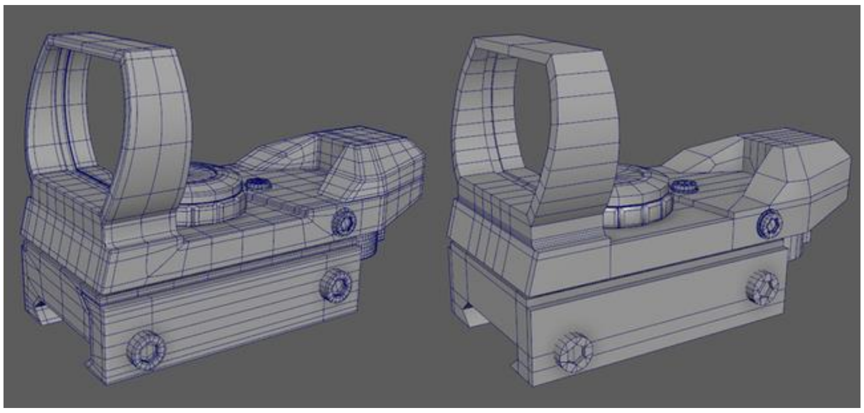
2.3.1 Retopology

In its most basic form, retopology is the modification of the topology of an object. It is the process of simplifying, in an optimized way, the quantity of polygons a 3D model has. When creating 3D models, there is a tendency to use more polygons than necessary. That's why, once the modeling is finished, it is important to do a retopology to eliminate all those unnecessary polygons.

The main use of retopology is to get a polygon mesh made up of tris and quads with fewer polys which results on a smaller file size that runs smoothly on a game engine. Through this, we recover a more efficient 3D surface that's better for texturing and animating. It can be developed by hand or with software tools.

As shown in *Figure 2.3.1*, the left gun scope has the same quality as the right one, but with less than half the polygons.

Figure 2.3.1 *High-poly vs Low-poly model*



Note: Image from Reddit post "First full high and low poly model" (spookr-, 2021)

This process is not easy and edge loops need to be preserved. Further, interior faces that will never be seen must be deleted always.

2.3.2 Bake high-poly to low-poly

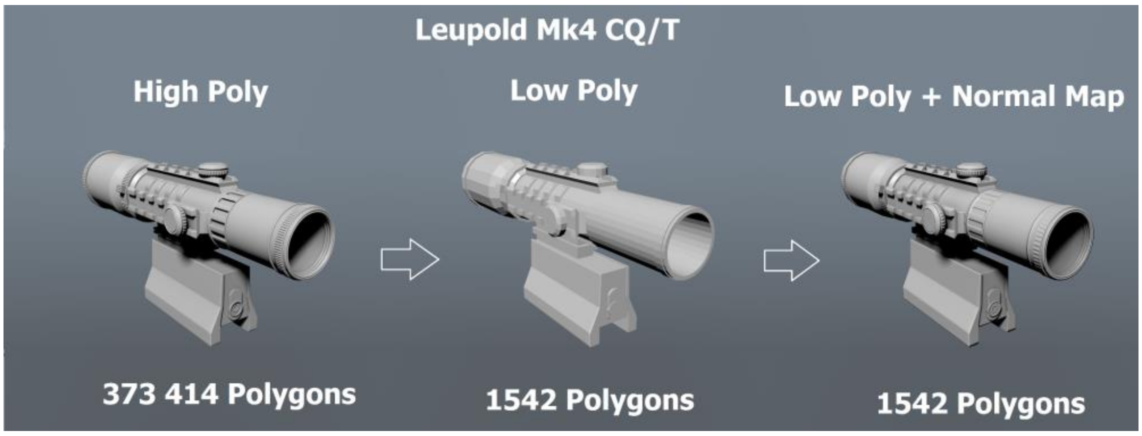
The process of baking the high-poly model to the low-poly is widely used in the game industry. It consists of baking the normal map of the high-poly to the low-poly model. The number of textures is the same on both models, so this technique is very useful since the file size is much smaller and a lot fewer polygons are computed as we are using the low-poly as if it was the high. Besides, game engines often struggle with processing higher-poly models.

To accomplish this, in this project the following steps will be done:

- A high-poly is sculpted in ZBrush.
- The model is duplicated and retopology is done to convert it to a low-poly.
- Low-poly model is exported to Blender to unwrap its UVs.
- The low-poly model is exported to Substance Painter and the high-poly model is baked into the low-poly.

As seen in *Figure 2.3.2*, a high-poly model has 373k triangles. From that, a retopology is developed to a low-poly model with 1.5k tris. Finally, normal map is baked from the high-poly to the low-poly model. This final model has the details from the high-res model with the polygons of the low-poly one.

Figure 2.3.2 *Baking high-poly to low-poly*



Note: Image from WordPress "Dancing robots blog"

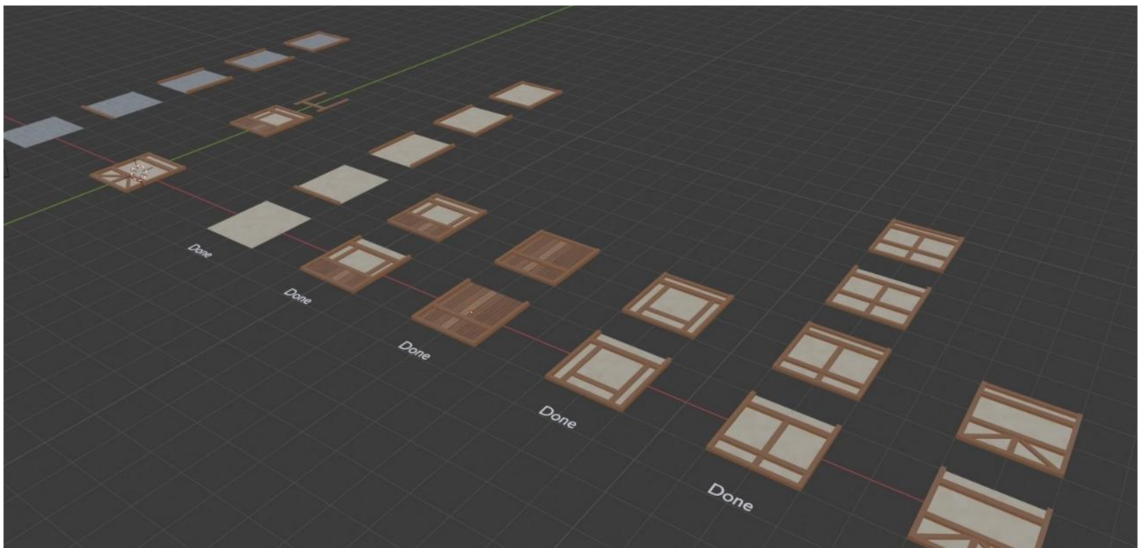
2.4 What’s an asset pack?

An asset pack is a container that holds any combination of anything that goes into a videogame - characters, objects, sound effects, maps, environments, textures, shaders, scripts, etc.

This project consists of an environmental asset pack. This type of asset pack allows the user to create his specific environment, in this case, a stylized medieval city. The game assets must have an artistic cohesion, so when all are together in the scene, they have the same style.

Besides, it is also helpful to have a wide variety of assets and variations, inside the same asset pack. It is hard to create a whole environment scene with assets from different asset packs that look good together. That’s why the assets are defined as “blocks” that allow the users to rapidly build their medieval city. Walls in *Figure 2.4.1* are walls developed for this project that will perform as blocks.

Figure 2.4.1 *Project wall blocks in Blender*



An asset pack must be original. It can be something unique that hasn’t been created or something is done before but that isn’t good enough, or is outdated, or abandoned by the developer.

2.4.1 Modularity

Modularity is a system composed of separate components (modules) that can be connected or integrated together. This allows components to be added or replaced without affecting the rest of the system. Also, it creates spaces of different scales through repetition of components and easily allows the addition or subtractions of some elements from another to build different structures.

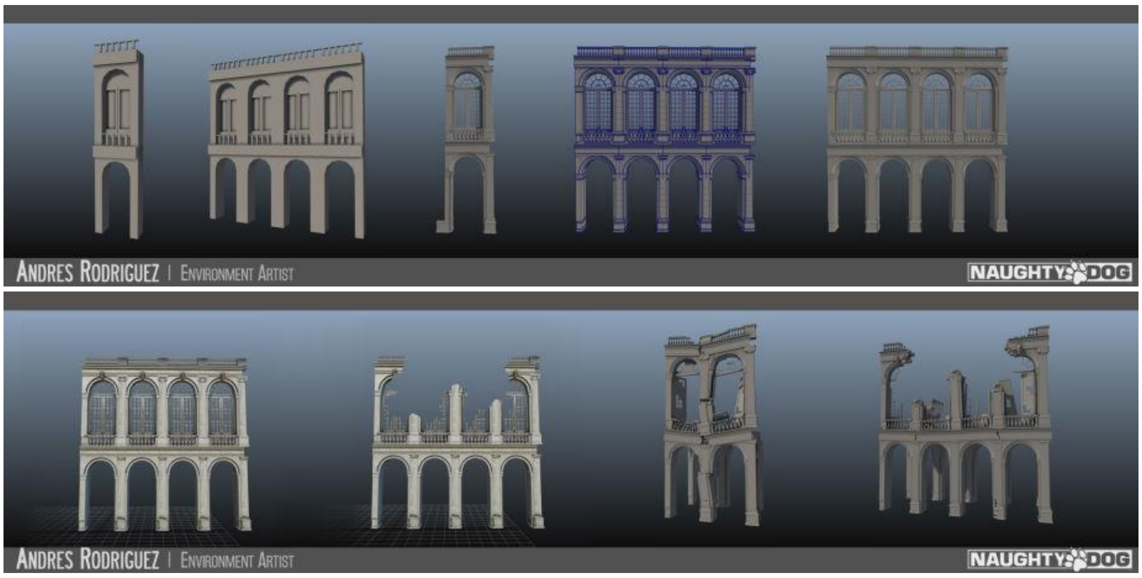
"Well-designed modular kits can greatly minimize the need for unique game assets"

Nataska Statham says this phrase in "Game Environment Art with Modular Architecture" (Entertainment Computing Volume 41, March 2022). She also presents that 3D games try to use modular architecture as an optimization technique for environment art and level design, but many of them struggle to apply it effectively.

In Scott Jones's article on "Modular Design Research in Computer Games" (2011, pages 7-11) he reviewed Epic Games UDK (Unreal Development Kit) and realized that Epic Games reuses the problem of having notorious repetition on modular assets, through the use of unique features in lighting, elevations, decals, and accessories. He also writes about how repetition on modular assets can confuse players into thinking he has already explored that section of the game.

As Andres Rodriguez says in the article "Making the Game World Come Alive" (80LV, 2016), repetitive looks can be simply avoided if assets are covered with objects like foliage. Also using decals, blends, world space overlays, and other shader techniques to break up similar surfaces. If broken versions of the same assets are introduced, this will create more visual interest. In *Figure 2.4.2*, Andres shows how he applied these techniques to some Uncharted 4 game assets.

Figure 2.4.2 *Andres Rodriguez Uncharted 4 modular assets*


ANDRES RODRIGUEZ | ENVIRONMENT ARTIST
NAUGHTY DOG
ANDRES RODRIGUEZ | ENVIRONMENT ARTIST
NAUGHTY DOG

Note: Image from Making the Game World Come Alive (80LV, 2016)

In this project, modularity is key so the users can create lots of different buildings and environments like they would be doing a puzzle. Building structures are thought to have the same dimensions and style so when added all together, the scene looks realistic and well-assembled. If the modularity is badly developed, the asset pack is useless as individual walls can't form any type of logical scenario.

2.5 Stylized art

Stylized art means something represented in a non-naturalistic conventional form.

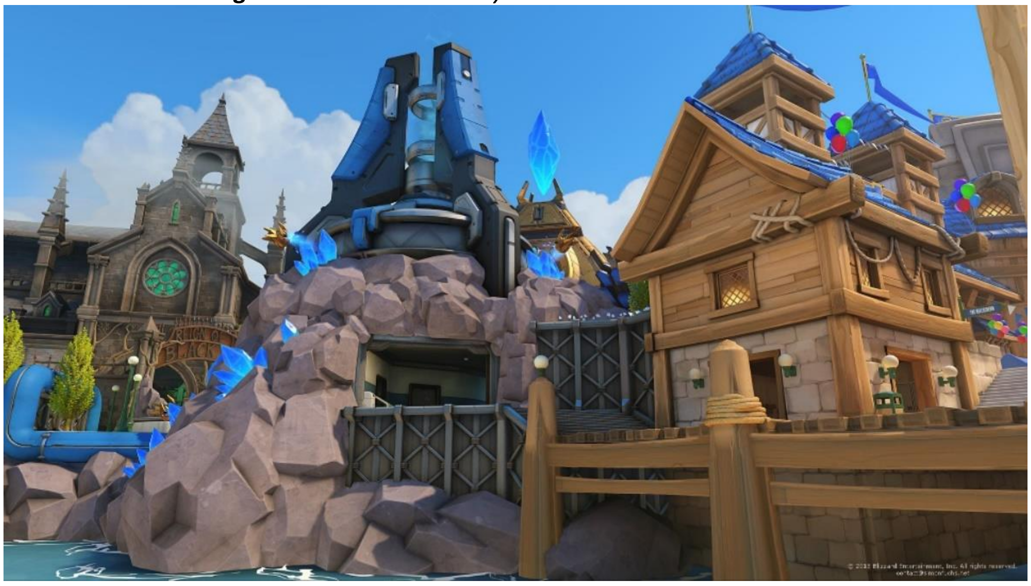
Kim Aava says in his article "Realism vs. Stylized: Technique Overview" (80LV, 2017) that with stylized art, the artist is free to play with shapes and colors, and exaggerate or remove details to enhance the look and feel in any direction. It breaks the illusion of reality as it wouldn't be viewed as 'realistic'.

Kim says, "stylization refers to a visual depiction, which represents an object without a full attempt and accurate representation of an object's realistic appearance". This means that stylized art is described as an art style that adheres to realism in detail rather than simplifications.

Stylized graphics are so varied and there isn´t a single guideline on how to create stylized art for games. However, Kim divided the stylized art into two categories which the author agrees with. These categories are:

- **Over-exaggerated.** This art focuses on larger details and shapes rather than small details. In this style, the shape is exaggerated. Depending on the importance of the asset is supposed to convey, it will or won't be enlarged. The video game Overwatch uses this kind of art.

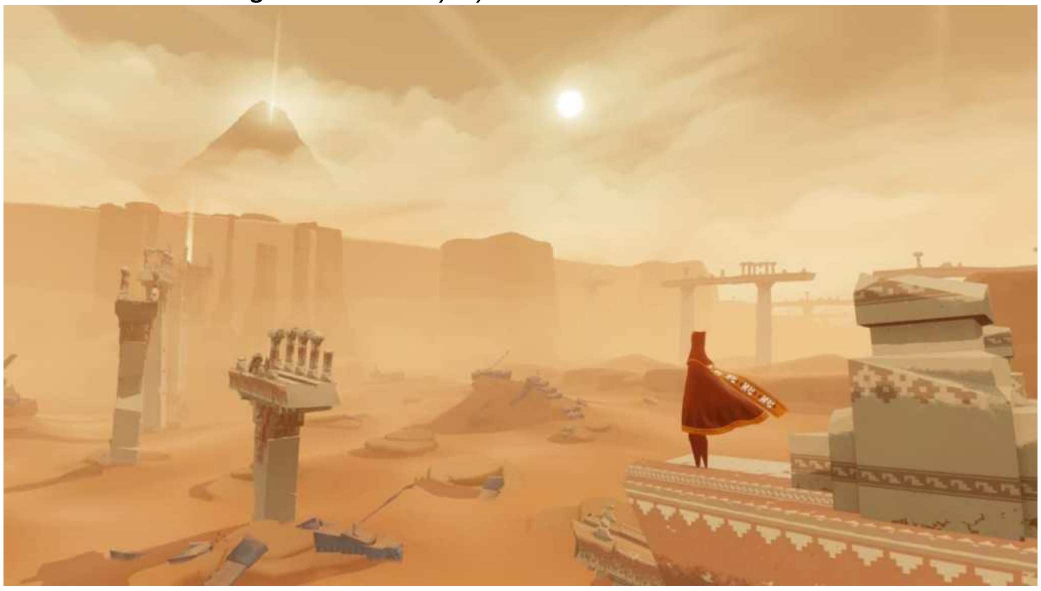
Figure 2.5.1 *Overwatch stylized art environment*



Note: Image from Simon Fuchs's post on ArtStation

- **Minimalistic.** It plays on simplicity, with very few details in color and shape. It usually lacks on medium and small details, normal maps and it uses a single-color map. It assembles to low-poly art, but it is simply painted. An example could be Journey video game.

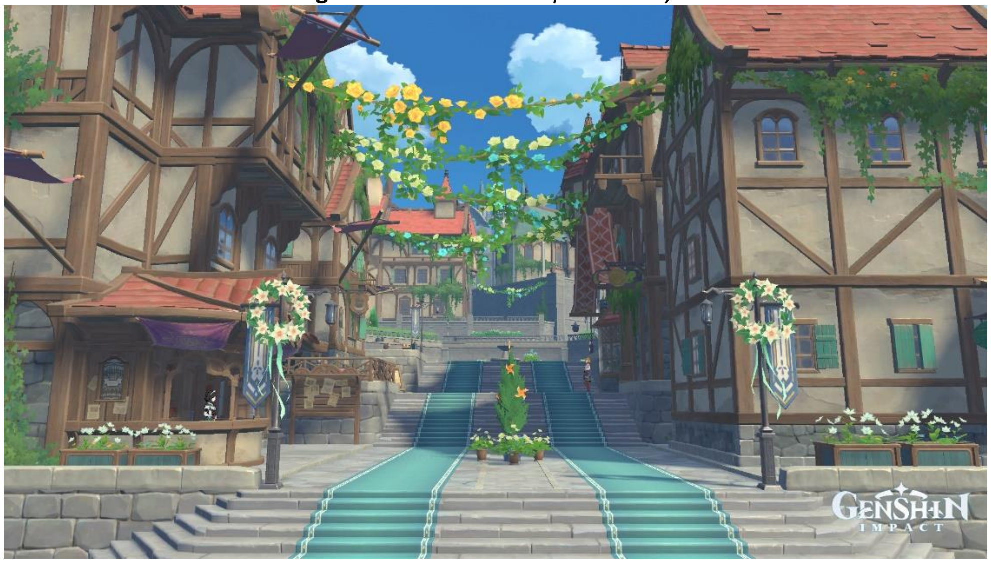
Figure 2.5.2 *Journey stylized art environment*



Note: Image from Journey game on Epic Games

As said before, there aren't standard guidelines that define a stylized art style rather than to be no realistic. In this project, the art style will be between over-exaggerated and minimalistic. Art won't be exaggerated in form; it will be closer to realistic. However, textures won't be minimalistic with a single color but with hand-painted textures. The developed art style in this project will resemble the game *The Legend of Zelda: Breath of the Wild* or *Genshin Impact* shown in *Figure 2.5.3*.

Figure 2.5.3 *Genshin Impact art style*



Note: Image from Genshin Impact Wiki - Fandom

2.6 Market analysis

The final purpose of this project is to develop the asset pack to publish it in Unreal Engine Marketplace. As there is no previous knowledge on how to develop this kind of project as well as how is this online market, it is needed a study of different packs.

Ten of the most relevant stylized asset packs from Unity Asset Store and Unreal Marketplace with the same medieval thematic of this project have been analyzed.

Table 2.1 *Market analysis*

ASSET PACK NAME (LINKS)	Company	Unique meshes	Price per mesh	Total price
Stylized Medieval	Manufactura K4	250	0,18 €	**43,77 €**
SUNTAIL - Stylized Fantasy Village	Raygeas	235	0,26 €	**61,64 €**
Stylized Medieval Village Pack	Emj3dArt	200	0,13 €	**26,79 €**
Stylized Medieval Village	AleksandrIvanov	212	0,08 €	**16,42 €**
Stylized Medieval Modular Asset Pack	Shiab Miah	46	0,24 €	**10,93 €**
Modular Medieval Castle Town	Sidearm Studios	203	0,43 €	**87,64 €**
Medieval Fantasy Pack	Pointy Circle	149	0,18 €	**27,38 €**
Modular Medieval Village Fantasy	Sidearm Studios	148	0,30 €	**43,81 €**
Modular Medieval Town Village	Sidearm Studios	63	0,35 €	**21,90 €**
FANTASTIC - Village Pack	Tidal Flask Studios	247	0,18 €	**44,66 €**
	AVERAGE	**201,5**	**0,21 €**	**35,58 €**

As shown in Table 2.1, each package has a link to the marketplace, the company that made it, the number of unique meshes it has, the price per each mesh, and the total price[3] of the pack.

The prices vary, so an average price per mesh of 0,21€ has been stipulated. Besides, asset packs have around 200 unique meshes, so users can create complete environments with their package.

In this project, it has been calculated to be around 150 unique meshes. If this is the case, this project asset package will be around 30€. However, the quality may not be the expected, so the price will be lower.

[3] [2] All total prices are consulted on 21st March 2022.

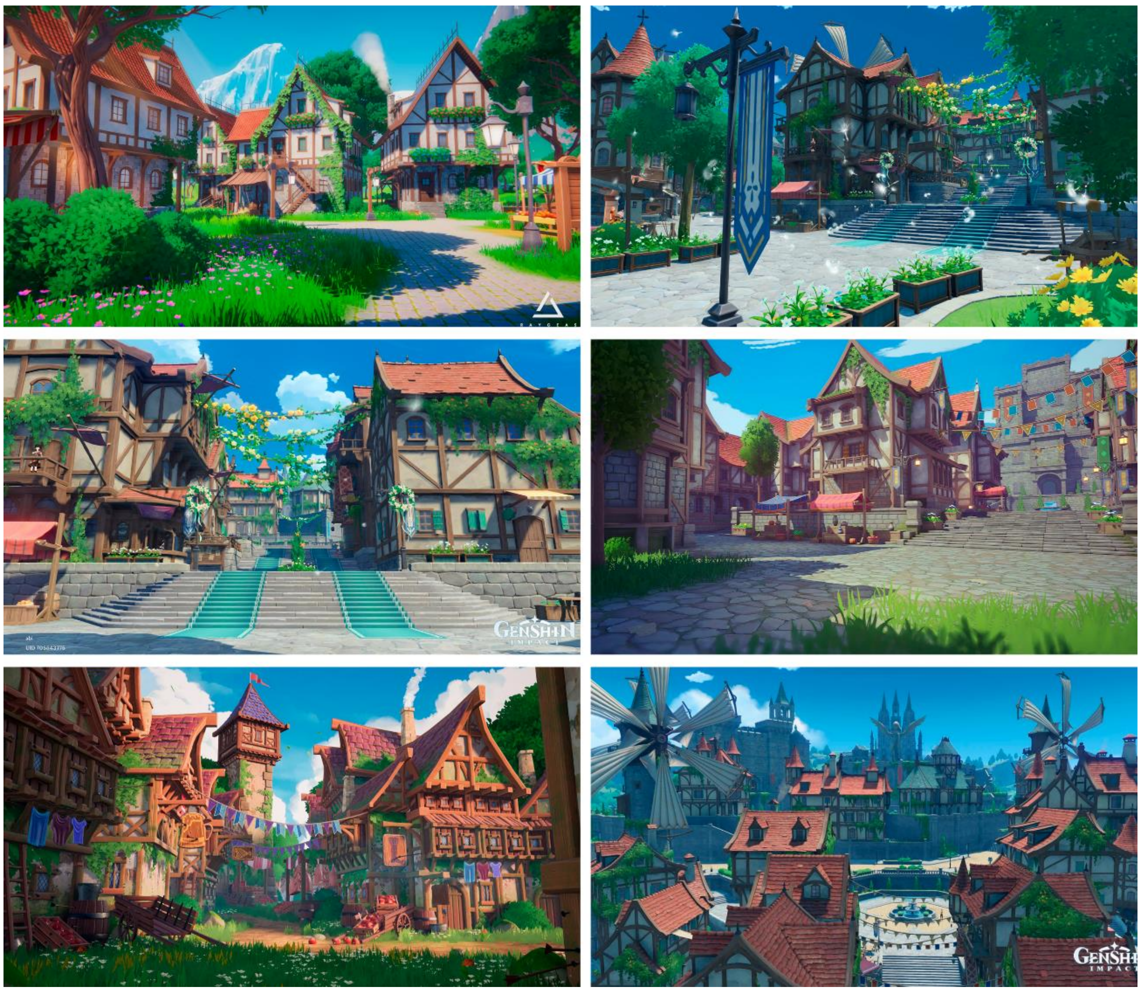
2.7 Desired art

Assets packs in the 2.6 Market Analysis section are all medieval and stylized, however, not all stylized styles are the same. Having so many asset packs as a reference will help to select the best parts of each and combine them into one pack.

Having this into consideration, the desired art for this project is intended to combine anime art style with elements from real-world cultures, in this case, the medieval one. Genshin Impact, which was born inspired by The Legend of Zelda: Breath of the Wild, is a perfect example of the art that it wanted to achieve. The city of Mondstadt (in Genshin Impact) will be the main reference point both for its art and modular structure.

Keeping this in mind, here there are several references of the desired art style for a better understanding.

Figure 2.7.1 *Desired art references*



Note: Images extracted from Genshin Impact, UE4, and 80LV articles.

3. Management and Planning

Management and planning are fundamental to develop a good project. To do so, it is key to create several realistic goals and detail a plan of action for meeting those goals. The first step was identifying the specific objectives, so good planning is made, and realistic goals can be achieved.

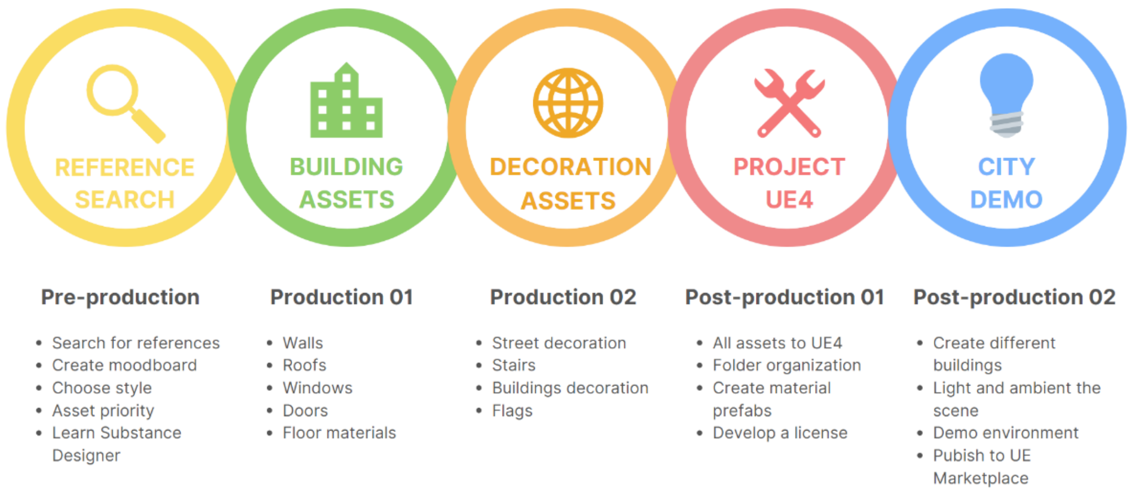
To achieve this, the project has been divided into five milestones according to the production phases. This helps to better plan the project since it is not possible to move on to another milestone without having successfully completed the previous one. This will set a good pace of work and will help in not having to make decisions in the middle of the project, which is vitally important since it may develop into a loss of time.

As shown in *Figure 3.1*, the first step is pre-production. Here is where the style of the asset pack will be defined as well as which assets to do. Also, we haven't started to learn the basics of Substance Designer since October, so we have enough knowledge to start building materials.

Afterward, the production process will be started. On production 01 the building assets will be built as they are the base for the asset pack. Once these assets are finished, production 02 phase will start and decoration assets, that are less important, will be developed.

In the end, post-production milestone will take place. All assets and materials will be done, and it will be time to organize assets into a project in Unreal Engine 4 during post-production 01. Additionally, different buildings will be created to create a demo environment, and at last, be able to publish an asset pack with a good result to the Unreal Engine Marketplace.

Figure 3.1 *Project Milestones*



Pre-production
- Search for references
- Create moodboard
- Choose style
- Asset priority
- Learn Substance Designer

Production 01
- Walls
- Roofs
- Windows
- Doors
- Floor materials

Production 02
- Street decoration
- Stairs
- Buildings decoration
- Flags

Post-production 01
- All assets to UE4
- Folder organization
- Create material prefabs
- Develop a license

Post-production 02
- Create different buildings
- Light and ambient the scene
- Demo environment
- Pubish to UE Marketplace

3.1 Gantt Chart

To manage and plan the entire project a Gantt Chart has been created. As there will be a heavy workload, this will help to have everything more organized as well as set a pace of work.

The chart has been subdivided into the milestones explained in *Figure 3.1.* If the milestones are done within the stipulated deadlines, assets can be made with more dedication and the final result will be better.

First of all, as seen in *Figure 3.1.1*, it is needed to have good references, asset styles, and goals, before jumping into the production of the project on the 21st of February, and that's when assets will start to become developed. Walls will take a good percentage of the production time since there are quite a few, but also the first Rubric (*Figure 3.4*) will be being drafted at the same time.

Figure 3.1.1 *Gantt Chart for Pre-Production and Production 01 (January to April)*

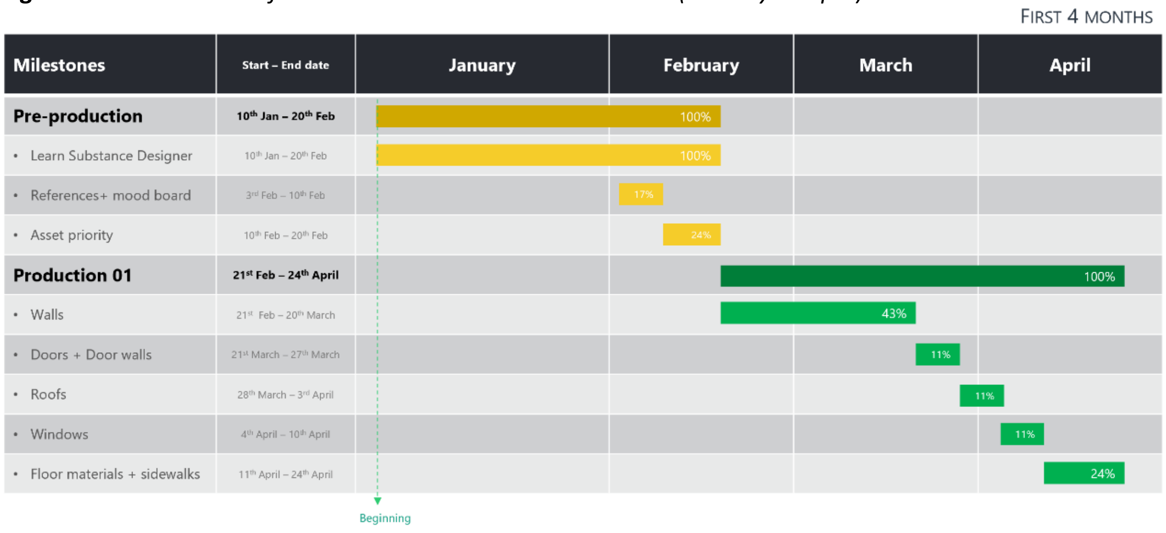
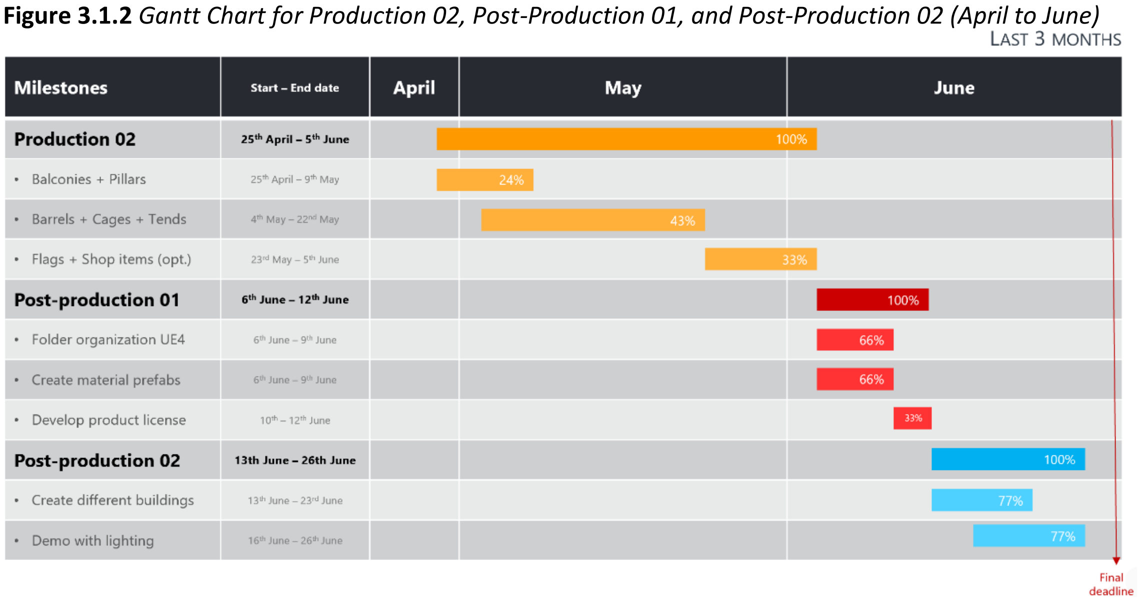


Figure 3.1.2 *Gantt Chart for Production 02, Post-Production 01, and Post-Production 02 (April to June)*



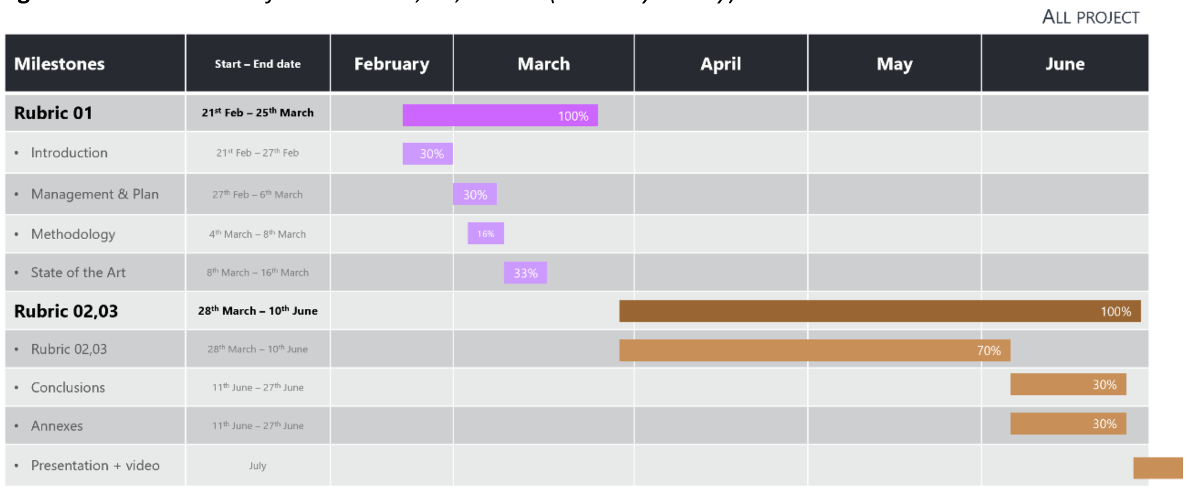
Milestones	Start – End date	April	May	June
Production 02	**25th April – 5th June**			100%
• Balconies + Pillars	25th April – 9th May		24%	
• Barrels + Cages + Tends	4th May – 22nd May		43%	
• Flags + Shop items (opt.)	23rd May – 5th June			33%
Post-production 01	**6th June – 12th June**			100%
• Folder organization UE4	6th June – 9th June			66%
• Create material prefabs	6th June – 9th June			66%
• Develop product license	10th – 12th June			33%
Post-production 02	**13th June – 26th June**			100%
• Create different buildings	13th June – 23rd June			77%
• Demo with lighting	16th June – 26th June			77%

As it appears in *Figure 3.1.2*, some decoration assets in Production02 are optional if time is running out. They are not key for the project, even though they will help the environment look better.

Despite it being shown that there is no creation of an environment and buildings until the Post-Production02, during the whole project assets will be tested in Unreal Engine 4 to check everything is working and looking correctly.

The project will finish in June where post-production milestones need to be polished and functional.

Figure 3.1.3 *Gantt Chart for Rubrics 01, 02, and 03 (February to July)*



Milestones	Start – End date	February	March	April	May	June
Rubric 01	**21st Feb – 25th March**		100%			
• Introduction	21st Feb – 27th Feb	30%				
• Management & Plan	27th Feb – 6th March		30%			
• Methodology	4th March – 8th March		16%			
• State of the Art	8th March – 16th March		33%			
Rubric 02,03	**28th March – 10th June**					100%
• Rubric 02,03	28th March – 10th June					70%
• Conclusions	11th June – 27th June					30%
• Annexes	11th June – 27th June					30%
• Presentation + video	July					

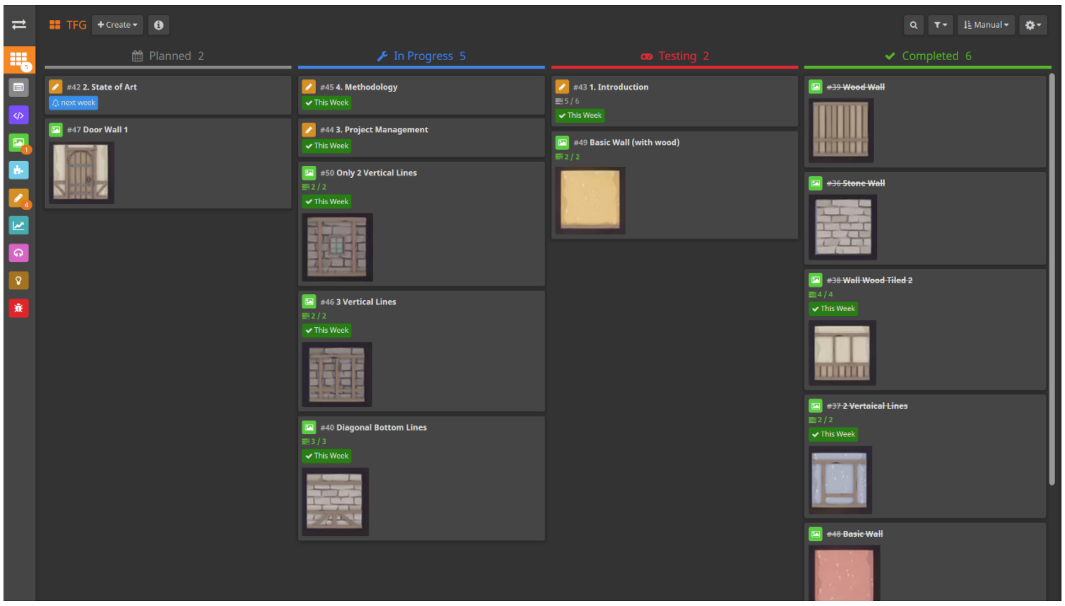
3.2 Tools and Procedures

Gantt Chart will help to manage tasks on defined periods of time. It is useful to have a general overview of how the project is going and what is next. However, inside each task, there are always subtasks.

To manage these subtasks Gantt is not as flexible as other platforms and that's when HackNPlan will take place. HackNPlan is a project management tool heavily inspired by agile methodologies.

This tool will help in managing short-term and weekly tasks that are derived from larger tasks. It is very helpful as it allows to have a general overview of how the week is going with just a quick view.

Figure 3.2.1 *HackNPlan board*



3.3 Validation Tools

As the project is individual, the author will be the one testing in Unreal Engine 4 if assets are correct, work as expected, have a similar style, and look good with the whole environment.

However, there is need for external opinions and testers. Marc Ripoll Tarré, who is the director of this project, has a wide knowledge of 3D art and design. He will be giving feedback and helping when it may be necessary.

Additionally, university colleges, that have good notions of 3D modeling and Unreal Engine 4, will be giving their point of view on how they see the project and how to improve it. To do so, individual interviews with colleges will be captured via online meetings.

3.4 SWOT

To identify what areas are being well performed, which are holding the project back, or what the competitors could exploit, a SWOT analysis is made. This analysis, shown in *Table 3.1,* will show the strengths, weaknesses, opportunities, and threats of the project.

Table 3.1 *SWOT Analysis*

	Helpful	**Harmful**
Internal Origin	**Strengths** • Optimized asset pack • Ready to use • Texture size selection	**Weaknesses** • Limited resources • Limited production time
External Origin	**Opportunities** • Community likes and purchases the asset pack • Very few well-developed asset packs with this thematic • Marketplace brings visibility	**Threats** • Big competitors in the market (SyntyStudios, JustCreate, …) • Better options for the same or less price.

3.5 Risk and Contingency Plan

During the project, some problems and obstacles will appear. They will delay the workflow and may make it hard to deliver all the milestones on time. If during the project some of these issues are encountered, here there is a list of how to deal with them.

3.5.1. Time Constrains

One of the biggest issues of the project is time. There are four months to develop this project and meanwhile, the author is coursing other subjects, which have exams and projects that also require some time.

To avoid some delivery problems, the Gantt chart must be followed up to date. Also, tasks can be started if all the other tasks of that week have been finished before time. Another solution is to reduce the number of decoration assets leaving the environment with the most important ones.

3.5.2 Estimated Quality

Assets may not have the same quality or style. If an asset or various assets look different from the rest of the environment, they will stand out negatively as the scene won't look realistic.

To solve this, testing sessions will take place in Unreal Engine 4 where assets will come together to see if they look good. In addition, professional artists along with some colleges will give their opinion and point of view on how to improve them.

3.5.3 Optimization

When developing the demo scene or creating some buildings, there might be lag spikes caused by bad optimization of the textures and meshes of the props. This will make the asset pack cumbersome to use or even unusable.

For this not to happen, assets will be constantly tested in Unreal Engine 4. Not only assets individually, but lots of them being rendered in the engine at the same time, so performance can be quickly tested.

3.6 Initial Cost Analysis

Every project has several costs. To calculate the cost of this project some things have been taken into consideration:

- The project will be developed by a single junior 3D artist. On average, working 4 hours a day, so the salary[4] has been estimated to be half time.
- Peripherals (mouse, keyboard, monitors, computer, etc.) and software have a depreciation of 3 years as shown in *Table 3.2*. These components have a depreciation[5] of 40% each year. The balance shows the total value of one year and the result shows only five months' cost.
- The total working time is 5 months, so the costs will be calculated in relation to this period.
- Electricity has been calculated for a computer and two screens, which approximately consume 150kWh working 4 hours each day of the week.
- Five software programs will be used but only two of them are free. Inside the Substance collection, there is Substance Painter and Substance Designer.

Table 3.2 *Depreciation Formula*

Depreciation Form

Object	**Original Cost**	**Residual Value**	**Time (years)**	**Balance**	**RESULT**
Mouse	100 €	40%	3	21,60 €	**9,00 €**
Heaphones	110 €	40%	3	24 €	**10,00 €**
Monitors	500 €	40%	3	108 €	**45,00 €**
Keyboard	100 €	40%	3	21,60 €	**9,00 €**
Computer	1.000 €	40%	3	216 €	**90,00 €**
				TOTAL	**163,00 €**

Depreciation = (Original Cost x Residual Value)
Result = Original Cost - Depreciation

Table 3.3 *Electricity Calculus*

Electricity	Consumption	Work hours	Price per kWh*	**TOTAL**
Gaming PC + 2 Monitors	150 kWh	± 4	0,71533 €	**13,05 €/month**

Note: Electricity price of 8th March 2022 (Spain)

[4] Salary data extracted from Indeed.es https://es.indeed.com/career/dise%C3%B1ador-3d/salaries?from=top_sb (Consulted on March 8, 2022)
[5] Depreciation rates from TaxAdda.com https://taxadda.com/depreciation-rates-income-tax-act/ (Consulted on March 8, 2022)

Table 3.4 *Direct and Indirect Costs*

	February	**March**	**April**	**May**	**June**	**TOTAL**
Direct Costs	1.016 €	1.016 €	1.016 €	1.016 €	1.016 €	**5.079 €**
Personal	898 €	898 €	898 €	898 €	898 €	**4.490 €**
Salary	898 €	898 €	898 €	898 €	898 €	**4.490 €**
Software	85,28 €	85,28 €	85,28 €	85,28 €	85,28 €	**426,40 €**
Zbrush	36,89 €	36,89 €	36,89 €	36,89 €	36,89 €	**184,45 €**
Substance 3D Collection	48,39 €	48,39 €	48,39 €	48,39 €	48,39 €	**241,95 €**
Blender	- €	- €	- €	- €	- €	**- €**
Unreal Engine 4	- €	- €	- €	- €	- €	**- €**
Hardware (depreciation)	32,60 €	32,60 €	32,60 €	32,60 €	32,60 €	**163,00 €**
Computer	18,00 €	18,00 €	18,00 €	18,00 €	18,00 €	**90,00 €**
Peripherals	14,60 €	14,60 €	14,60 €	14,60 €	14,60 €	**73,00 €**
Indirect Costs	54,07 €	54,07 €	54,07 €	54,07 €	54,07 €	**270,35 €**
Internet	41,02 €	41,02 €	41,02 €	41,02 €	41,02 €	**205,10 €**
Electricity	13,05 €	13,05 €	13,05 €	13,05 €	13,05 €	**65,25 €**
TOTAL COST	1.070 €	1.070 €	1.070 €	1.070 €	1.070 €	**5.350 €**

In conclusion, the total cost of five months of developing this project will cost 5.350€.

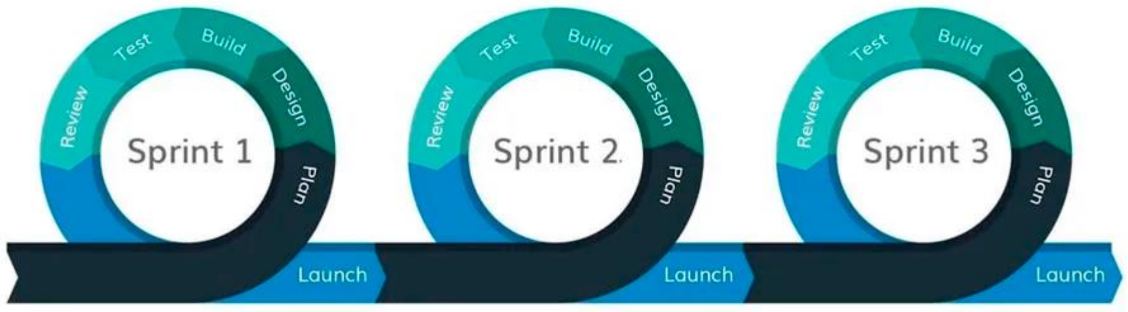
4. Methodology

The project is divided in four stages represented in the Gantt chart. Inside each milestone, there are a set of tasks that also have sub-tasks. To achieve good results and complete every task on time the Agile methodology is perfect for this project.

The Agile methodology simply means "continuous incremental improvement through small and frequent releases". This allows to break the project into several phases and work into continuous stages. Agile method is based on adaptive planning with early deliveries on a continual improvement with flexible responses to change requirements and the problems to be solved.

The project is divided into sprints. On each sprint, a planning phase will be the starting point. After this, assets will be designed and later build. And before starting another sprint a testing and review phase will take place where the assets will be tested in Unreal Engine 4 with its textures, proportions, and some lighting.

Figure 4.1 *Agile Methodology*



6

If any sprint is not finished on time all the project will be affected as the following sprint won't start. That is why it is so important that the dates of the sprint are perfectly followed.

This tool will help on not moving to the next tasks until the current one is tested and reviewed to be completely functional.

To assign these tasks within each sprint HackNPlan is going to be used (explained in Tools and Procedures). This tool allows to create tasks and organize them in columns in a very visual and schematic way. These columns are: To do, In Progress, Testing, and Done. As HackNPlan uses Agile methodology these columns are equal to Plan, Design, Test, and Launch in *Figure 4.1.*

[6] Image extracted from https://www.datavaults.eu/extracting-software-requirements-the-agile-way/

5. Development

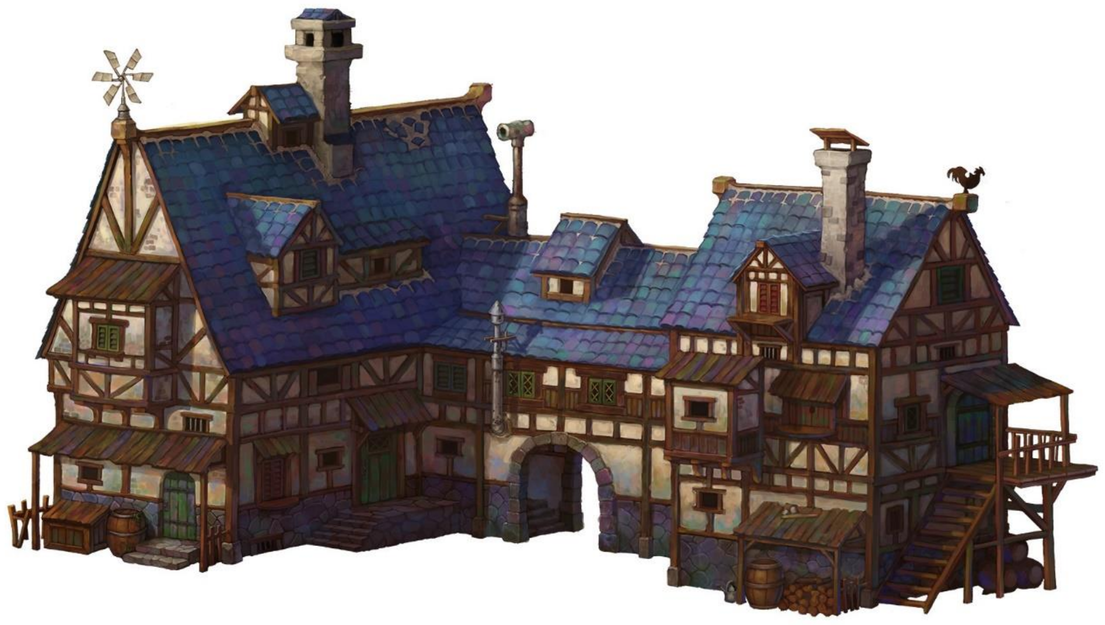
Once the pre-production milestone has been finished, the development of the project did take place. The mission was to follow all the required deadlines for each task. However, some tasks required more dedication than others as well as some problems and mistakes have appeared. During this section, the author tries to give a general view of what he encountered while developing this project, explaining both the positive and negative points, what he did good and bad and what could have been done differently.

5.1 Wall materials

One of the tasks on the pre-production milestone was to learn how to develop procedural materials in Adobe Substance Designer.

Following several tutorials and trying Substance Designer on several personal projects, it is time to create some materials. As Walls are the first thing to develop, we need to divide how many materials we will need. Medieval house walls are simple, their base is usually made of clay, cobblestone, or wood panels. Then, they have different wood stick patterns that make the wall more resistant.

Figure 5.1.1 *Sketch of medieval house*



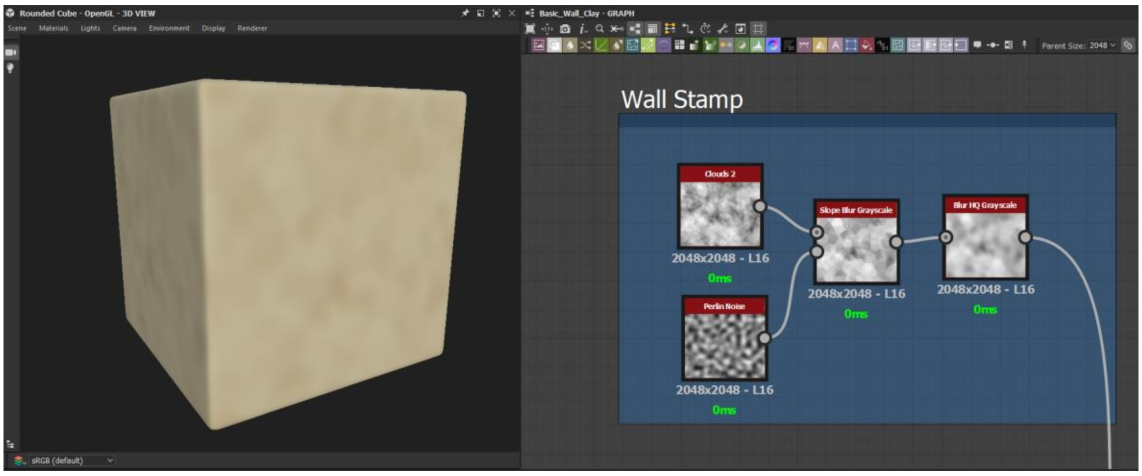
*Note: Image from Minkuk HAN post "Medieval House (2017)" on Art*Station

As seen in *Figure 5.1.1*, wall materials can be divided as clay, cobblestone, wood panels, and simple wood.

5.1.1 Clay

It is the simplest texture of all the project. It is developed by combining one texture generator (clouds 2) and a Perlin noise with different blurs.

Figure 5.1.2 *Clay Material*

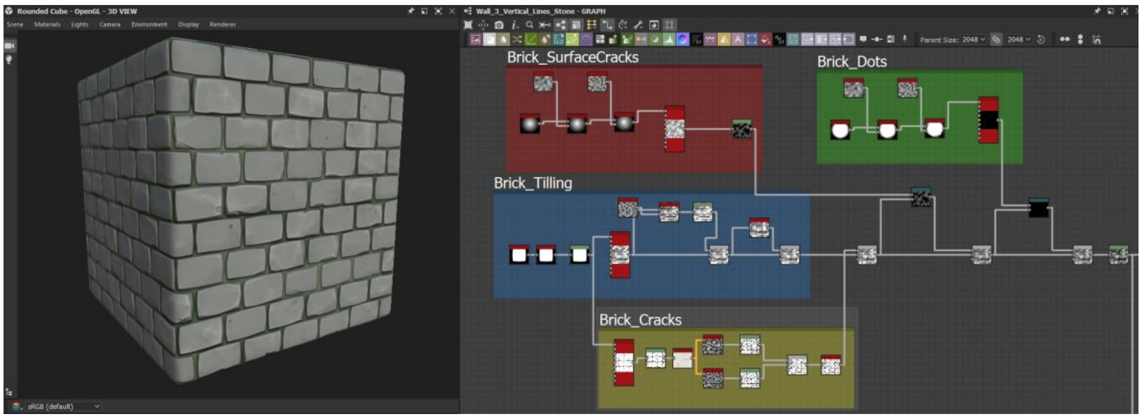


5.1.2 Cobblestone

It is created by four main blocks:

- **Brick Tilling.** With one rounded square polygon and a tile sampler, a simple 6 x 10 square tile is created. Then, with a Perlin noise, we add irregularities to each rock's edges.
- **Brick Cracks.** Here with different Flood Fills we give random angles to each stone and add random cracks.
- **Brick Surface Cracks.** We add irregularity to each brick surface with a set of noises and blurs.
- **Brick Dots.** As well as on surface cracks, some dots are added to random bricks.

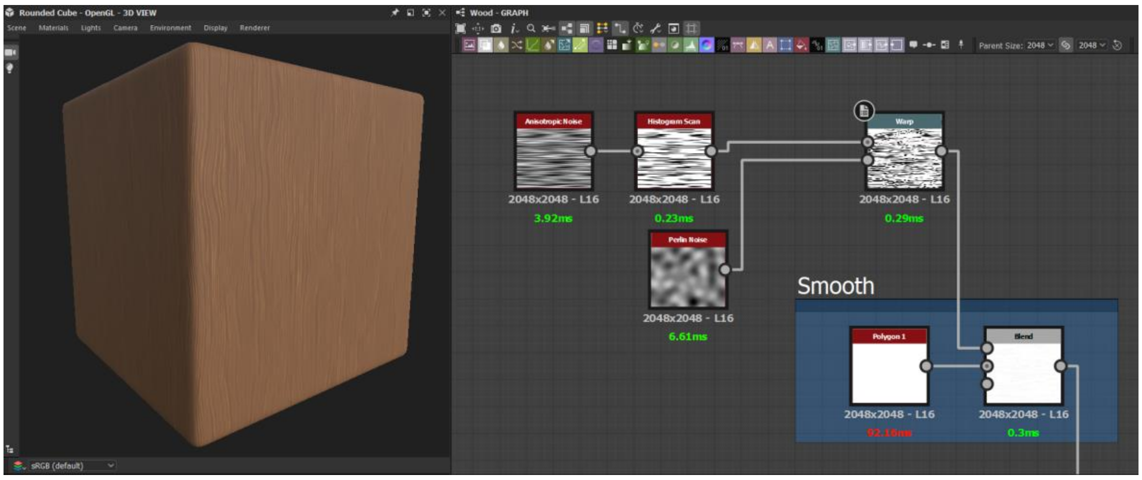
Figure 5.1.3 *Cobblestone Material*


Rounded Cube - OpenGL - 3D VIEW
Wall_3_Vertical_Lines_Stone - GRAPH
Brick_SurfaceCracks
Brick_Dots
Brick_Tilling
Brick_Cracks

5.1.3 Wood

It is created by warping an Anisotropic noise and a Perlin noise. Then, with a blend node and a white node, the result is smoothed.

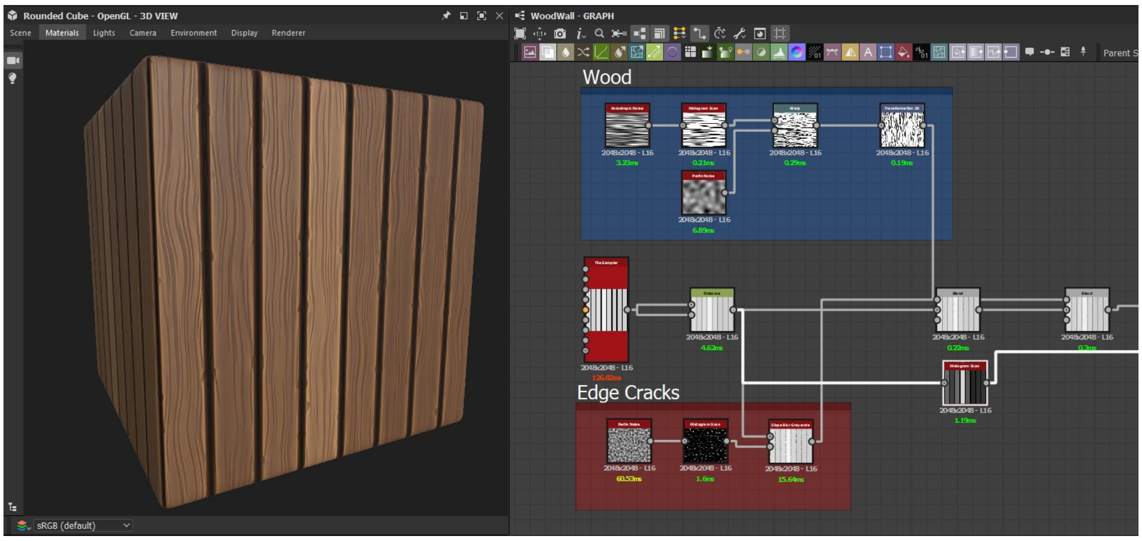
Figure 5.1.4 *Wood Material*



5.1.4 Wood Wall

Taking the wood as a base, we create eight vertical tiles. On each tile edge, irregularities are added to simulate broken wood. Finally, in the tile sampler, we add some color random, so there is a little variation in the color of each tile.

Figure 5.1.5 *Wood Wall Material*

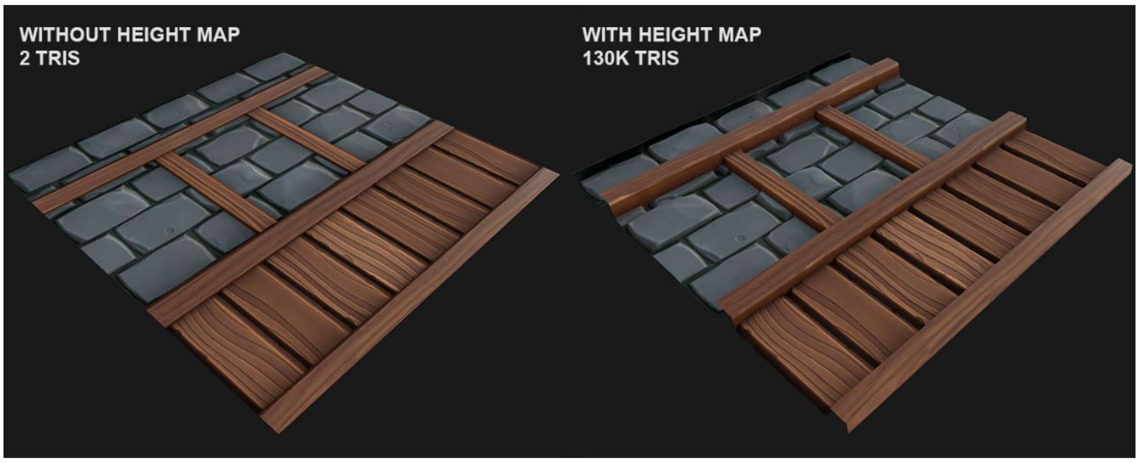


5.2 First problem

Now that all wall materials are developed and after some polishing, the result is the expected, we start to build walls on Substance Designer. After a week of hard work, all the wall patterns that were planned to be on the asset pack are finished. It was the time to export them from Substance Designer and import them with its correspondent materials to Unreal Engine 4. This is where the first problem of the project appears.

The author hasn't got into account that all the textures have been developed using a height map. So, when trying to import the texture to UE4 and adding it to a simple plane, which has two tris, the material is shown flat as seen in the left wall in *Figure 5.2.1*.

Figure 5.2.1 *Without height map vs with height map*

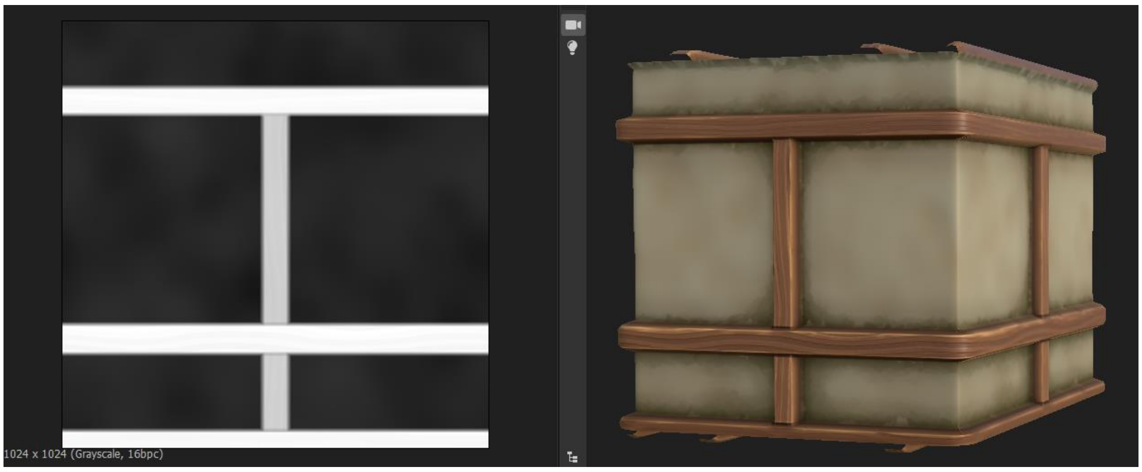


5.2.1 Height map

To understand which was the problem, first, we need to have a basic understanding of how does height map works.

A height map is a grayscale image that represents displacement. Black represents the minimum height and white the maximum. Having this in mind, as shown in *Figure 5.2.2*, on a simple wall, clay would be black (no displacement), and the wood rectangles would be white (maximum displacement).

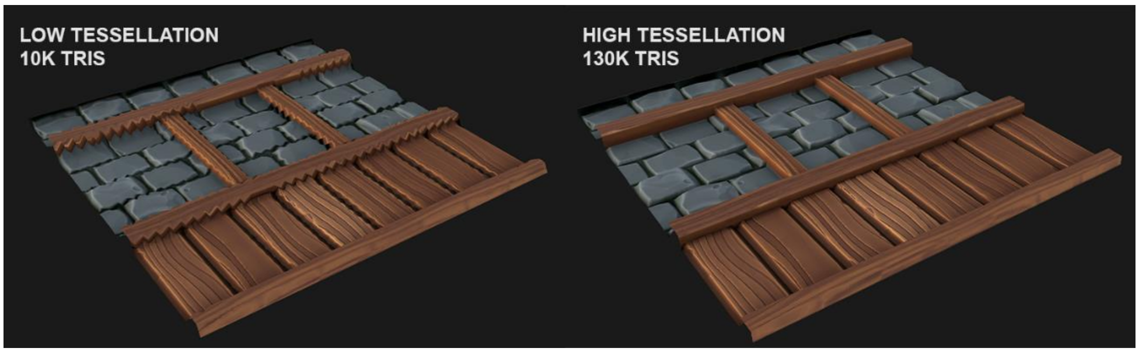
Figure 5.2.2 *Example of height map displacement*



In Substance Designer, height map can be rendered with Tessellation. And this is where the problem takes place. Tessellation subdivides the geometry procedurally on the GPU, which adds new vertices between the existing vertices. To see the texture with good resolution, we should have the plane divided into thousands of triangles, which isn't optimal for videogame projects. The rendered model in *Figure 5.2.2* has more than one hundred thirty thousand triangles.

One possible solution was to use a low tessellation factor, so the geometry of the plane will be divided into just thousands of triangles. But it isn't until polygons are small that the height map starts looking good. In *Figure 5.2.3,* it is shown the same wall with different tessellation factors. On the left wall, the tessellation factor isn't set very low, and the wall still looks very bad and triangulated where the displacement is high.

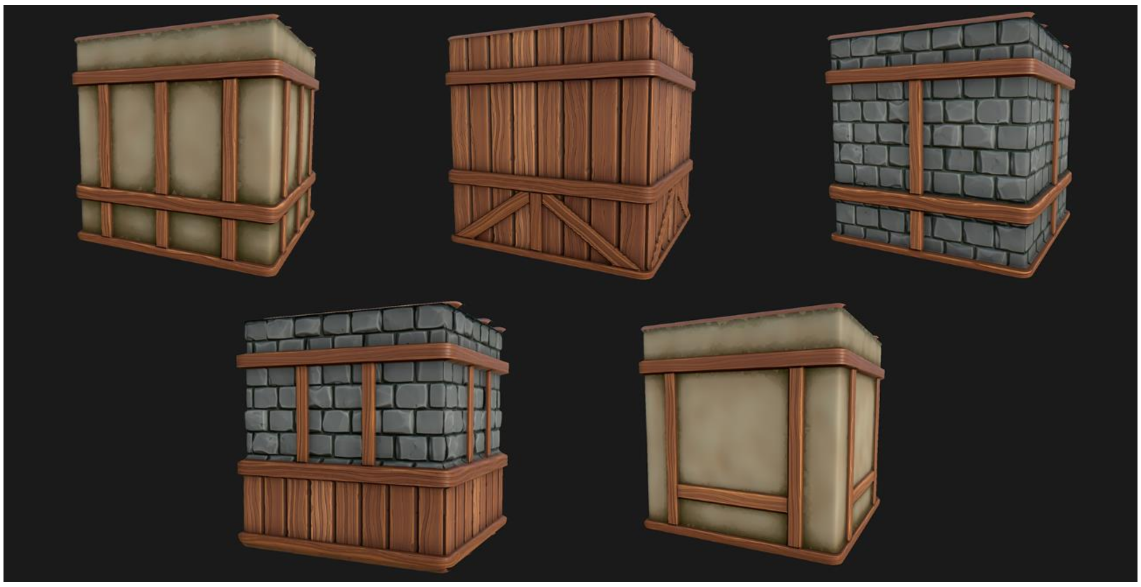
Figure 5.2.3 *Low vs High tessellation factor*



Having tried different tessellation factors and realizing height map was not optimal for real-time videogame computations, more than seventeen walls developed in Substance Designer were useless for the project. Some of them are shown in *Figure 5.2.4*.

From here, the project workflow changed into modeling the walls in Blender, doing their UV mappings there, and assigning corresponding textures to each part of the wall.

Figure 5.2.4 *Different walls developed in Substance Designer*



5.3 Starting to 3D model

Before starting to explain each asset group (walls, roofs, stairs, doors, etc.), there are several things that these assets have in common when modeling them as well as when doing their UV mapping. These things may not happen to all assets, but they do in most cases. So, in order not to repeat the same explanation repeatedly, it will be taken into account that these properties affect all assets.

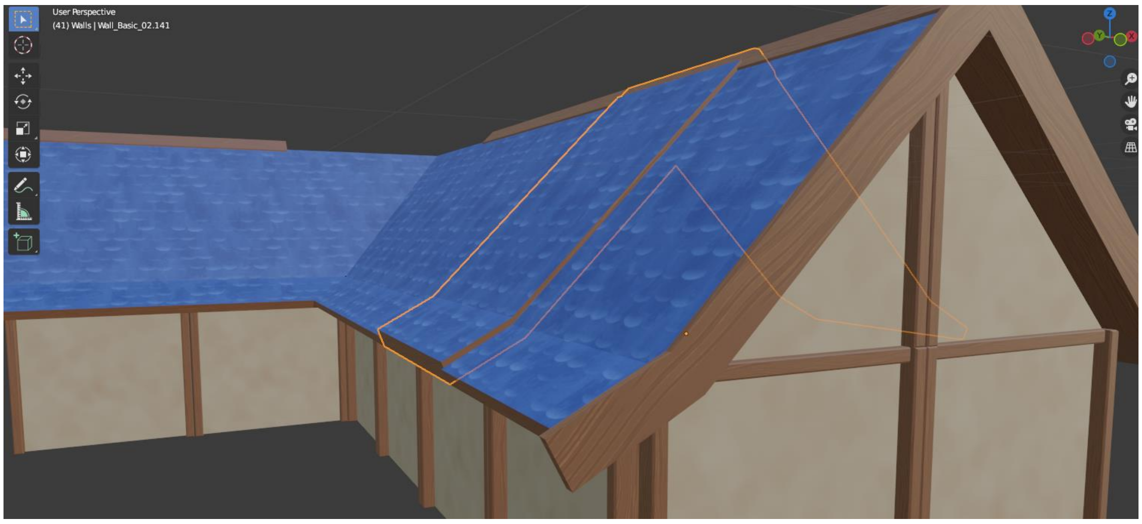
5.3.1 Fit perfectly

This development part is one of the most important ones. Studying how assets will be fitting together and how the user can build so many different types of buildings, with different sizes, patterns, and shapes, is the key point for a modular asset pack.

Almost every asset needs to be modular. Modularity, which is largely explained in point *2.4.1 Modularity*, stands for having building blocks that can fit together to form environments. This modularity needs to be studied in most of the asset groups, so when trying to place several pieces together, they fit perfectly.

This may happen on roofs. Roofs do need to have perfect continuity. If the width of a roof does not match the other roof pieces, the whole set will look bad. This occurs in *Figure 5.3.1,* where every roof fits perfectly except the one highlighted and breaks continuity.

Figure 5.3.1 *Example of modular misfit*



5.3.2 Optimization

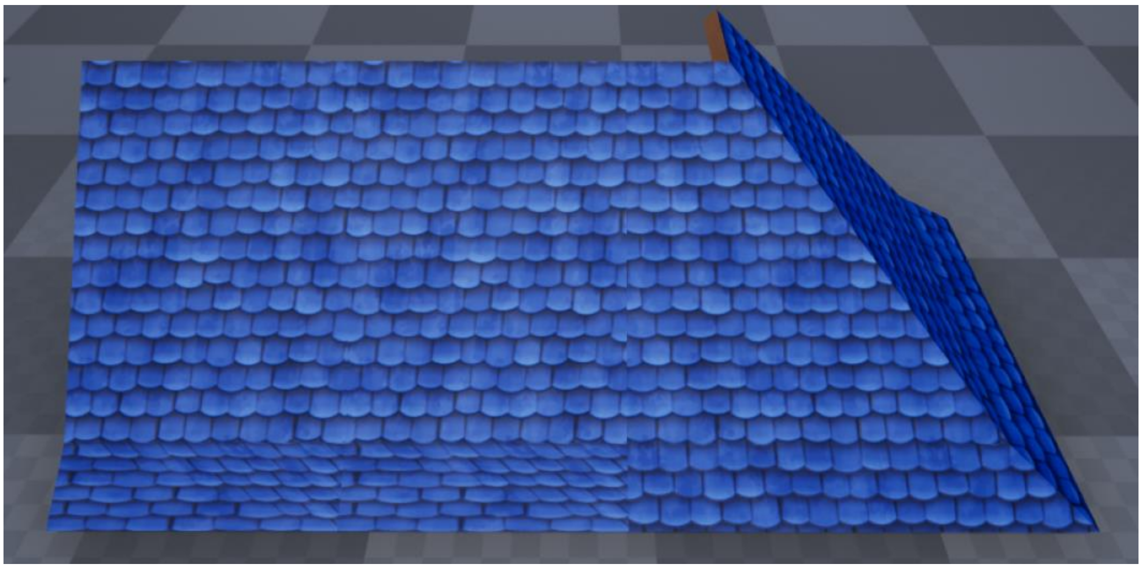
Subsequently, assets shall be optimized. When modeling assets, it is tented to use more polygons than is finally necessary. All these polygons need to be corrected or deleted.

We will need to see if an asset has interior faces. If it does, we will delete them, as they aren't going to be seen and are also computed. Then, retopologizing the assets that can have fewer triangles without losing any quality. Always trying to have polygons with three or four vertices, as some programs can't compute five or more vertices faces.

5.3.3 UV continuity

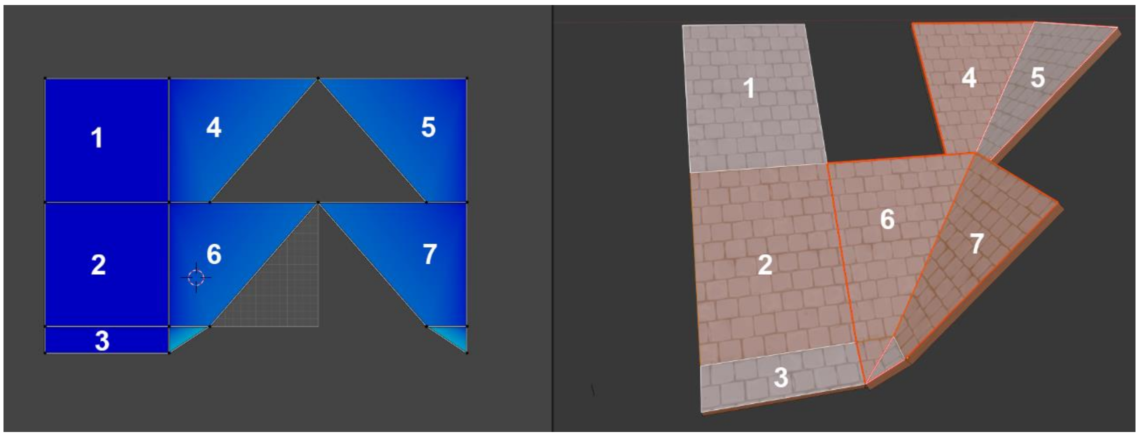
In most of the assets, UV mapping is developed so there is not a jump between textures. As well as in *5.3.3.1*, if textures don't have continuity, the whole set will look bad. When doing the UV mapping, UVs must be wrapped correctly so when placing assets together, they fit perfectly. In *Figure 5.3.2* one of the right roof textures is poorly displayed, while all the other ones are correct.

Figure 5.3.2 *Example of UV mapping poorly displaced*



Roofs are the perfect example where UVs need to be perfect as no separation between them can simulate or fake any distortion. UVs need to be placed correctly so all the texture has the same size on all objects. For instance, in *Figure 5.3.3*, all cobblestone material in roofs (placed for better visualization) has the same square stone size and it continues smoothly from roof to roof. In this case, UV configuration is well-executed, so UVs don't look stretched or have different magnitudes.

Figure 5.3.3 *Good UV configuration on UV continuity*



Note: This UV placement is for representative purposes only. Roof UVs are placed correctly on a 2K UV Tile.

5.4 Walls

Having all wall materials developed and knowing that Substance Designer walls aren't suitable for the project, walls are going to be modeled in Blender. They are the main part of building a house, that's why they are the first asset group to be modeled.

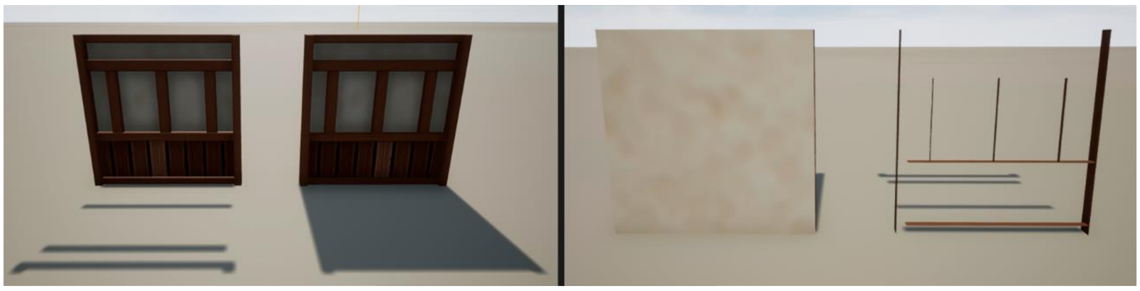
As most of medieval town walls were designed with triangular and rectangular patterns the modularity of this project's walls can be divided into the same shapes. These shapes make it easy for the user to build houses since putting them together is very intuitive and straightforward.

Before starting to develop a wide variety of walls, the first step is to create a simple base to start trying how modularity is going to be approached. As previously explained in *2.5 Stylized Art* the dimensions of the assets are going to be close to reality.

To develop this base wall, first, a simple plane with an established dimension is needed. This plane will have a width and a height of three meters, which is an approximate real measure. So, when exporting the wall to Unreal Engine, there is no need to scale each wall.

Also, there is need for a second parallel plane. This second plane is moved a little from the base plane and it is flipped so the wall has a normal-axis in both directions. If this second plane was not placed, the wall would be invisible from behind. This wouldn't be a problem since the interior of the houses isn't done in this project. However, light would pass through the wall, making it not having shadow from behind. Furthermore, it would be harder for any user of the asset pack who wanted to do the interior of the houses. This can be seen in *Figure 5.4.1.*

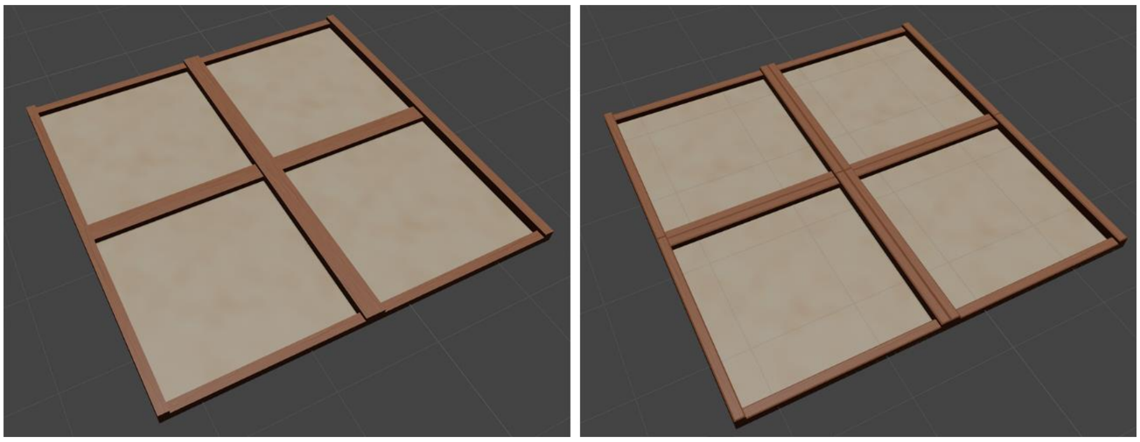
Figure 5.4.1 *Wall with vs wall without flipped plane*



From here, the wall's wood frames will be created with simple cubes at first. Vertical wood frames are bigger than horizontal ones, so it breaks continuity and gives a better 3D perspective. However, when putting some walls together, as seen in *Figure 5.4.2* it is seen that there is no difference between one wall and the other.

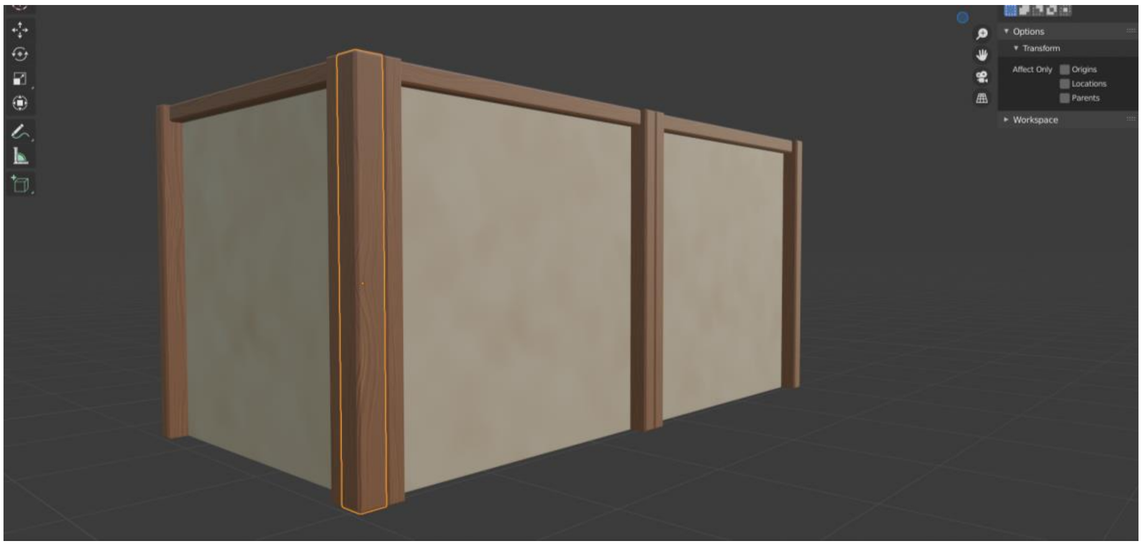
To solve this problem, a Bevel modifier is applied to each top edge of each wooden cube. As the modifier name itself indicates, the Bevel modifier bevels the edges of the mesh it is applied to. This allows a clear difference to be observed between different walls despite forming part of the same structure.

Figure 5.4.2 *Wall without vs wall with Bevel modifier*



Walls are finally looking good when they are placed together. However, when trying to build a close the walls doing a square, walls create unfilled space in the corners. To solve this small problem, a wood piece that works as a column can be placed if the user wants it with the filled or unfilled space. This wood piece (*Figure 5.4.3*) is only beveled in on two faces since if it had all four faces beveled, it would go from having 28 triangles to 44 for a very small piece.

Figure 5.4.3 *Wood column for wall corners*

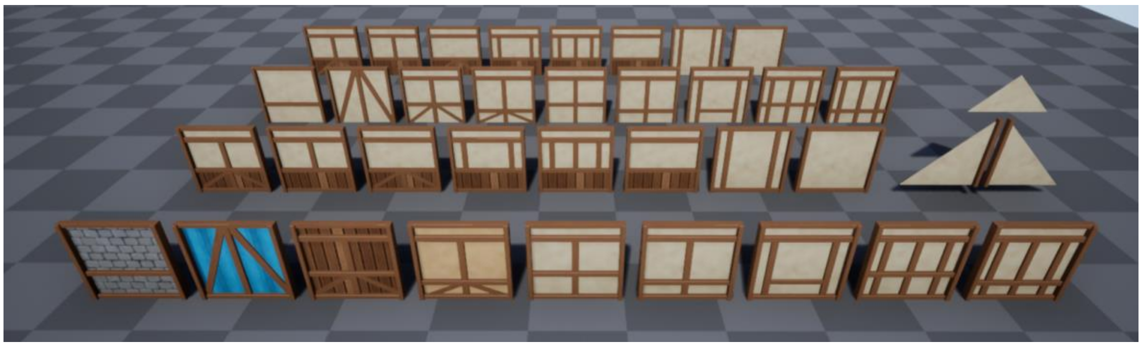


Once the base wall is well constructed, it's time to build several different wall patterns. As they have been developed in a modular way, each wall can always combine with the others. When finishing to develop every wall, with all their correct UV maps and checking that everything works as expected, it has been tested if walls would look better without the lower wood part. Since both ways seem correct and do not affect modularity, it has been left to the users to decide which wall they want to use. **Framed walls** are the ones fully framed on four sides whereas **unframed walls** are open at the bottom.

In total, there are 17 unique types of walls (6 composite and 11 simple), 10 unique door frame walls (5 rounded and 5 rectangular), and 3 triangular roof walls. To all this, it must be added the walls that do not have the lower wooden part. This leaves a **total of 47 walls**, as shown in *Figure 5.4.4.*

To conclude with this part, each wall has been developed with a clay texture. Even so, the users can displace any material they want on Unreal Engine 4 (stone, wood, wood panels, clay) on the wall base, as well as on the walls that have wooden panels below. The UVs have been set to work with any material and look correct.

Figure 5.4.4 *Project Walls*



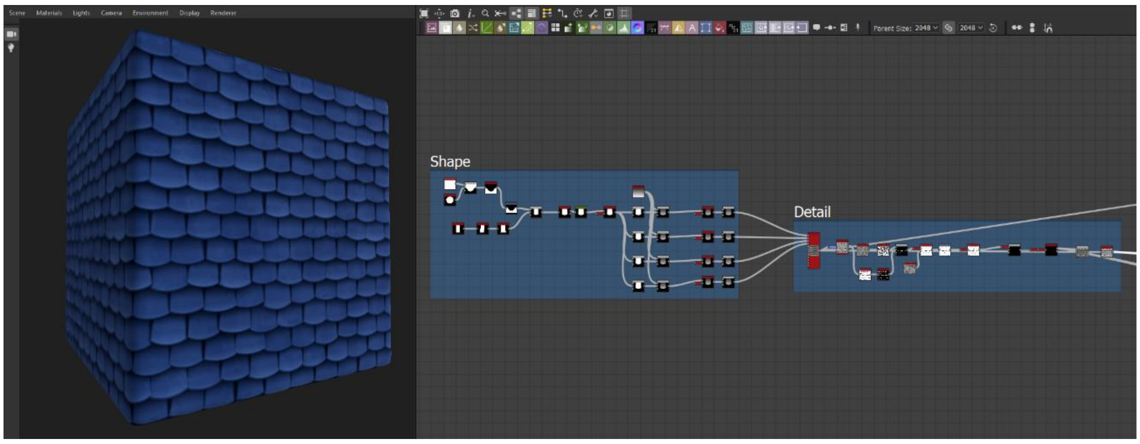
5.5 Roofs

As within the walls, the first thing to do is the roof materials. Roofs are divided into two different materials: roof tiles and wood. Having already created a wood material for the walls (*5.3.1 Wood*), only a roof tile material is going to be needed. However, if all roofs look the same, the environment is going to look repetitive and flat. That's the reason why two roof materials are being created.

5.5.1 Stone tile roof material

First, four different tile shapes are created, each one with a Grayscale Linear to give them depth. They are very similar to each other, but with varied sizes. The shape of these tiles is wider and more squared than the clay type. Then, through a Tile Sampler, we randomize the position and the rotation of all tiles, and finally, dirt and grunge details are given along with the color.

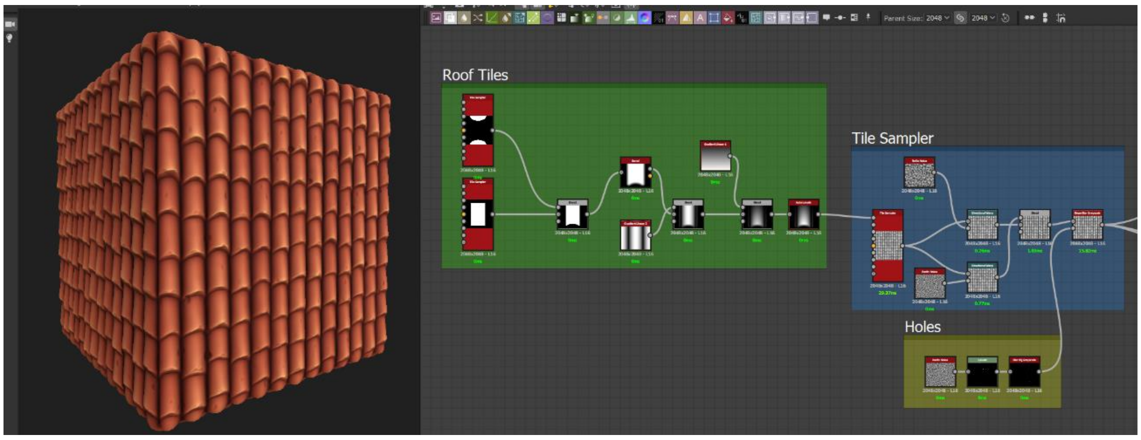
Figure 5.5.1 *Stone tile roof material*



5.5.2 Clay tile roof material

These roof tiles are convex, with the shape of a barrel. To develop them, the shape is created, and then a Gradient Linear to give depth. After that, they enter through a Tile Sampler which randomly moves up and down each line column of roof tiles. Finally, erosion is added to each bottom edge tile and as well as several holes to the whole roof.

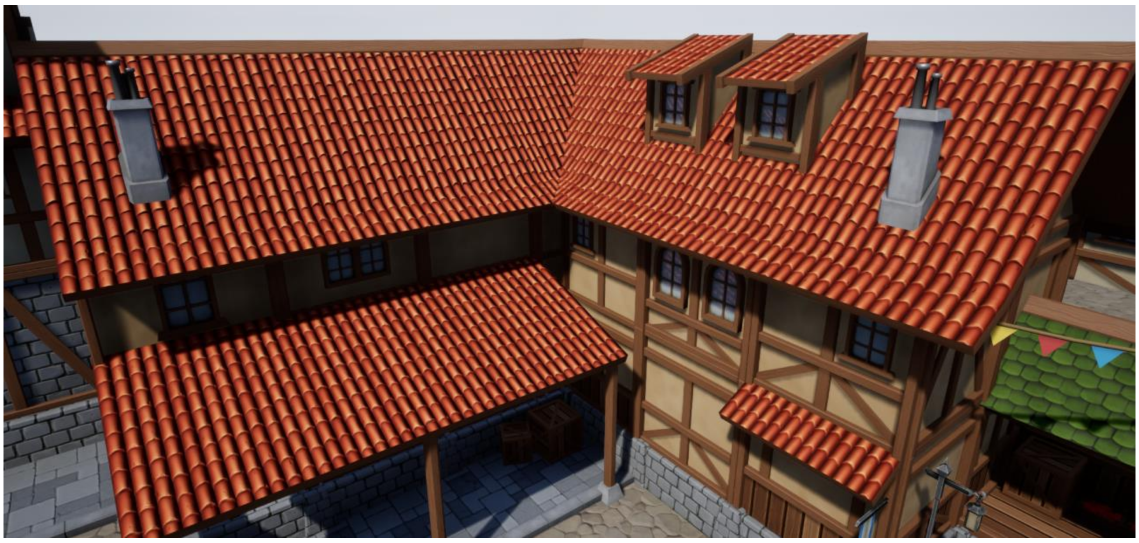
Figure 5.5.2 *Clay tile roof material*



5.5.3 Roofs modeling

Roofs are the most difficult modular part of the whole project. What some asset packs do is to have preestablished roofs that the user can mount on top of the house walls. Nevertheless, this type of roof methodology closes a lot of creativity range to the users since the options which houses can be built are greatly reduced. For this reason, roofs are also modular in the project asset pack, so the users can create every house wall base they want without any limitations.

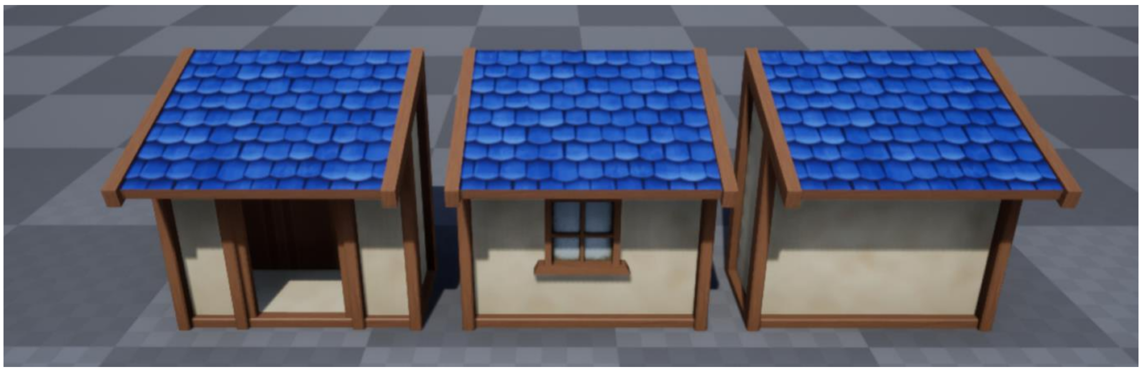
Figure 5.5.3 *Example of house modular roof*



The type of roofs that can be created are open gable, box gable, dormer, and shed, among others.

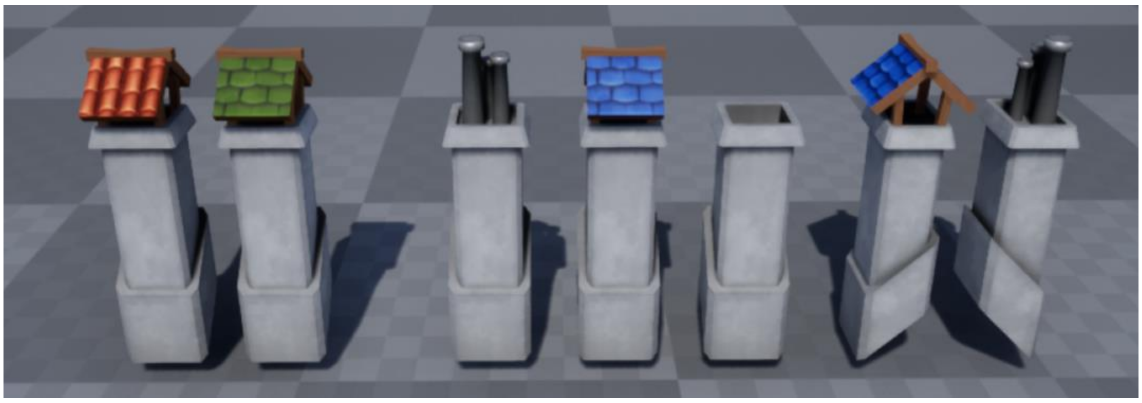
After much research and many failed prototypes, the project ended up having a modular roof. In addition, wall extensions with roof parts that come out of the basic structure, shown in *Figure 5.5.4,* have also been created to give more diversity to the environment scene.

Figure 5.5.4 *Wall extensions with roof parts*



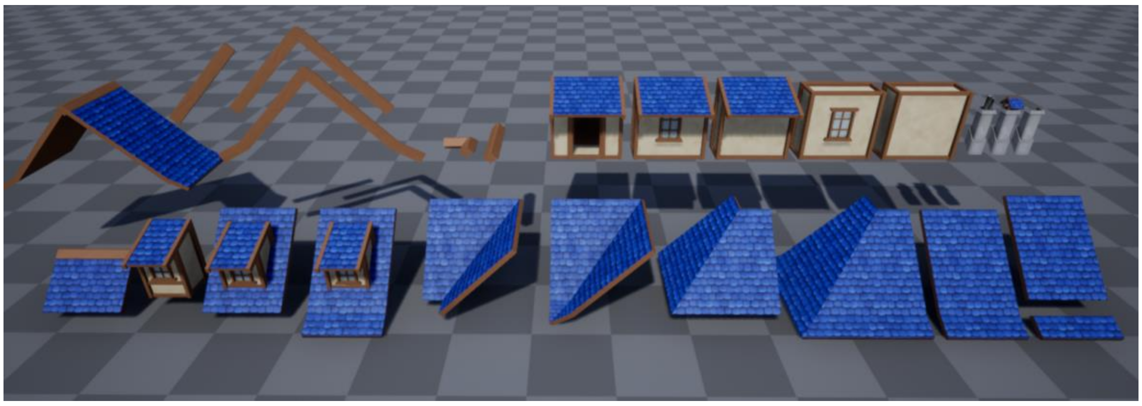
To give more variation to the scene, three types of chimneys have been developed. All of them aren't attached to roofs and can be placed anywhere. Additionally, one of them has a roof part than can be modified with any roof material, so it fits the roof style.

Figure 5.5.5 *Chimney types*



To top it off, some wood parts have been built up, so roof intersections can be covered up. This house structure ended up having all the different props seen in *Figure 5.5.6.*

Figure 5.5.6 *Project Roofs*

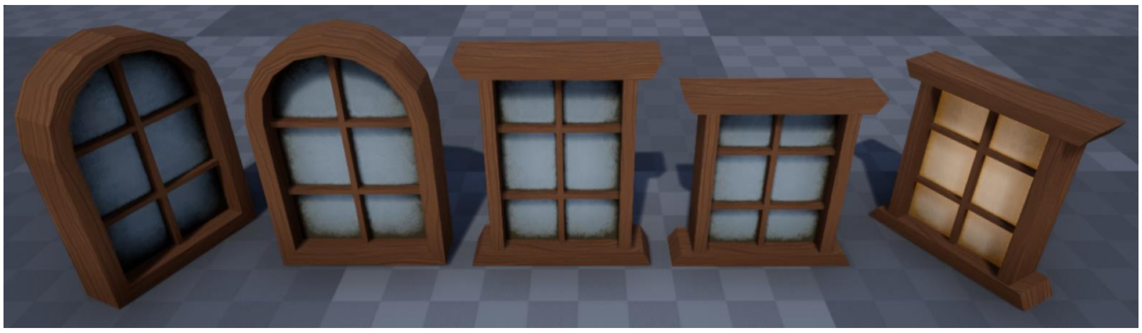


5.6 Windows

Unlike the development of roof assets that took a long time, windows have been generated very quickly. There are three windows, two of them are rectangular and the other is circular. The number of triangles is between 50 and 75 for window.

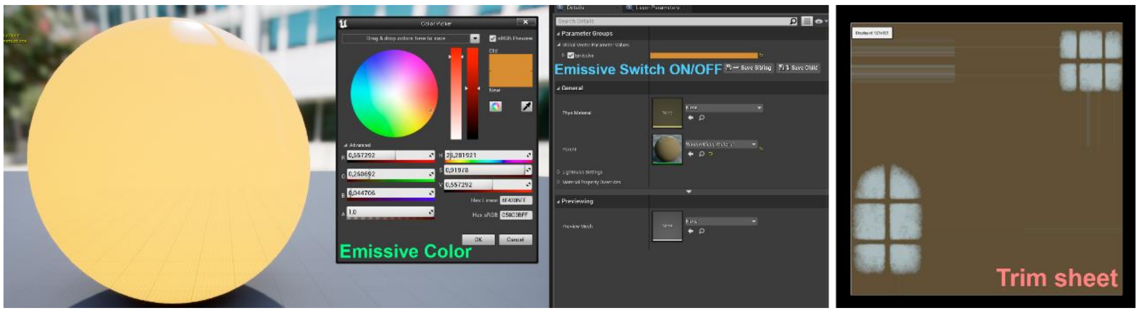
The good point is that they can be placed wherever the user wants to as they aren't attached to anything as some asset packs do.

Figure 5.6.1 *Project Windows*



Windows have trim sheet textures, so both rectangular and rounded windows share one material and one texture. Additionally, there is a material instance of the material of the window with emissive that can be switched on or off, as well as change the emissive color.

Figure 5.6.2 *Window emissive and trim sheet*



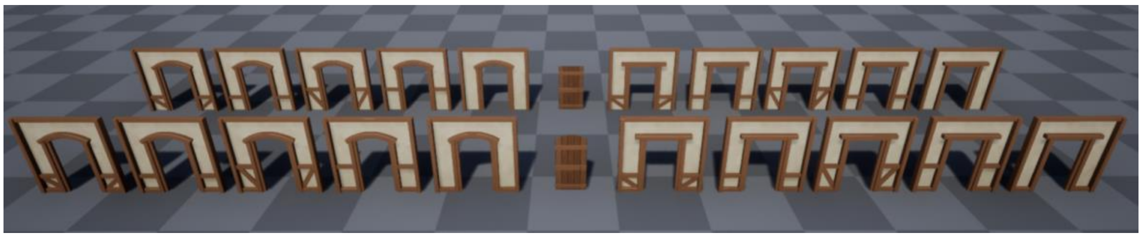
5.7 Doors and door frames

Even though there aren't going to be interior parts on this project asset pack, houses have a door to get into. The environment would look very unrealistic if there weren't at least one door on each house.

First, door frame walls are needed. The simplest wall is going to be used as base. To give more dynamism to the scene, there are two types of doors. One is rectangular and the other is circular at the top, as it happens with the windows. The wall panel material (*5.1.4 Wood wall*) is perfect for the door base. Then it has three wooden plates on each side as well as a doorknob.

Several door frames are built for each type of door and type of wall. In total there are six unique frames, three for each door.

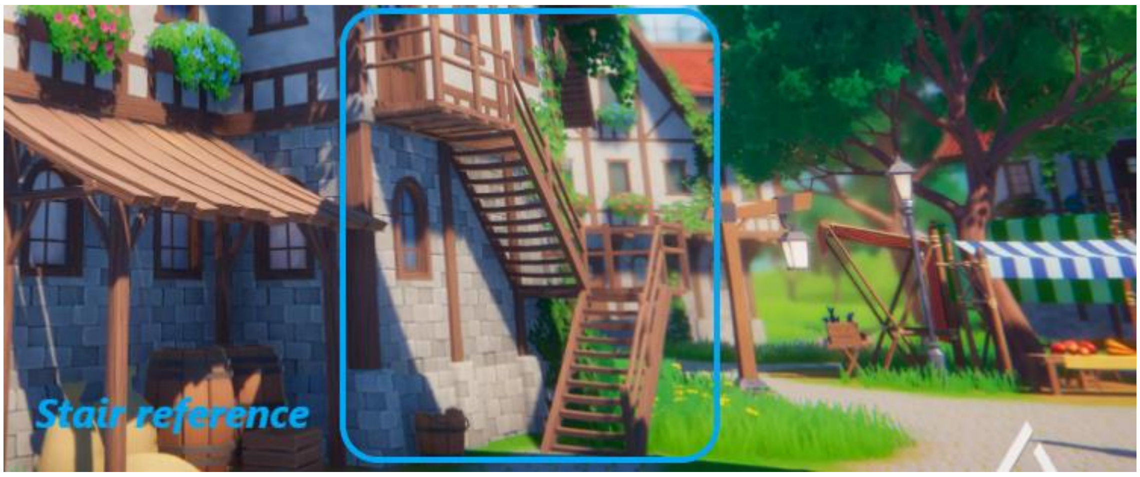
Figure 5.7.1 *Project doors and door frames*



5.8 Stairs and balconies

Initially, this step was only to make wood stairs that lead from the ground to some door. The references to develop this were the ones shown in *Figure 5.8.1.* After just finishing to model the stairs, the author realized that he could also do balconies with the same assets just by closing the stair entrance of the house.

Figure 5.8.1 *Stair reference*

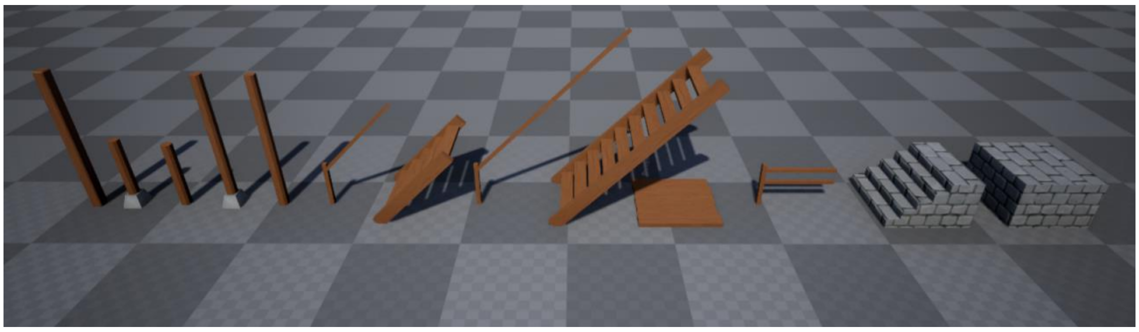


Note: Image from SUNTAIL – Stylized Fantasy Village

After finishing unwrapping the UVs and placing them correctly with the wood material, there was something wrong. Balconies and stairs were suspended in the air as there was no gravity. So, simple wood columns with a stone base fixed this problem.

As shown in *Figure 5.8.2*, there is one stair bigger than the other one. The small one has the height of a house pillar, later explained in *5.9 House pillars.* And the large stair has a height of any wall on a 45 angular degree, so from corner to corner.

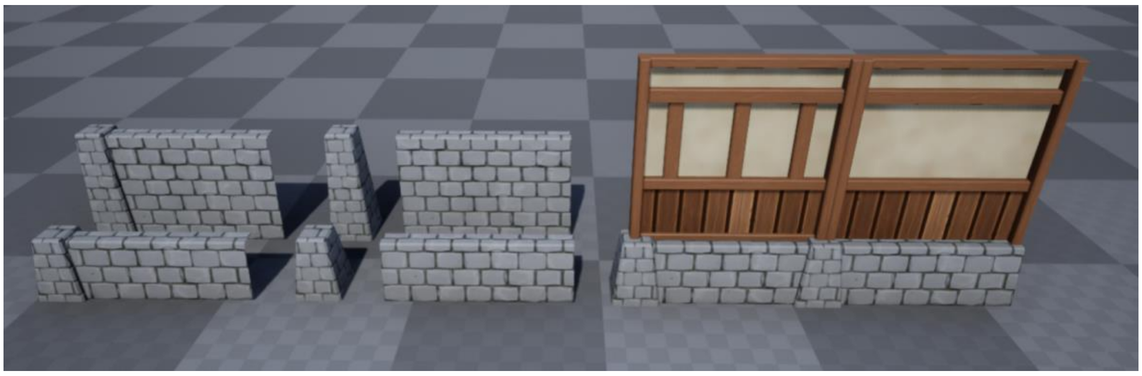
Figure 5.8.2 *Project stairs and balconies props*



5.9 House Pillars

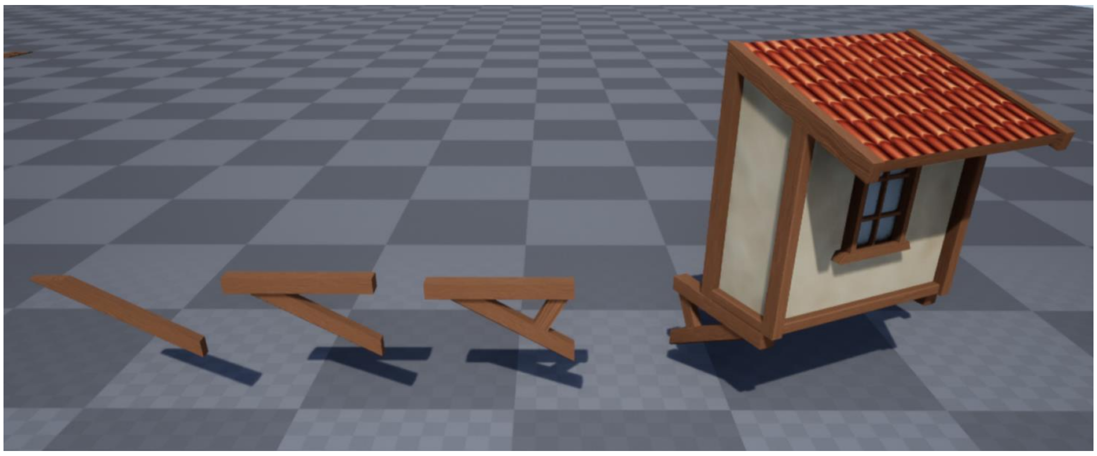
While developing this project, medieval village and house references have been often consulted. Most of these houses have a stone foundation where the house is built on top. Some asset packs do this by making the first bottom line of walls made of stone material. However, the author didn't like this approach and searched for another one. This one consists of creating a quarter of the height of the wall and some stone pillars placed between each wall, that give the illusion of sustaining the whole house.

Figure 5.9.1 *Project House Pillars*



Medieval houses also have chambers that protrude from the structure, as extensions. These extensions were supported by columns or by beams, like the ones shown in *Figure 5.9.2,* to bear the full weight of those structures.

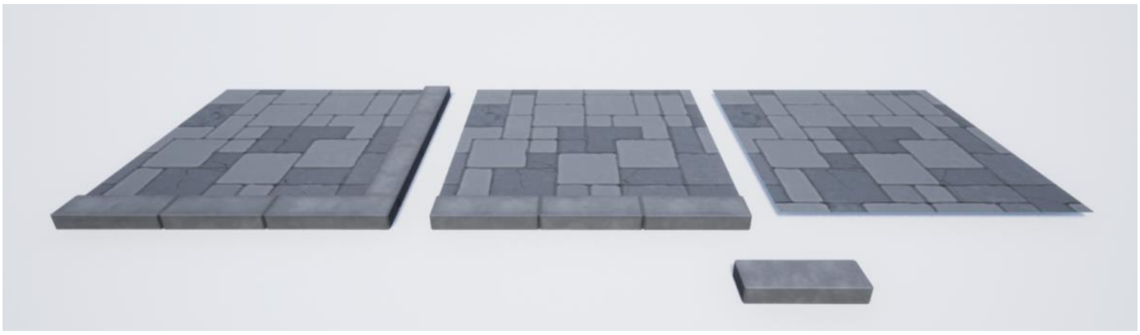
Figure 5.9.2 *Project wood supports*



5.10 Sidewalks

Medieval cities didn't have sidewalks. However, throughout the history they have been implemented in cities to distinguish where citizens can walk and where vehicles can go. They break the monotonous ground texture and give the city a more advanced perspective.

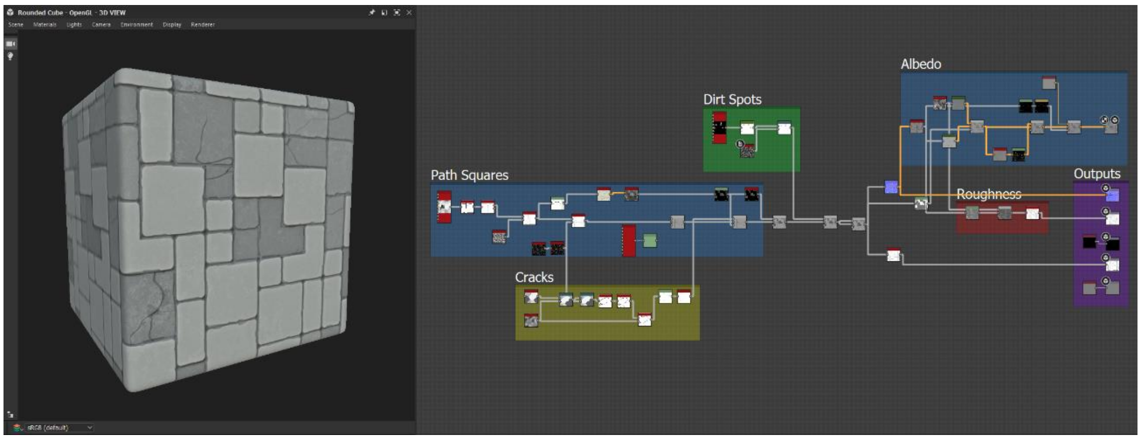
Figure 5.10.1 *Project sidewalks*



5.10.1 Sidewalk material

The creation of this material was simple. Squares are created with several random erosions on the edges with some noise maps. Then, cracks, dirt spots, and different stone tile colors are placed to break the monotonous pattern and give some variety to the material.

Figure 5.10.2 *Sidewalk material*



5.11 Ground material

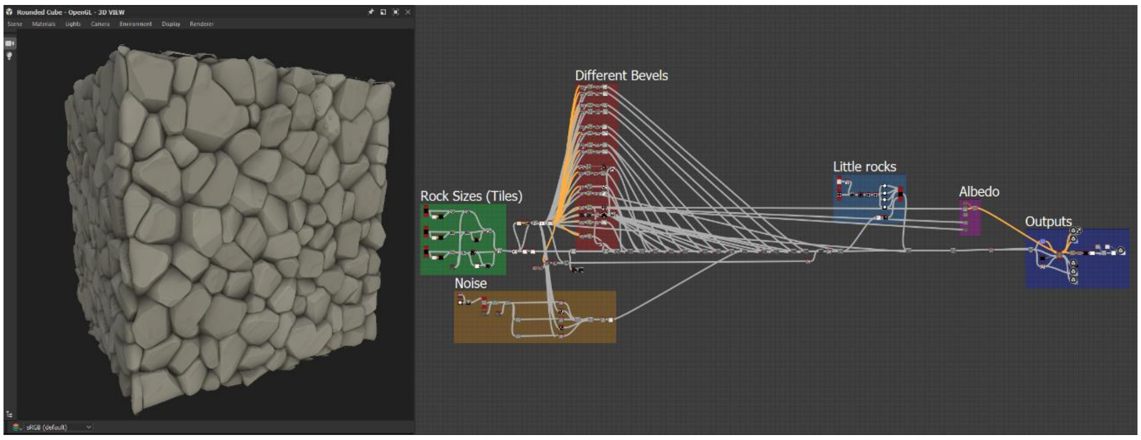
Medieval cities streets were composed of big stones crushed on the dirt ground. This material had some dirt parts among the stones that is blended inside Unreal Engine, so the user can decide whether to have dirt or not.

Like all materials, it has been developed inside Adobe Substance Designer. As shown in *Figure 5.11.1* graph it has been created with:

- **Rock Sizes:** several tile samplers generate three rock sizes (big, medium, small)

- **Different bevels:** as the name says, different bevels affect each rock from different angles.
- **Noise:** some grunge noise is added on top of each stone to break flatness and give irregularity.
- **Little rocks:** some rocks have little rocks that act like dust.

Figure 5.11.1 *Ground material*



5.12 Street decorations

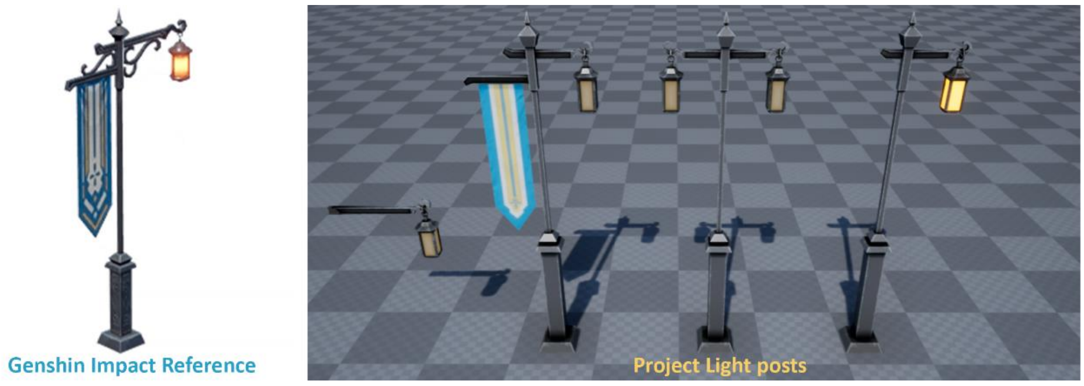
Street decorations help to break the monotony and make the village have a warmer and livelier atmosphere. In other sections, references have been extracted from numerous sites. However, in this section, much of the inspiration has come from the videogame Genshin Impact. Therefore, on several occasions there will be comparative images of the references and the props for this project, always trying to give each asset a personal touch.

5.12.1 Light posts

Light cannot appear out of nowhere. Something must generate it and must be placed somewhere. That's when light posts take place, as they allow to fill a night scene with lights that come out from them.

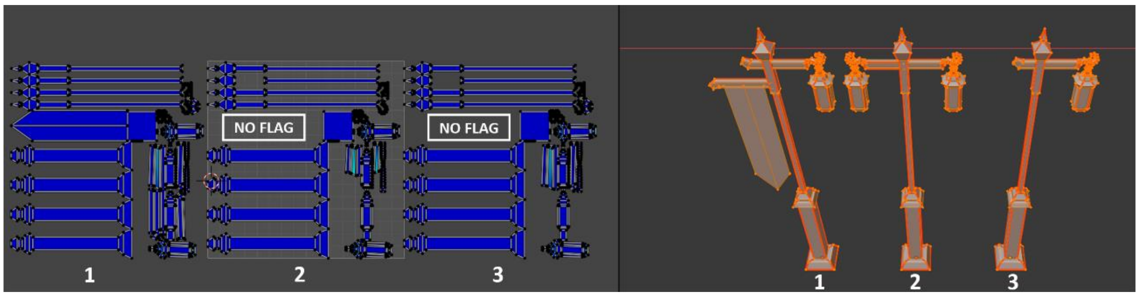
First, light posts are modeled taking into reference Genshin Impact which has only one type of metallic light post, so from that reference, four different light posts are created.

Figure 5.12.1 *Project light posts & Genshin Impact light post reference*



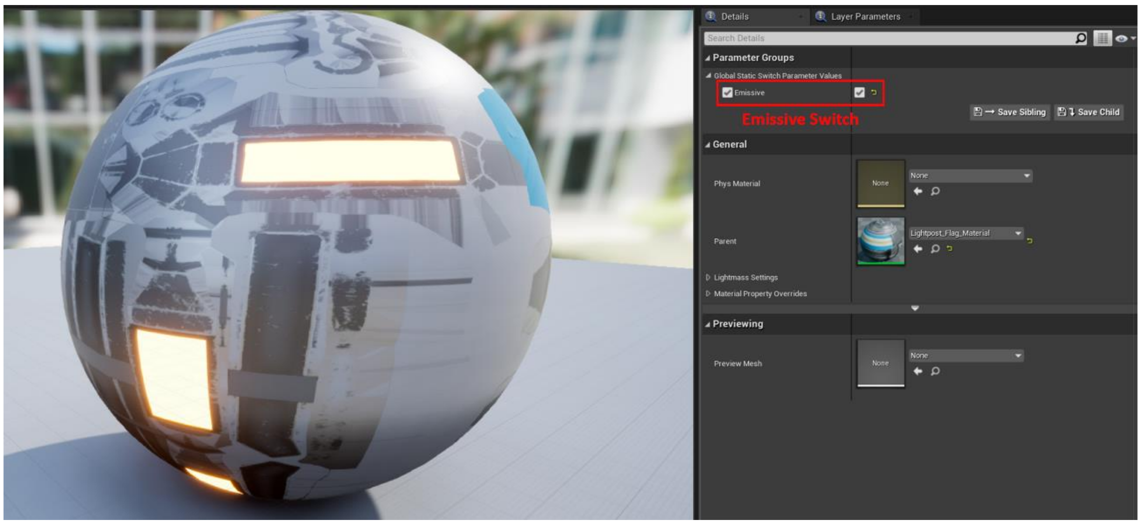
As the first light post created was the one with the flag on it, all other light posts share material and textures. This happens as their UVs are unwrapped and placed in the same position as the textures of the flag light post. In *Figure 5.12.2* are three light posts. As it can be seen, all UV displacement is the same, however, light posts 2 and 3 do not have a flag, so on that texture coordinates, there will be no UV displaced.

Figure 5.12.2 *Light posts shared UVs*



Additionally, inside Unreal Engine, a material instance of Lightpost_Material has been created. This allows the user to enable or disable the emissive through a Switch on the material Blueprint as shown in *Figure 5.12.3.*

Figure 5.12.3 *Emissive Switch light post*



Parameter Groups
Global Static Switch Parameter Values
[✓] Emissive [✓]

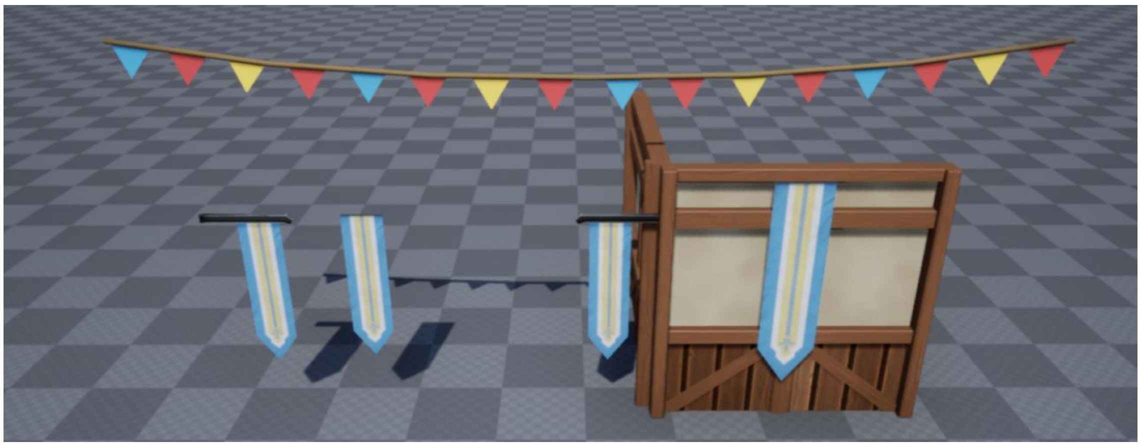
General
Phys Material: None
Parent: Lightpost_Flag_Material
Lightmass Settings
Material Property Overrides

Previewing
Preview Mesh: None

5.12.2 Flags

As we developed a light post with a flag on it, reusing the material previously created we did two more flag assets. One with just the flag and the other with a metal piece so it can be placed as if it was drilled into the wall. Moreover, one festivity flag is done.

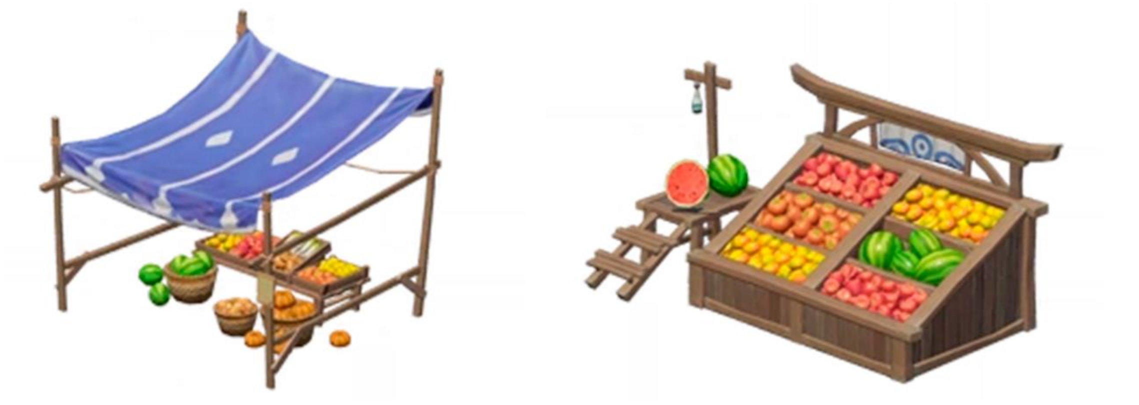
Figure 5.12.4 *Project Flags*



5.12.3 Street shops (table & benches)

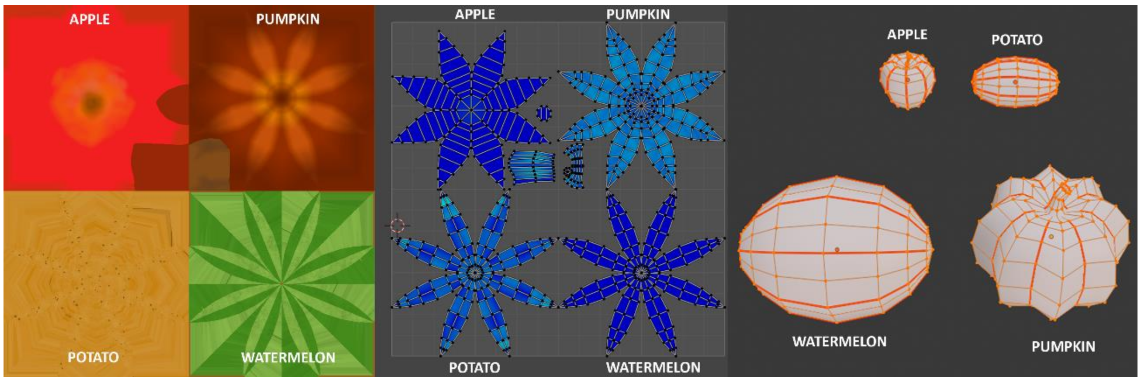
One thing that most asset packs have in common is street shops. Blacksmith, book shops, pubs, fish shops, butcher shop, among others. But among them all, the street shop that brings more color to the scene is fruit shops. So, as well as with other props, the two fruit shops from Genshin Impact (*Figure 5.12.5*) have been taken as reference, always giving some modifications.

Figure 5.12.5 *Genshin Impact reference fruit shops*



To develop these stores, first the basic structures, crates, boxes, and four different types of vegetables (watermelon, potato, apples, and pumpkin) were modeled and unwrapped. Wood parts were textured by already developed materials, however, vegetables and the awning needed to be painted, so it's done in Substance Painter. As fruits are small props that don't require high details, a trim sheet was built in a 1024x1024 pixels texture.

Figure 5.12.6 *Fruits trim sheet texture*



Now, all is textured except the awning. To give more variety to the scene, we set that the awning color texture can be modified on Unreal Engine. So, users can create as many material instances as colors they want. Furthermore, this awning material could be used for house interior walls, even though it wasn't the goal.

In addition to all that, so the vegetables weren't placed on the ground for the awning street shop, two tables, two benches and different crates, and boxes have been developed. Besides, there is a vegetable fruit shop prefab and two crates with apples and potatoes that have been created using Blender gravity simulator, so the placement of those is realistic.

Figure 5.12.7 *Project vegetables assets*

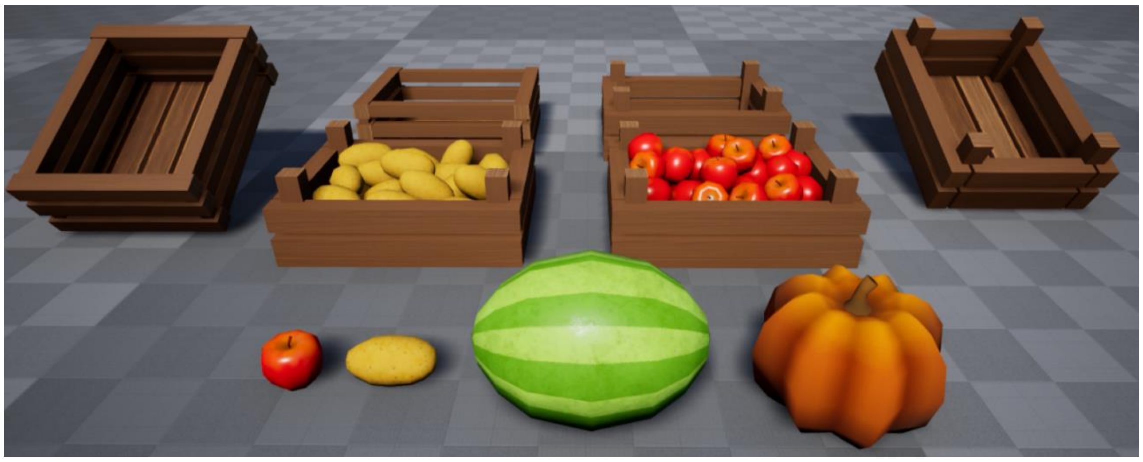
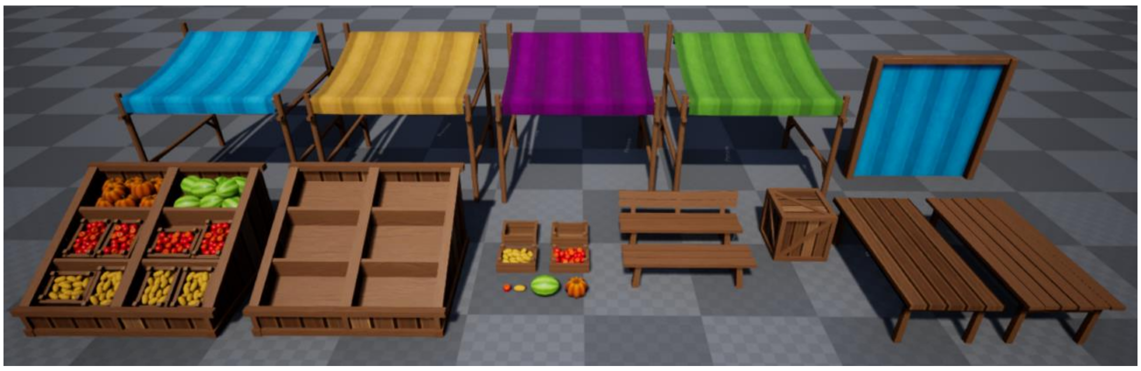


Figure 5.12.8 *Project Street shops*



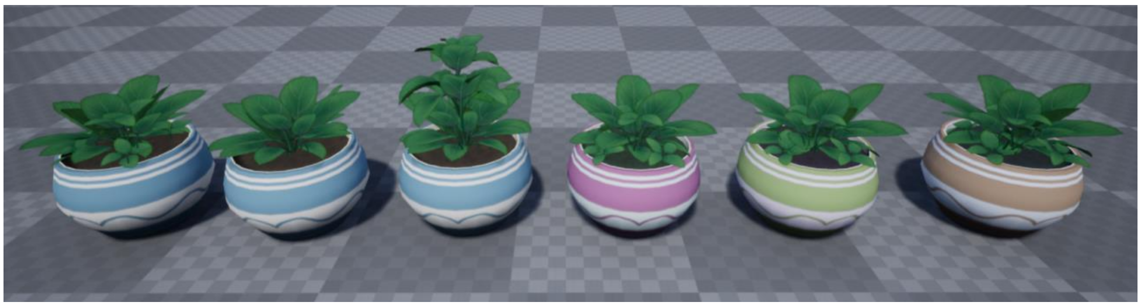
5.12.4 Plants

Plants were the last thing developed for the project. It wasn't planned to be any plants, but when finishing to develop the Demo Scene, something was missing. Plants are the only organic assets in the whole scene and even though there are just a few, they give a realistic and livelier feel than if they weren't.

First, a flowerpot and three different leaves were modeled in Blender, so later they have painted in Substance Painter with two different materials. One for the flowerpot and other for the leaves as they have an opacity map.

This allowed to build three different plants. Then on Unreal, three material instances have been created for the flowerpot. So, plants can have four different pot colors (blue, green, pink, and yellow) as shown in *Figure 5.12.9.*

Figure 5.12.9 *Project plants*



5.13 Import props to Unreal Engine 4

This step has been constantly happening during the curse of the project. It was main to test meshes, materials, colliders, modular construction, color correction, and how everything looked all together. Several important points have been appearing which needed to be solved before the release of the asset pack.

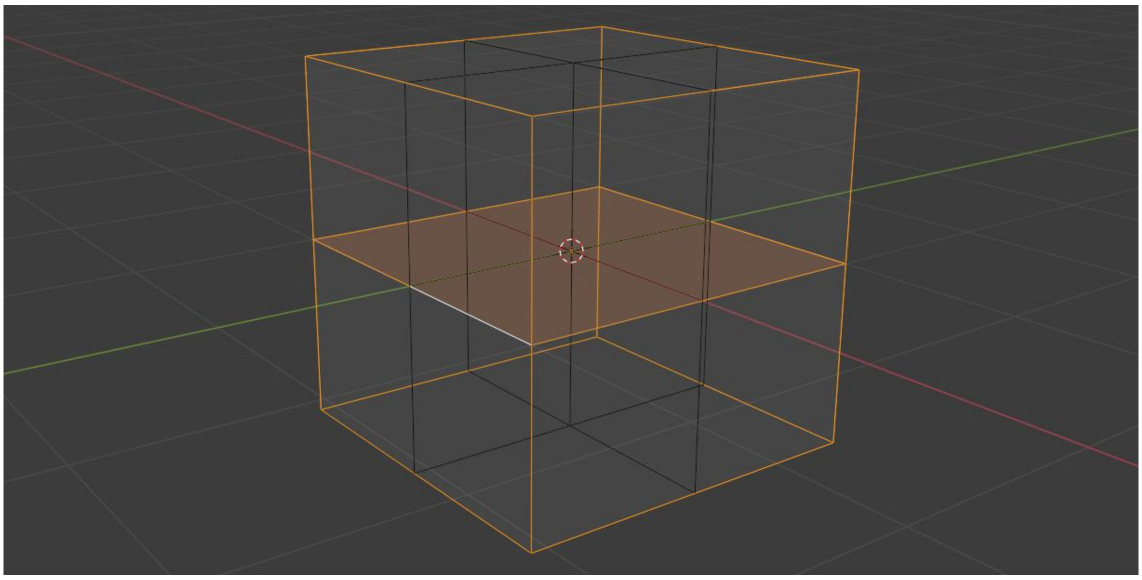
The project has been developed in Unreal Engine 4 - ***version 4.27***.

5.13.1 Pivot points

When starting this project, pivots points weren't even a thing to consider, as it was the author first time creating a large number of assets. However, this wasn't a personal project, so the assets needed to be intuitive and easy to work with.

The pivot point of the assets was set to be on the origin of the geometry. This mean that Blender would move the origin (pivot point) of the object to its center. This can be easily seen in *Figure 5.13.1.* where the origin (red and white stripped circle) is in the center of this cubical object.

Figure 5.13.1 *Origin point example*



This may not seem a real problem, as the pivot point allow the asset to be moved, rotated, and scaled from that center point. Nevertheless, as in rotation and scale it is better to have the pivot point in the center, it doesn't happen the same with the movement. To build a scene with a well-planned asset package as this, the main thing the user would do is move and rotate the objects. However, only in special occasions he would have the scale them, and if he does, he would scale the majority of the meshes. This is a pivot position has two problems:

- **Problem 1.** Asset position will be incorrect when importing it into the scene. This happens, as the pivot point snaps in the Z axis to the surface where the mesh is placed, in this case the ground.
- **Problem 2.** Snapping between meshes (mainly in roofs and walls) is also incorrect. The pivot point of the asset only snaps to another asset vertex. As the pivot is in the center and not on a side of the mesh, the snapped mesh will be displaced and collapsing with the other mesh *(see Figure 5.13.3 – Problem 2).*

To solve these problems, the pivot point needs to be:

- **Solution 1.** Pivot point need to be on the lowest part of the object, so when importing the asset to the scene, the whole mesh is visible without trespassing the ground (as happens in *Figure 5.13.2 – Problem 1*).
- **Solution 2.** On meshes that need snapping, add the pivot point to a so it can be snapped to the other mesh vertex on the same axis level.

Figure 5.13.2 *Pivot point problem 1*

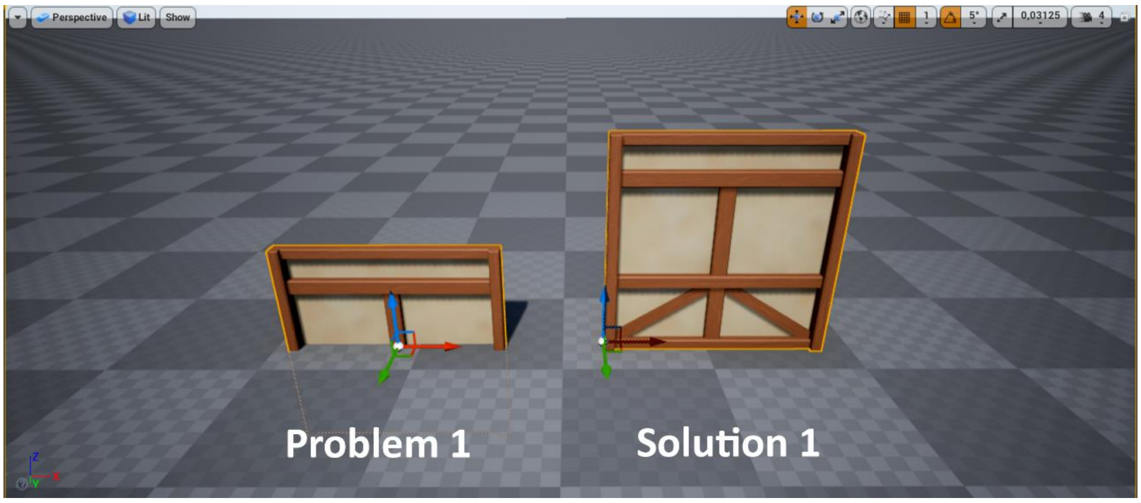
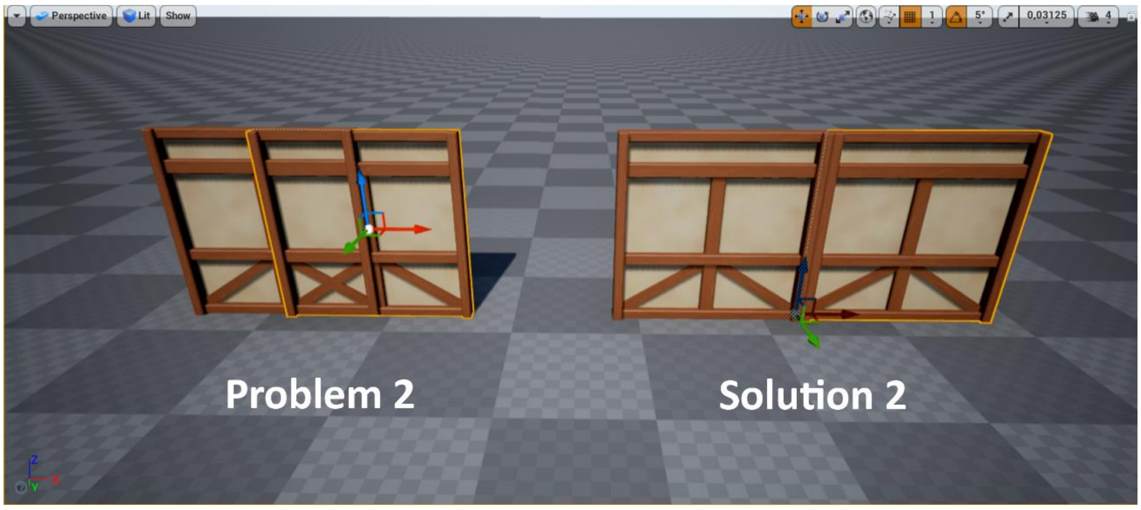


Figure 5.13.3 *Pivot point problem 2*



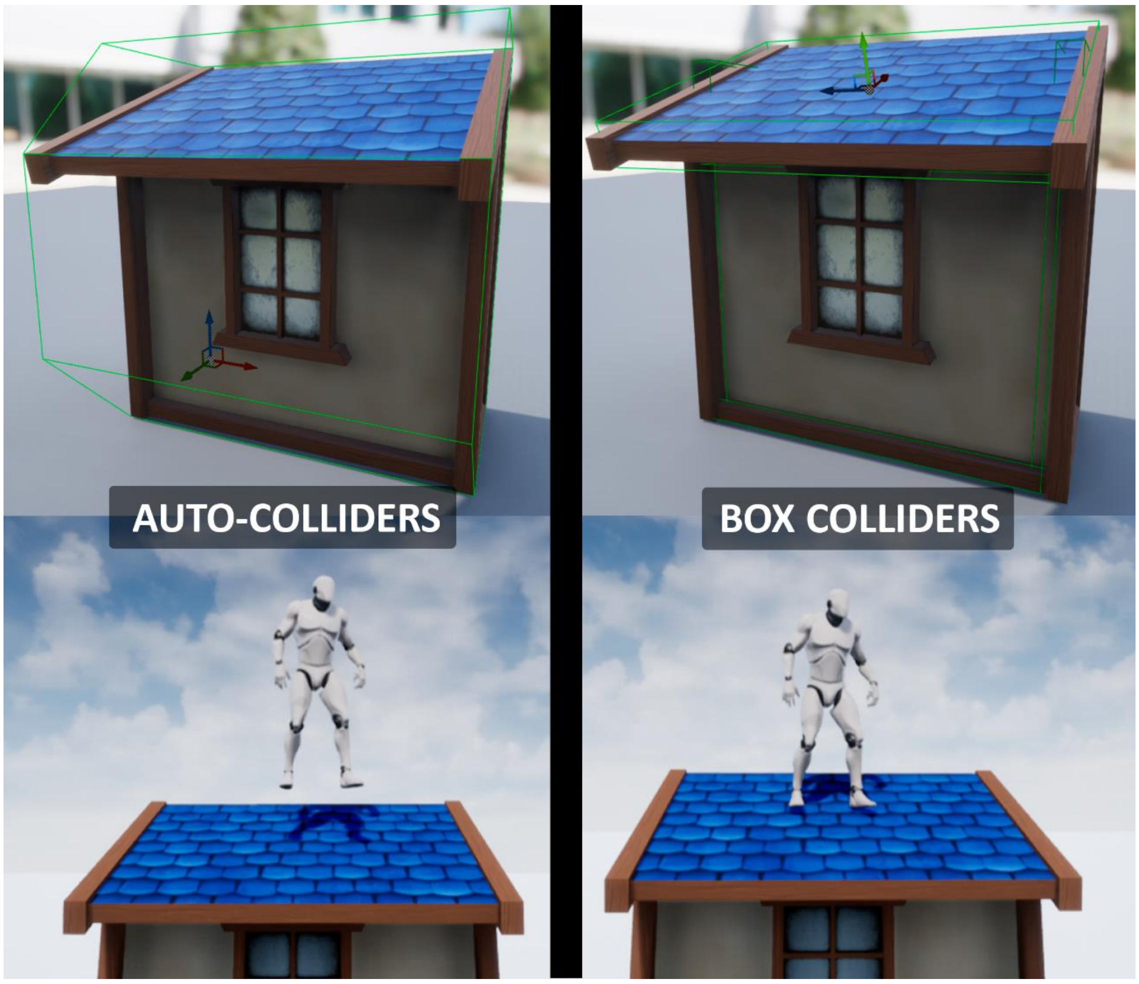
5.13.2 Colliders

It wasn't until testing the scene with the Unreal Third Person Character that some colliders were found to be wrong.

Unreal Engine has the option to generate mesh colliders when importing meshes. This function works pretty well for simple meshes as walls, columns, or pillars. Nevertheless, when meshes have some wide angles, this collider doesn't work as expected. That's why some asset package collision meshes have been rebuild using simple box colliders.

As shown in *Figure 5.13.4*, auto-colliders generated by Unreal Engine make the character float at the top of the roof, think that doesn't happen when we generate better colliders with a few box colliders.

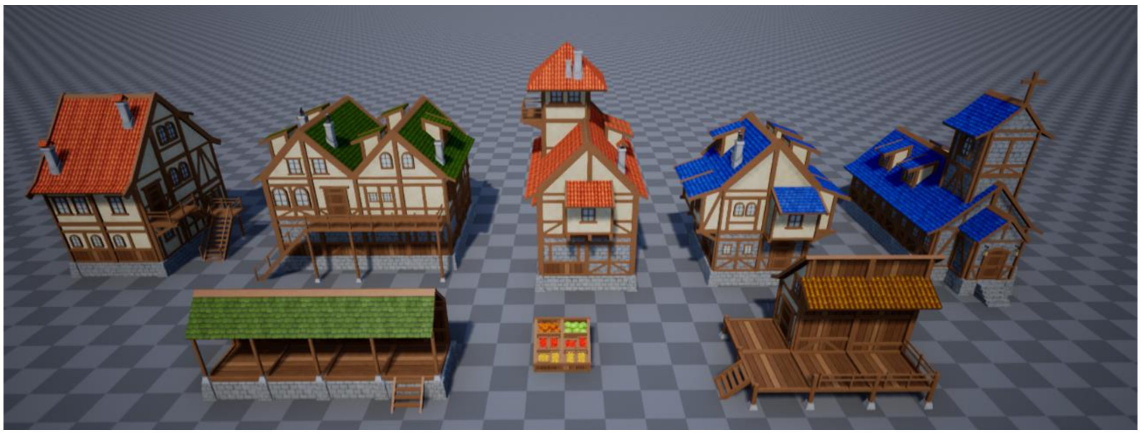
Figure 5.13.4 *Auto-colliders vs Box Colliders*



5.13.3 Prefabs

As mentioned before, the asset package must be designed to be as comfortable as possible for the users. So, creating several prefabs will allow user to drag and drop pre-built houses and constructions. These prefabs only have the necessary materials and collisions. The only bad thing is that they can't be modified, however, users can create their owns.

Figure 5.13.5 *Project Prefabs*

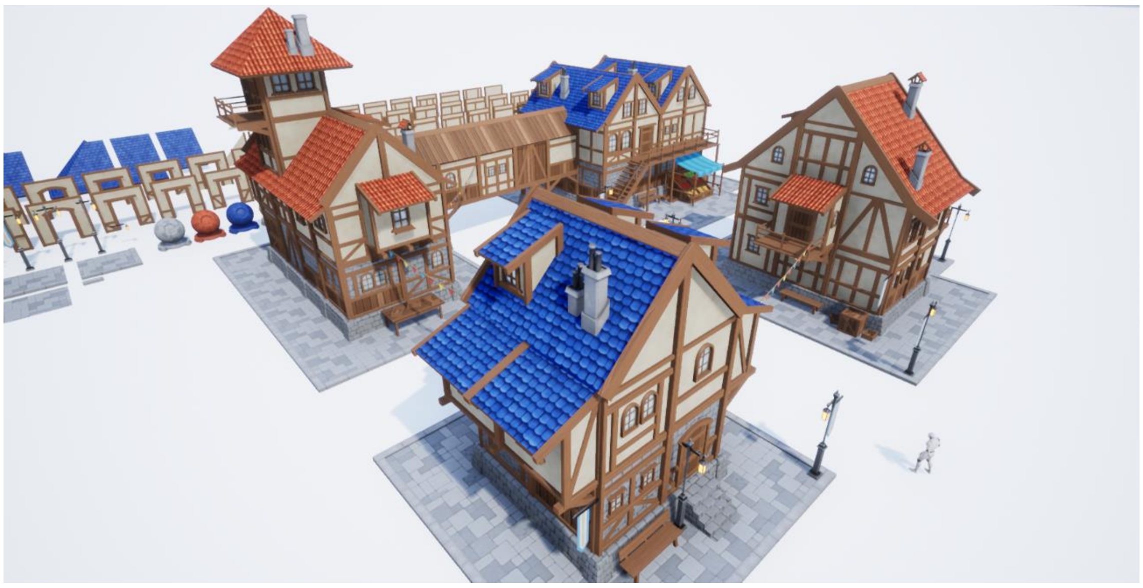


5.14 Creation of a demo scene

After having all the necessary meshes, materials, and having the PropScene finished, it was time to create a demonstration scene so users can see the full potential of the asset package.

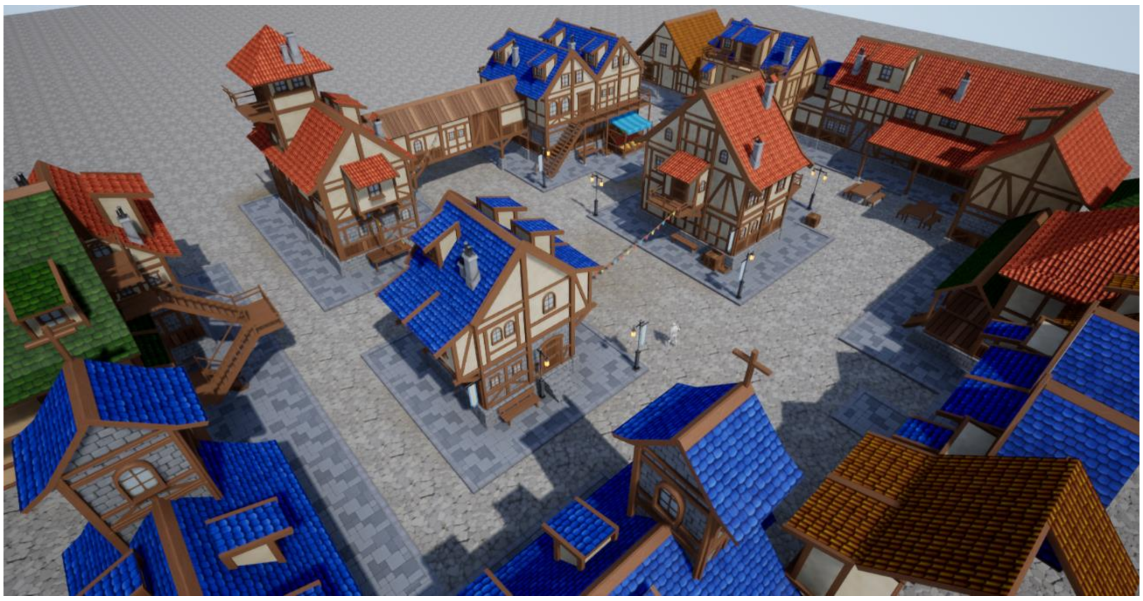
First, some houses were developed by looking at different references. These houses are the ones in the center of the DemoScene. As all four houses have two floors, a bridge between two of them was assembled so the scene is uneven.

Figure 5.14.1 *DemoScene – First iteration*



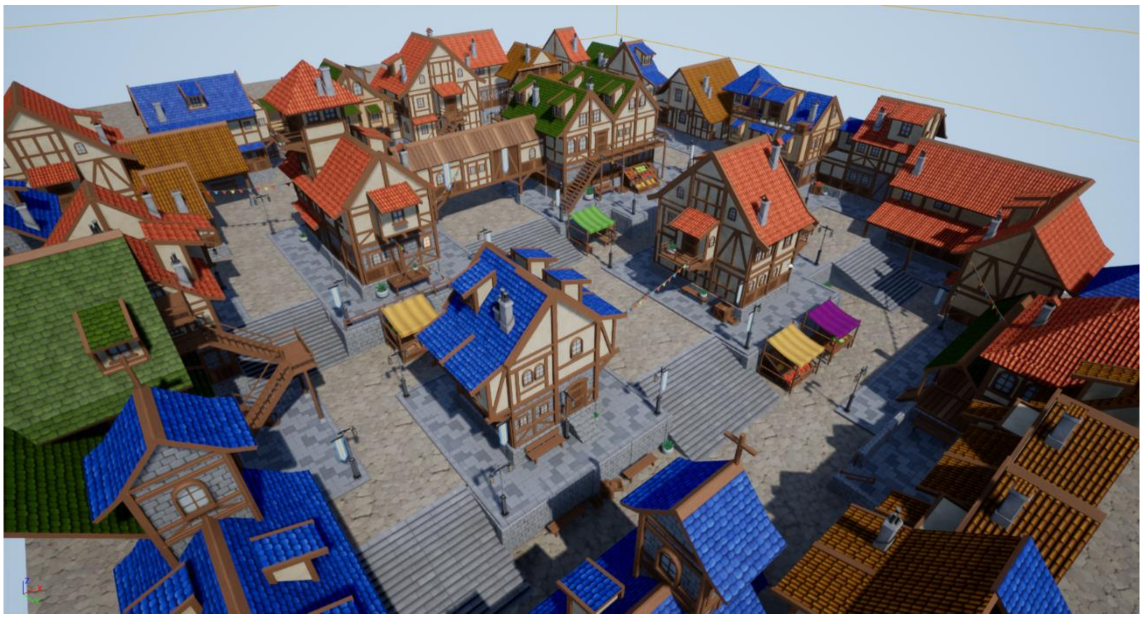
Once these houses were built, it was time to start to construct the peripheric area, so the scene is closed, and the player has a stablished zone to play on. Additionally, the ground material was placed, along with roof and clay material instances to give the scene some variety of colors.

Figure 5.14.2 *DemoScene – Second iteration*



The next step was finishing to close the village, as well as place all the street decoration assets. The scene was too flat and monotonous, so creating several large stairs and wall pillars allowed to give the scene some verticality and contrast.

Figure 5.14.3 *DemoScene – Third iteration*

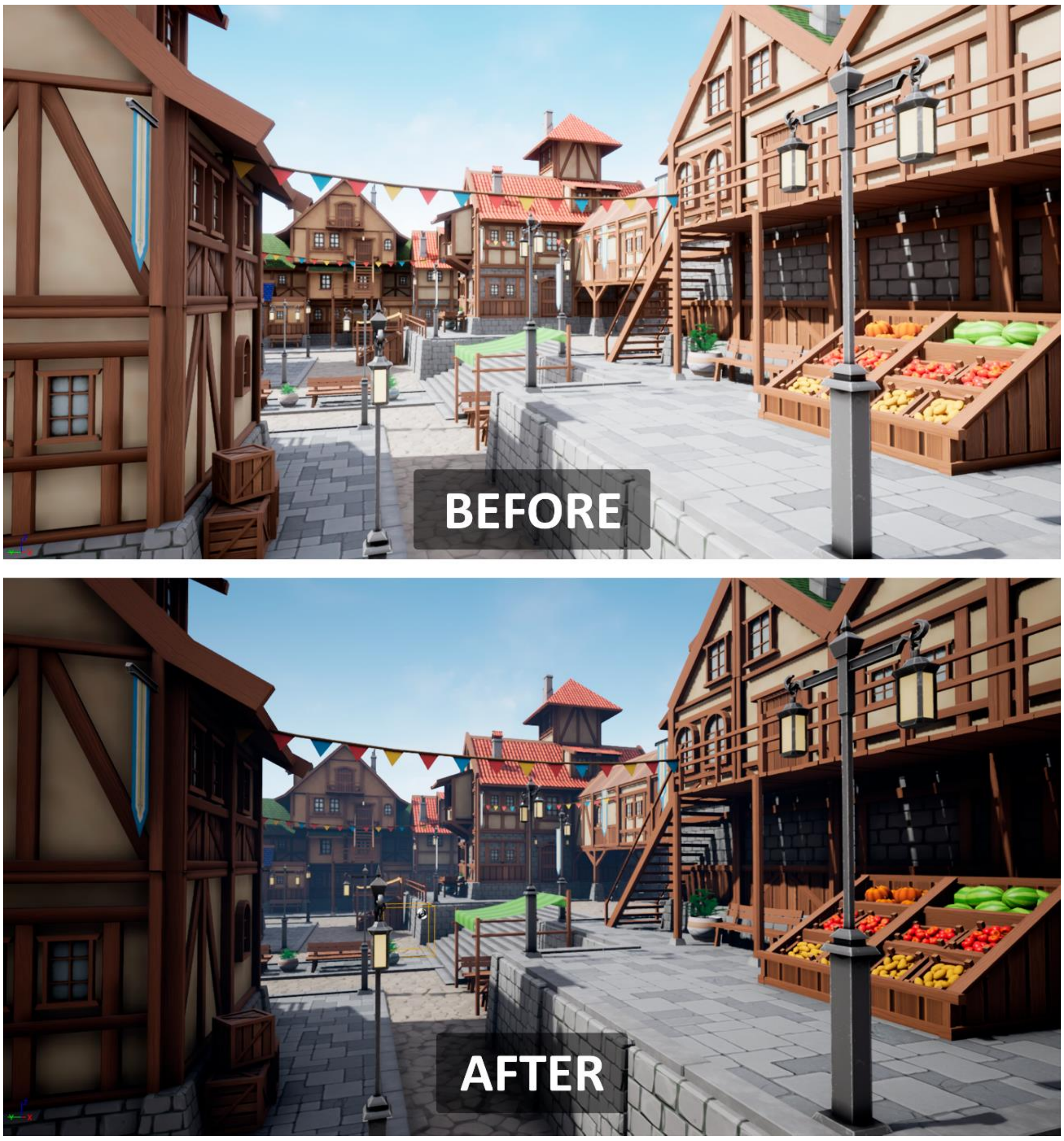


After placing several street decorations as shops, tables, boxes, crates, benches, flags, plants, and light posts, along the scene, it was time to play with several light values to give the scene the best possible look. This was a very tedious task since the author had no experience in this area. After several YouTube video tutorials and lots of different light settings, the scene has increased aesthetically. It has been a noticeable difference between before and after, as can be seen in *Figure 5.14.4.* This was mainly because of:

- **SphereReflectionCapture:** It captures how the environment looks from that point of view. That capture is then used as reflections on the surfaces that are inside the influence radius of that reflection capture actor.
- **ExponentialHeightFog:** Creates more density in low places of a map and less density in high places. The transition is smooth, so there's never get a hard cutoff as altitude increases. This can be perfectly noticeable on the 'before-after' comparation.
- **PostProcessVolume:** This allowed to tweak the overall look and feel of the scene. Ambien occlusion, image effects as vignette intensity, color grading temperature, some saturation, as well as minimum and maximum exposure.

After tweaking light settings, it can be seen that the Sun doesn't burn so much the image, there is a density fog that creates a good and realistic ambient, shadows are darker and ambient occlusion is better balanced.

Figure 5.14.4 *DemoScene – Final iteration (before and after)*



5.15 Prepare project for Unreal Engine Marketplace

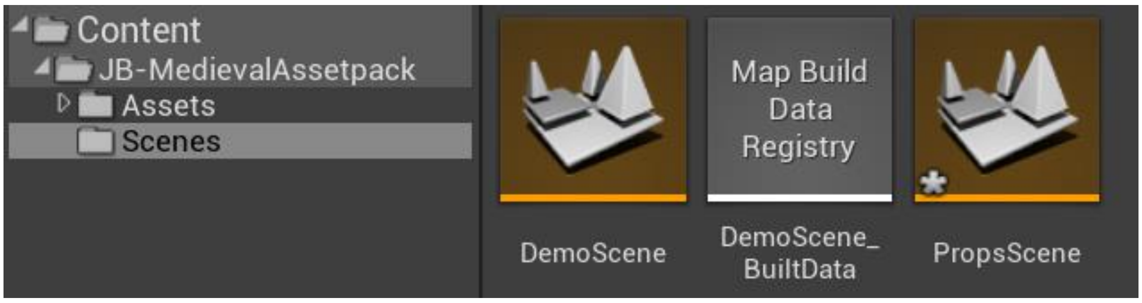
5.15.1 Folder organization and naming

This step is very important as it needs to be as simple and intuitive as possible, so users don't get lost trying to find anything.

Even though this process has been carried out throughout the entire project, it has not been until the asset pack has been prepared to be exported to the Unreal Engine Marketplace, that everything has been perfectly organized.

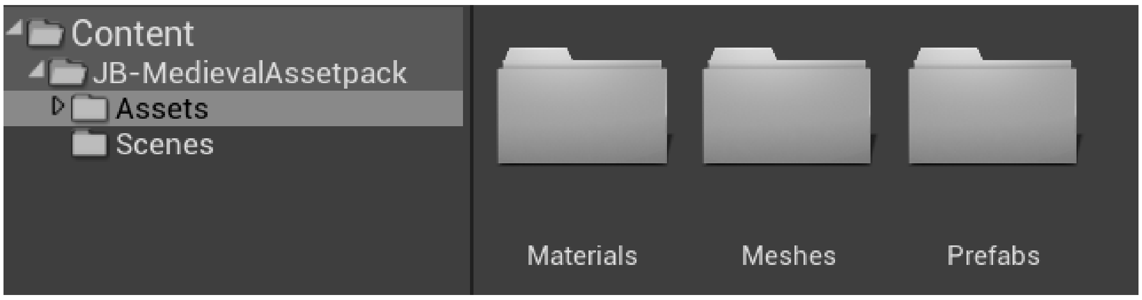
When opening the asset pack and clicking the main folder, the **assets** and **scenes** folder pop up. Inside **scenes** there is a DemoScene, showing a demonstration environment ready to be played in, and a PropsScene with each unique mesh, materials, and prefabs.

Figure 5.15.1 *Scene folder*



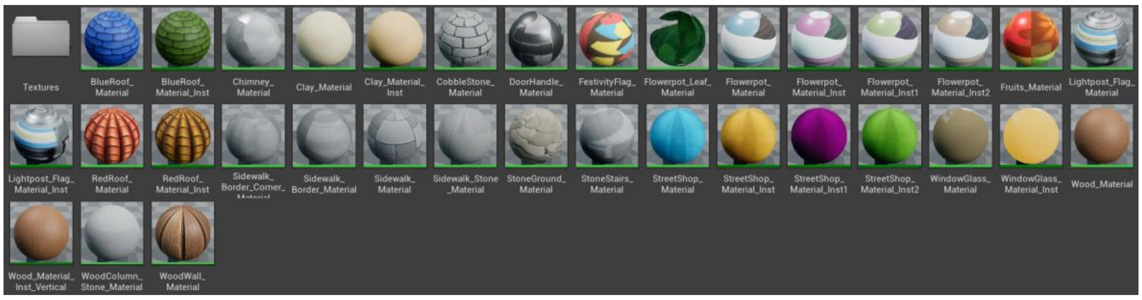
On the other hand, there is the **assets** folder. This one is more holds all **materials, meshes,** and **prefabs.** Going from left to right folder, the first folder is the materials one.

Figure 5.15.2 *Assets folder*



Inside materials folder there are a total of **22 materials**, **12 material instances** and a textures folder. Materials naming is *Name_Material* and for material instances is *Name_Material_Inst.*

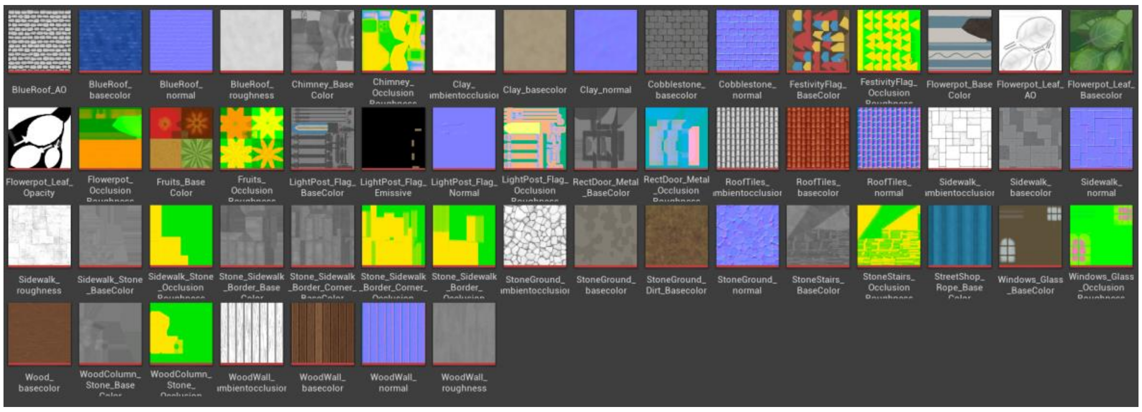
Figure 5.15.3 *Materials folder*



Diving into textures folder there are **55 textures** of 512x512, 1024x1024 and 2048x2048 pixels. Naming textures are as follows:

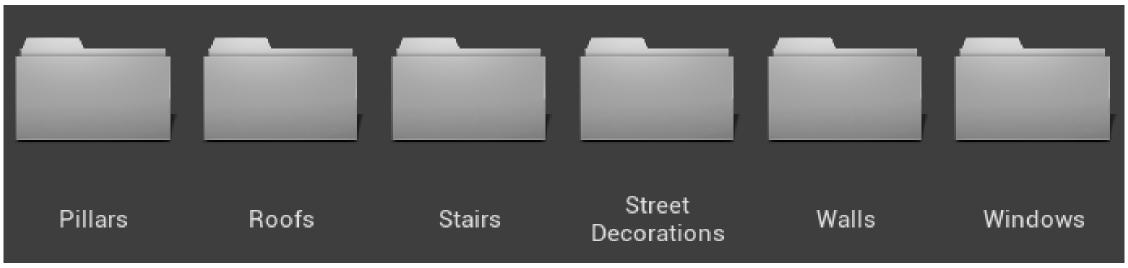
- *Name_AO* or *Name_AmbientOcclussion* for ambient occlusion map.
- *Name_BaseColor* for albedo map.
- *Name_Normal*.
- *Name_Roughness*.
- *Name_Emissive* for emissive or self-illumination map.
- *Name_OcclusionRoughnessMetallic* that combines each RBG channel to one map. Red channel for occlusion, green for roughness and blue for metallic.

Figure 5.15.4 *Textures folder*



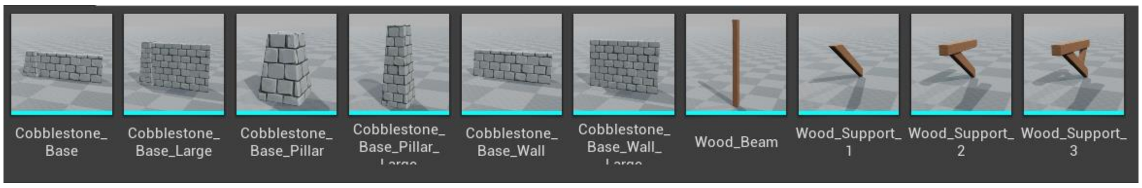
Going back to assets folder, now it's the turn of **meshes**. Those can be divided into 6 different subfolders that will be explained next.

Figure 5.15.5 *Meshes folder*



First folder on meshes is **pillars**. It holds stone base wall pillars, and several wood beams supports. There are "normal" base walls or *_large* ones.

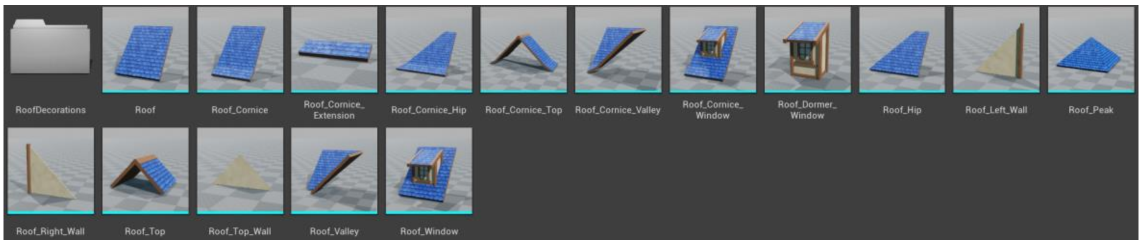
Figure 5.15.6 *Pillars folder*



The next meshes folder is **roofs.** It contains **roof decorations** as well as all the roof pieces. Roof naming can vary between if they have a cornice or not, so:

- Roof with cornice. *Roof_Cornice_RoofPart.*
- Roof without cornice: *Roof_RoofPart.*

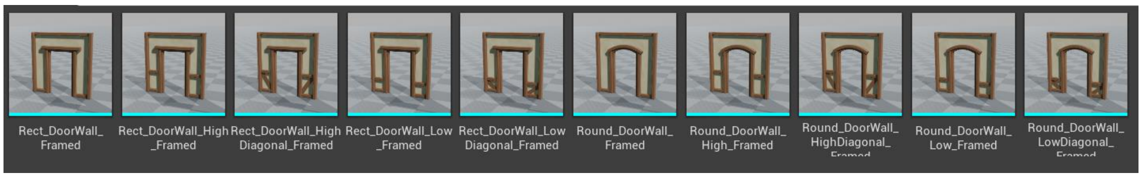
Figure 5.15.7 *Roofs folder*



Following the next folders there are **stairs** and **street decorations**. Both of them have simple naming, so there is no need to explain them here.

However, the next folder is the most complex one. Inside **walls** there are 4 subfolders: door walls, extensions, framed walls, and unframed walls (*explained on 5.4 Walls*). On door walls there is a subfolder for **framed** and for **unframed** walls, as well as a name for **rectangular** or **rounded** door type.

Figure 5.15.8 *DoorWalls folder*


Rect_DoorWall_Framed
Rect_DoorWall_High_Framed
Rect_DoorWall_HighDiagonal_Framed
Rect_DoorWall_Low_Framed
Rect_DoorWall_LowDiagonal_Framed
Round_DoorWall_Framed
Round_DoorWall_High_Framed
Round_DoorWall_HighDiagonal_Framed
Round_DoorWall_Low_Framed
Round_DoorWall_LowDiagonal_Framed

Walls folder naming is pretty simple once it is explained. First, there are two types of walls, which also have two types of walls based on materials.

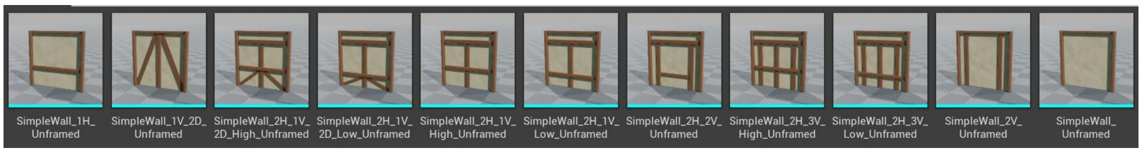
- **Framed**. Wood beams frame walls on the four directions.
  - **Composite or simple**. Depending on if the wall has one material or two.
- **Unframed**. Wood beams do not frame the wall at the bottom.
  - **Composite or simple**. Depending on if the wall has one material or two.

And inside composite or simple there is also differences on naming:

- **Simple.** Only interior wood beams are counter for naming. And the other of importance on counting beams is:
  - **Horizontal.** Written as HorizontalBeamsNumber + H. → *1H, 2H, 3H…*
  - **Vertical.** Written as VerticalBeamsNumber + V → *1V, 2V, 3V…*
  - **Diagonal.** Written as DiagonalBeamsNumber + D → *1D, 2D, 3D…*

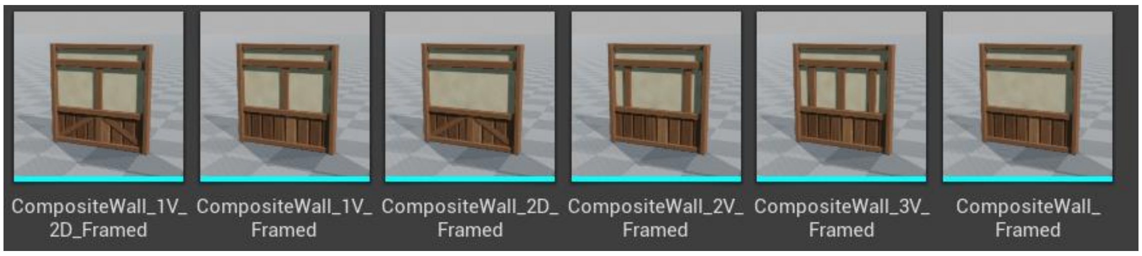
Taking as a reference the third wall of the *Figure 5.15.9*. we needed to count first horizontal beams, then vertical and finally diagonal. So, the wall naming would be: *SimpleWall_2H_1V_2D_High_Unframed* (simple as it is a simple wall, 2 horizontal beams, 1 vertical beam, 2 diagonal beam, high middle beam and unframed).

Figure 5.15.9 *Simple unframed walls folder.*


SimpleWall_1H_Unframed
SimpleWall_1V_2D_Unframed
SimpleWall_2H_1V_2D_High_Unframed
SimpleWall_2H_1V_2D_Low_Unframed
SimpleWall_2H_1V_High_Unframed
SimpleWall_2H_1V_Low_Unframed
SimpleWall_2H_2V_Unframed
SimpleWall_2H_3V_High_Unframed
SimpleWall_2H_3V_Low_Unframed
SimpleWall_2V_Unframed
SimpleWall_Unframed

- **Composite.** All composite walls have the same 2 horizontal beams (one on the middle of the wall and the other on the top), so counting goes like simple walls but without the horizontal factor (only vertical and diagonal). For example, the fourth wall on *Figure 5.15.10* would be: *CompositeWall_2V_Framed*. (Composite as it has two materials, 2 vertical lines (horizontal doesn't count) and framed as the wall is fully framed).

Figure 5.15.10 *Composite framed walls folder*



As last folder there are the three types of **windows**: rounded and rectangular.

5.15.2 Technical information

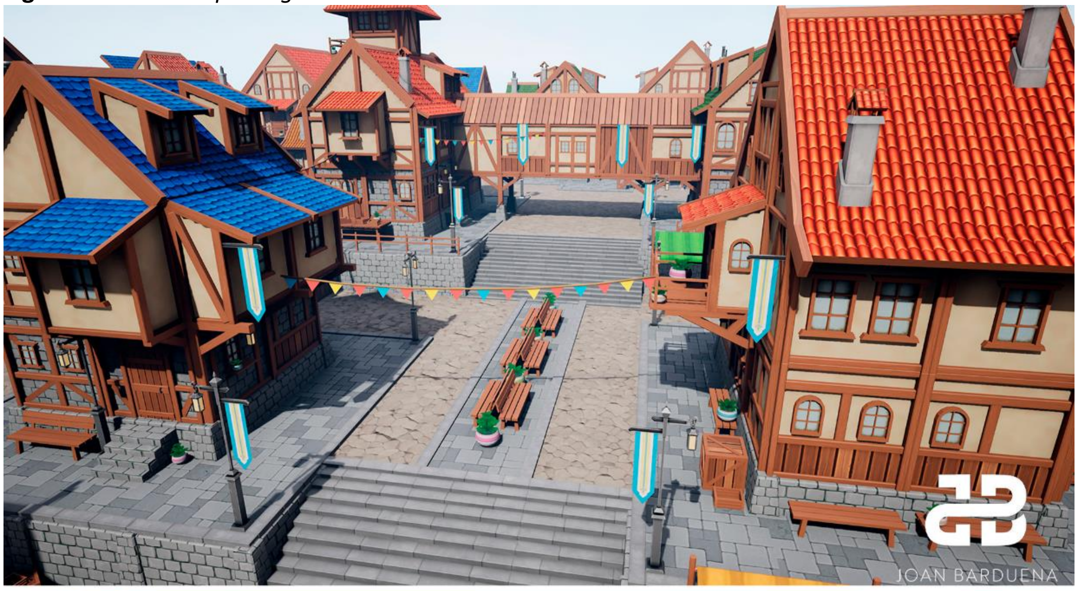
Now that the asset package is ready to be uploaded to the marketplace, it can be count how many assets it has ended up having:

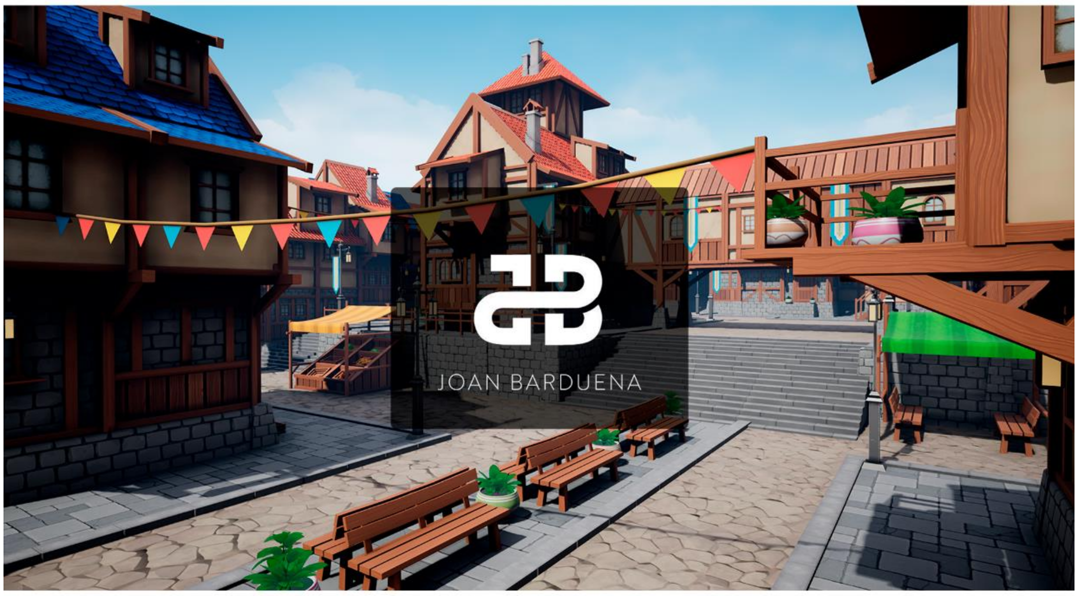
- **Number of Unique Meshes:** 144
  - 34 Walls (half framed, half unframed).
  - Number of Unique Meshes: 144
  - 34 Walls (half framed, half unframed).
  - 10 House Pillars.
  - 16 Roofs + 10 Roof Decorations.
  - 14 Stairs/Balcony pieces.
  - 30 Street Decorations (light post, street shops, plants, flags).
  - 2 Doors + 20 Door Walls.
  - 5 Wall chamber extensions.
  - 3 Windows.
- **Collision:** Box collisions.
- **Number of Materials and Material Instances:** 22 Materials + 12 Instances
- **Number of Textures:** 55
- **Texture Resolutions:** 512, 1024, 2048.
- **Supported versions:** 4.27 +

5.16 Upload project to Unreal Engine Marketplace

Asset package is ready to be uploaded to Unreal Engine Marketplace. But before this, asset package needs to be presented the best way possible. Several screenshots and a video of the Demo_Scene will help to show the full potential of the project. Unreal Engine has a High-Resolution Screenshot mode, that will render the game frame selected in the viewport multiple times and concentrating all tiles into a single image file.

Figure 5.16.1 *Asset package screenshots – 1*





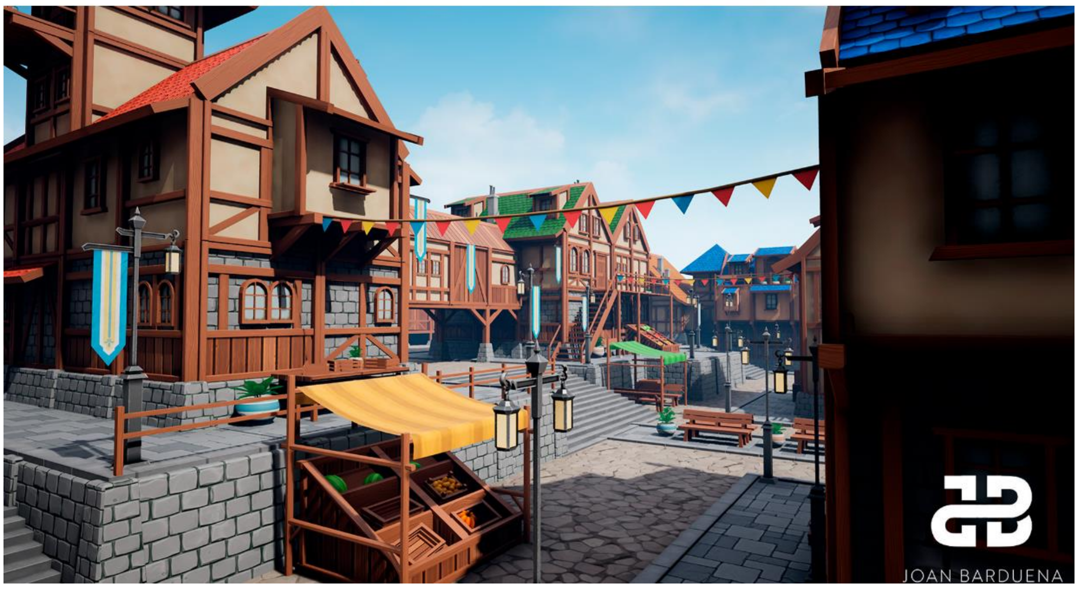
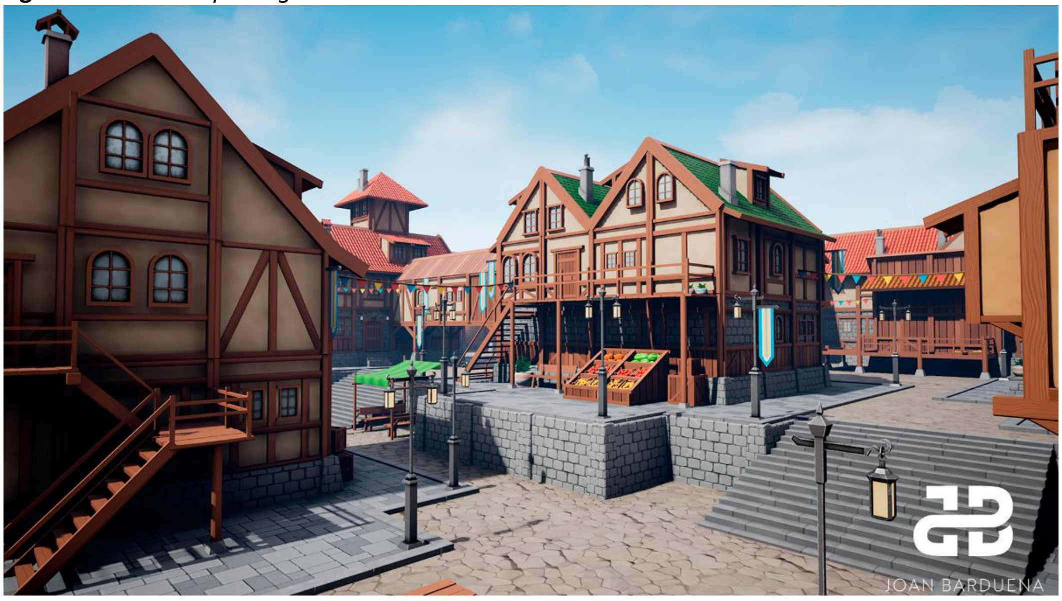
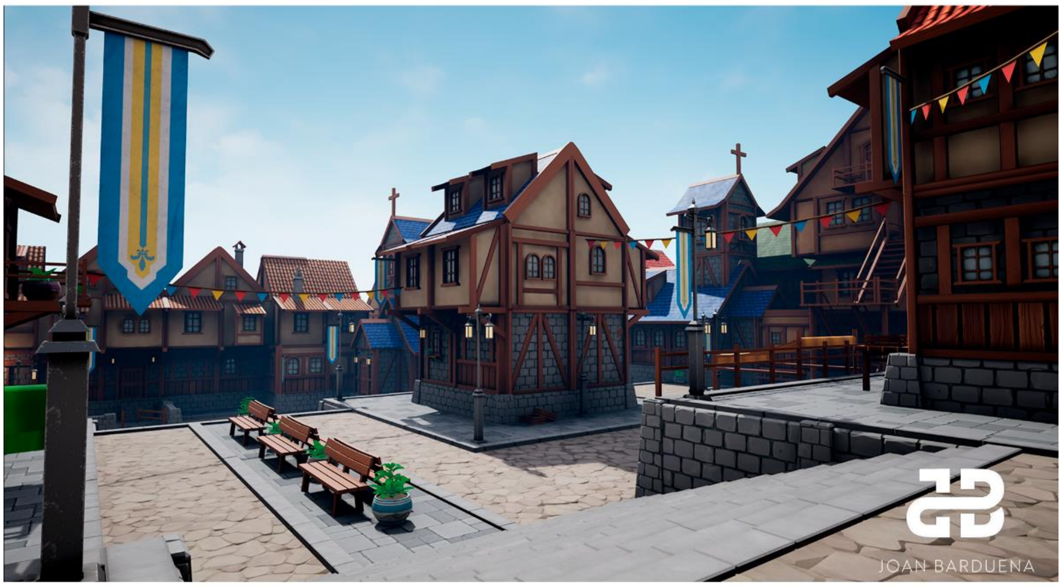


Figure 5.16.2 *Asset package screenshots – 2*





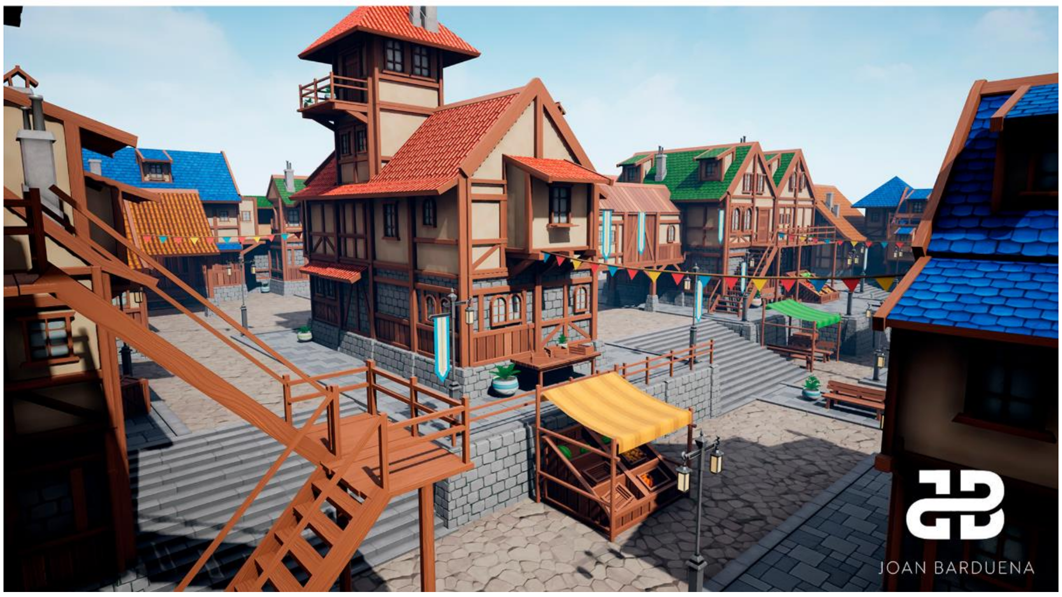


Figure 5.16.3 *Asset package screenshots – 3*

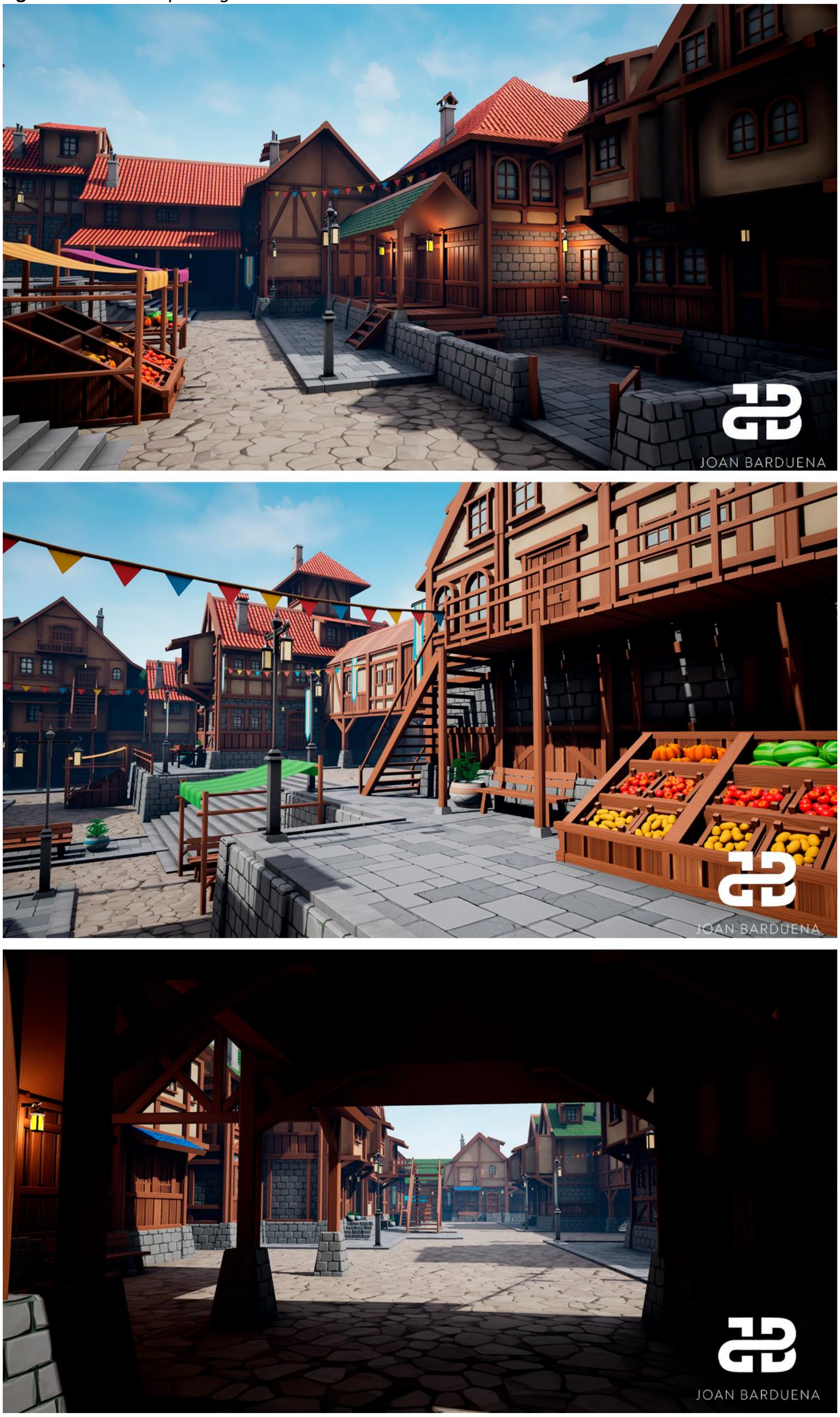
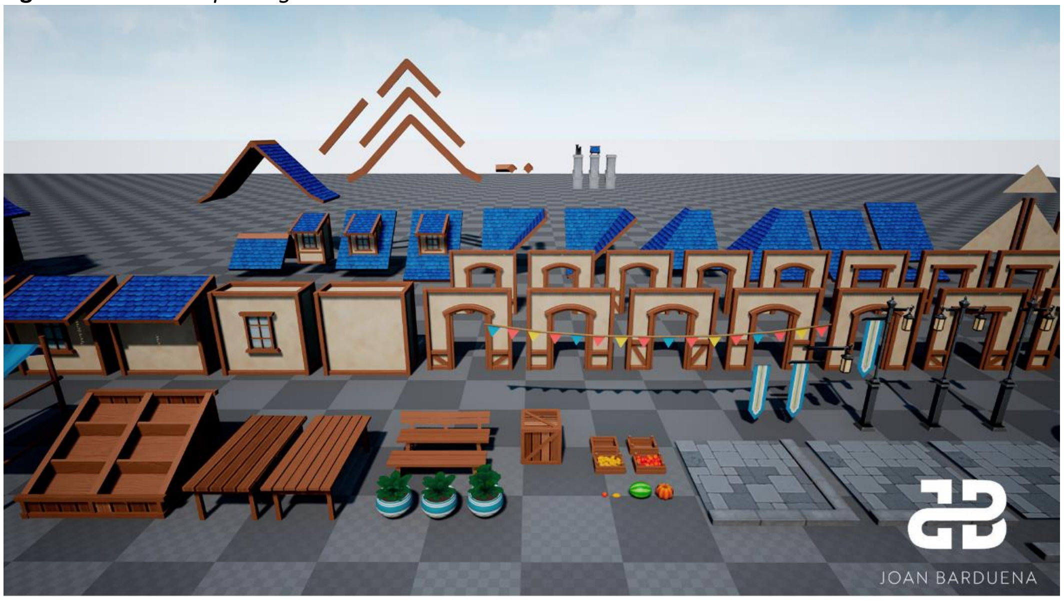
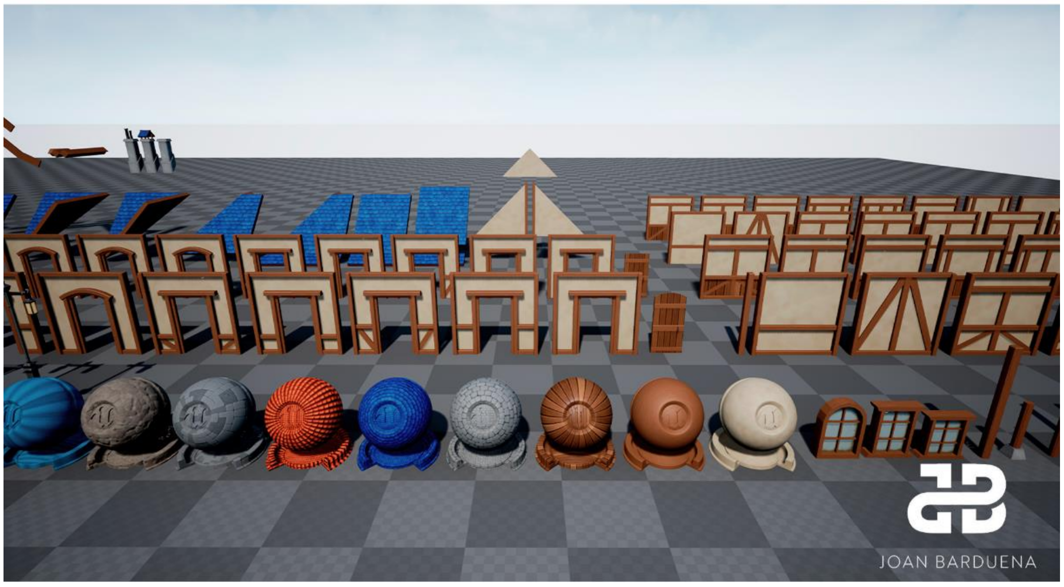
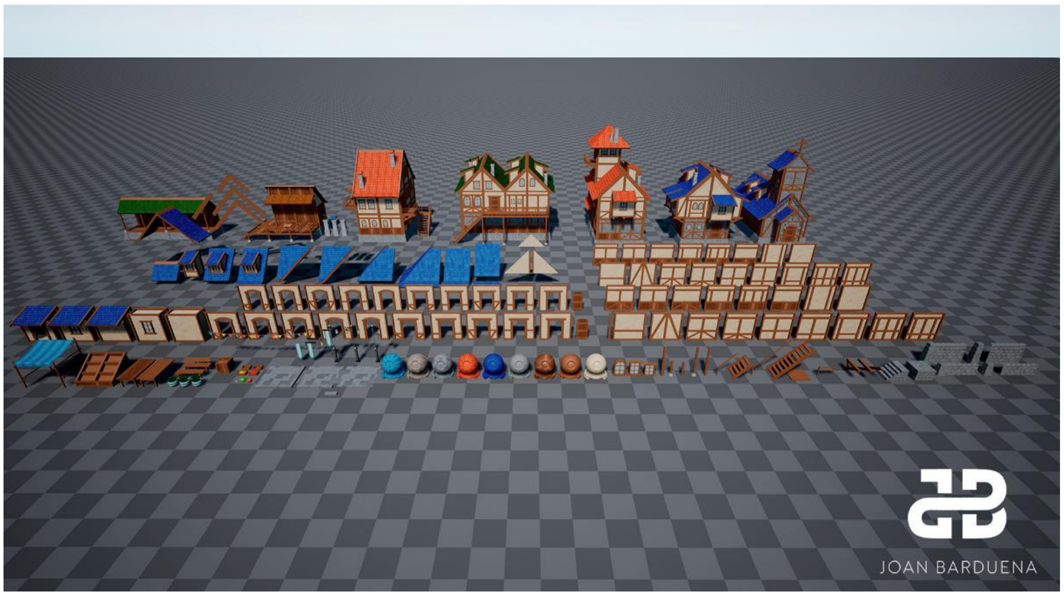


Figure 5.16.4 *Asset package screenshots – 4*

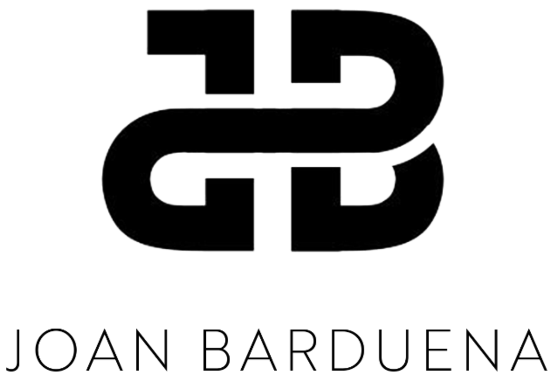






As seen in the screenshots, a brand logo was created for this project. It represents the letters JB, the asset package author's initial letters.

Figure 5.16.5 *Project brand logo*



After this is done, a video trailer of the asset package is developed. This video shows the DemoScene from different angles from a cinematic and a Third Person Character perspective. Besides, the PropsScene is shown with all meshes, materials and prefabs created.

- This video was uploaded to YouTube: https://youtu.be/HXrqzWvX4gs

5.16.1 Becoming an Unreal Engine Marketplace Seller

After having the screen shots and the video, everything is ready to publish the project. We enter to Unreal Engine Marketplace and create a Seller account. This process had to be done in advance as it can take up to 15 business days to have the account validated. This is a very serious process as Unreal ask for a lot of personal information. Also, some bank account or PayPal needs to be linked where the money will be transferred.

Several days later, the account was accepted to be part of the seller's community. After filling all the required fields dictate by the Marketplace Guidelines[7] to upload the package, this enters into **pending approval** product status. The product is required to be reviewed for Epic Games.

The delivery date has come, and Epic Games (the company that runs Unreal Engine) only does a review once a day for the product. The asset package **failed** for folder problems, and it was still in **Pending Approval** (*Figure 5.16.6*), so the link disposed below is a link to the asset package itself. Once the asset package is accepted by Epic Games, the link will contain a .txt file with a link to the Unreal Engine Marketplace, instead of a link to the actual asset pack.

[7] Unreal Engine Marketplace Guidelines https://www.unrealengine.com/en-US/marketplace-guidelines (consulted on 23 June 2022)

Stylized Medieval Village – Joan Barduena:

https://www.mediafire.com/file/f6cfz98fkgn8tbs/BarduenaJoan_MedievalAssetPack.zip/file

Figure 5.16.6 *Project Pending Approval*

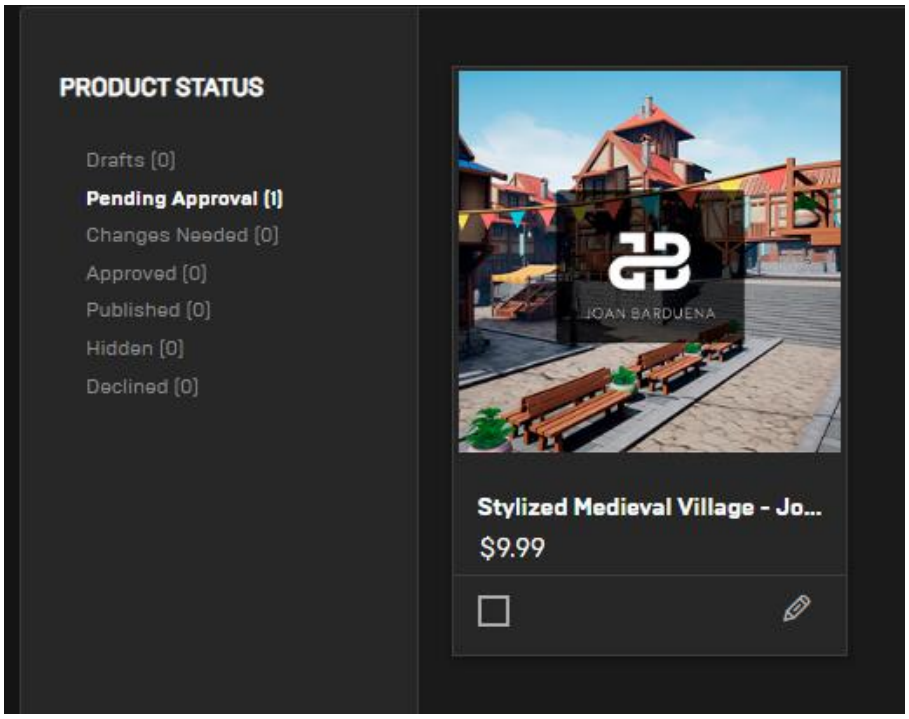
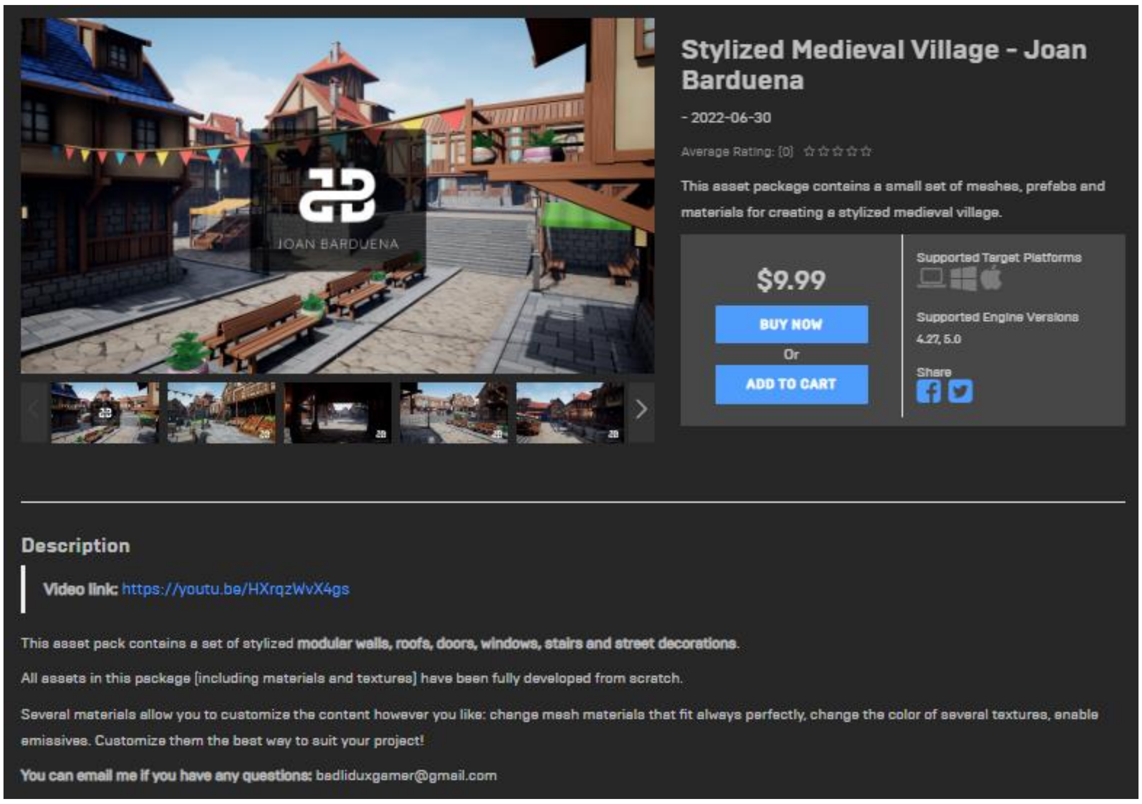


Figure 5.16.7 *Project Preview on Unreal Engine Marketplace*



This asset package contains a small set of meshes, prefabs and materials for creating a stylized medieval village.

Video link: https://youtu.be/HXrqzWvX4gs

This asset pack contains a set of stylized **modular walls, roofs, doors, windows, stairs and street decorations.**

All assets in this package (including materials and textures) have been fully developed from scratch.

Several materials allow you to customize the content however you like: change mesh materials that fit always perfectly, change the color of several textures, enable emissives. Customize them the best way to suit your project!

You can email me if you have any questions: badliduxgamer@gmail.com

6. Conclusions and future work

To conclude with the project, it can be seen that in a few months it has been possible to develop a decent asset package satisfactorily. Anyone who is aiming to create their own stylized medieval village with thousands of different combinations, can do so without any experience in three-dimensional asset workflow with this project. Having said this, the main objective of this project has been reached, which was developing a ready and easy-to-use asset package and publishing it in the Unreal Engine Marketplace.

Taking into consideration all the objectives set at the beginning of the project, it can be affirmed that all of them have been accomplished. Nevertheless, some objectives could be more fulfilled, for example creating a wider variety of building pieces and street decorations, which could be contemplated in the future. All these objectives could be achieved since the planning and management phases has been followed, trying to have all for each milestone date. However, as several problems occurred there were weeks where work accumulated, and crunch was necessary. This happened especially at the end of the project, as at the planning phase, no margin week was left. As a consequence of this, there were some testing sessions but there was no time to document them on the project.

When uploading the asset package into the UE Marketplace, it has been noticed that this asset package lacks several things to compete against some of the other projects studied in the market analysis. That's why the price is lower than expected. Additionally, there aren't perceivable bugs, but with a better implementation, several materials could be deleted using better UV displacement and more trim sheets.

For future work, asset package might have a wider variety of props and materials as well as resolving the bugs, errors and possible improvements told on the review section of the UE Marketplace. Furthermore, it could be good to adapt the package into Unity and also upload it to the Unity Asset Store.

7. Bibliography & Webography

- Unity. "Game Development Terms". n.d.
  https://unity.com/how-to/beginner/game-development-terms

- Rodriguez, Andres. "Making The Game World Come Alive". *80LV*, 23 June 2016.
  https://80.lv/articles/andres-rodriguez-environment-art-interview/

- Statham, Nataska. "Game environment art with modular architecture". *Entertainment Computing Volume 41*, March 2022.
  https://www.sciencedirect.com/science/article/pii/S1875952121000732

- Jones, Scott. "Investigation into modular design within computer games". *Pages 7 to 11*. May 2011.
  http://wiki.polycount.com/w/images/7/70/Investigation_into_modular_design_within_computer_games_v1.0.pdf

- Shah, Karan. "Sculpt, Model and Texture a Low-Poly Skull in Blender". *EnvatoTuts+,* 9 June 2009.
  https://cgi.tutsplus.com/articles/sculpt-model-and-texture-a-low-poly-skull-in-blender--cg-7

- Toma, Jeroen. "Insight into Game-Ready Asset Workflow for Baldur's Gate 3". *80LV*, 4 February 2021.
  https://80.lv/articles/insight-into-game-ready-asset-workflow-for-baldur-s-gate-3/

- Charré, Stéphane. "Using Asset Packs for Environment Design". *80LV*, 29 January 2018.
  https://80.lv/articles/using-asset-packs-for-environment-design/

- Fox Render Farm. "What is the difference between pre-rendering and real-time rendering?", 8 September 2020.
  https://www.foxrenderfarm.com/share/what-is-the-difference-between-pre-rendering-and-real-time-rendering

- Denham, Thomas. "Texture Maps: The Ultimate Guide For 3D Artists". *Concept Art Empire.* n.d.
  https://conceptartempire.com/texture-maps/

- A23D. “Different maps in PBR Textures”. n.d.
  https://www.a23d.co/blog/different-maps-in-pbr-textures/

- Unreal Engine. “Marketplace Guidelines”. n.d.
  https://www.unrealengine.com/en-US/marketplace-guidelines

- Torarev, Kirill. “Realistic vs. Stylized: Technique Overview”. *80LV.* 12 December 2017.
  https://80.lv/articles/realistic-vs-stylized-technique-overview/

- Lazar, Marie. “Creating a Stylized Chaparral Environment in UE4”. *80LV.* 8 December 2020.
  https://80.lv/articles/creating-a-stylized-chaparral-environment-in-ue4/

- Burton, Arti. “Making a Stylized Grove in 3ds Max, ZBrush & Unreal Engine”. *80LV.* 13 June 2022.
  https://80.lv/articles/making-a-stylized-grove-in-3ds-max-zbrush-unreal-engine/