

Go To Hell: tech doc.

Group 18

Casandra : Artist

Ivo : Artist

Luca : programmer

Michiel : programmer

Table of contents

Table of contents	2
Intro	3
Development stack	3
Art Software:	3
Development Software:	3
Game Engine	3
Render Pipeline	5
Units & Metrics	6
Texture settings:	9
Target Platform	10
Target Audience	10
Target Context	11
Programming guidelines	12
Name Conventions:	12
Practices:	12
Folder structure	14
Naming conventions	16

Intro

"Go to Hell" is a multiplayer couch co-op game about two detectives dismantling a secret cult ritual. The detectives arrive too late and are then tasked with fighting the demon that the cultists summoned. Player one must then astral project into a different plane to inflict damage on said demon, while player two must protect his friend's defenseless body from the demon's minions. They must take turns astral projecting, for it is dangerous to stay in the astral plane for too long. The longer they stay in the astral plane and the more damage they take, the more sanity they lose. Losing sanity will grant the detectives buffs and debuffs, but running out of sanity will knock them unconscious. If both players go unconscious, you lose the game. Fight your way to the demon and defeat it to win the game.

Development stack

Art Software:

Software	Use	Version
Maya	Modeling/animation and rigging	2023
ZBrush	Sculpting	2021
Substance Painter	Textures	Latest
Procreate	Textures	Latest
Photoshop	Textures	Latest
Materialize	Textures	1.78

Development Software:

JetBrains Rider	IDE	2023.3.3
Unreal Engine	Game Engine	5.3.2
C++	Programming Language	C++20
Perforce P4V	Source control	Latest
HacknPlan	Project Management	Latest

Game Engine

Why are we using Unreal Engine instead of Unity? While Unity is often regarded as more intuitive and user-friendly than UE5, we've opted for the latter to demonstrate our proficiency in handling more complex systems. Unreal Engine 5 offers the advantage of showcasing our ability to work with intricate programming environments. Its dual programming systems, C++ and blueprints, present an additional challenge that we're eager to tackle. While some developers may rely solely on blueprints in Unreal, we recognize the benefits of leveraging both systems for optimal performance. Moreover, UE5 boasts exceptional rendering capabilities, which are crucial for our horror-themed game where visuals play a pivotal role. Additionally, considering our game will target consoles, UE5 is a preferred choice over Unity, which is more tailored to mobile game development.

IDE:

We will be using JetBrains Rider as the IDE. Rider has the ability to enforce naming conventions, ensuring consistency across the codebase. It provides features such as code autocompletion and suggestions for improving code quality, thereby reducing the time required for troubleshooting and optimizing code. Rider is well-suited for Unreal Engine development; it offers seamless integration with standard Unreal classes and can efficiently auto-generate them, thereby accelerating the production process.

Plugins:

We will utilize the “Enhanced Unreal Documentation” plugin for Rider. This plugin provides easier access to Unreal documentation directly within the IDE, eliminating the need to search for information online.

Render Pipeline

Forward rendering:

The forward renderer in Unreal is great for simple scenes, with a small number of lights and few overlapping lights. It is good at rendering transparency, and it's optimized well for VR. It doesn't have support for screen space effects however, and it's missing some translucency options compared to the deferred renderer.

Deferred rendering:

Deferred rendering in Unreal works great with larger amounts of lights and overlapping lights, as it uses multiple passes to render out the final image. It also has more support for translucent materials, something that will be critical for the astral projection part of the game. It also supports screen space effects.

While it has more options for translucent shaders compared to the forward renderer, it is not as efficient when rendering transparent objects.

We will be using the deferred renderer in Unreal Engine going forward.

While the forward renderer has its strengths and has better performance in some areas, quite a few of them aren't relevant for our game, so we believe the deferred renderer is still the best choice for our project. This is mainly due to the fact that it supports more screen space effects (SSR, SSAO and contact shadows), and having many overlapping lights will have a smaller effect on performance. Since the game will be set in a dungeon with multiple smaller lights such as candles and torches, and lights will be moving around a lot, the deferred renderer will be more efficient.

Units & Metrics

The entire project will make use of the **metric system**. Because it is based on powers of 10 it is easy to scale things up and down, no matter the program.

We will try to keep the scale in Unreal as consistent and close to real life as possible. While the game is stylized, it is still better for both gameplay and visuals (lighting).

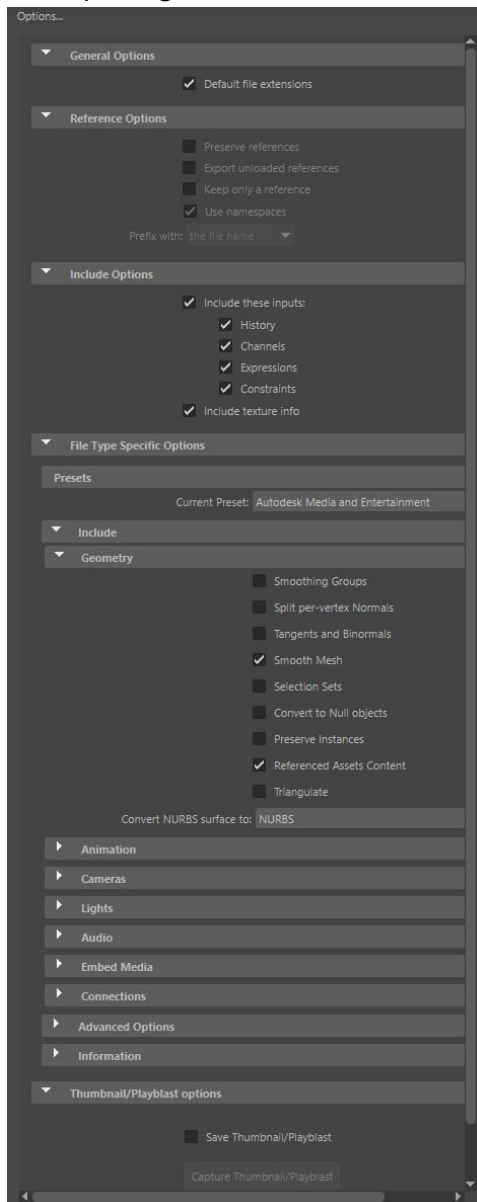
1 unit in Unreal is 1 cm, in Maya it is the same.

We will always do final FBX exports from Maya, so we will check the size there. This way there should be no scaling in the game engine.

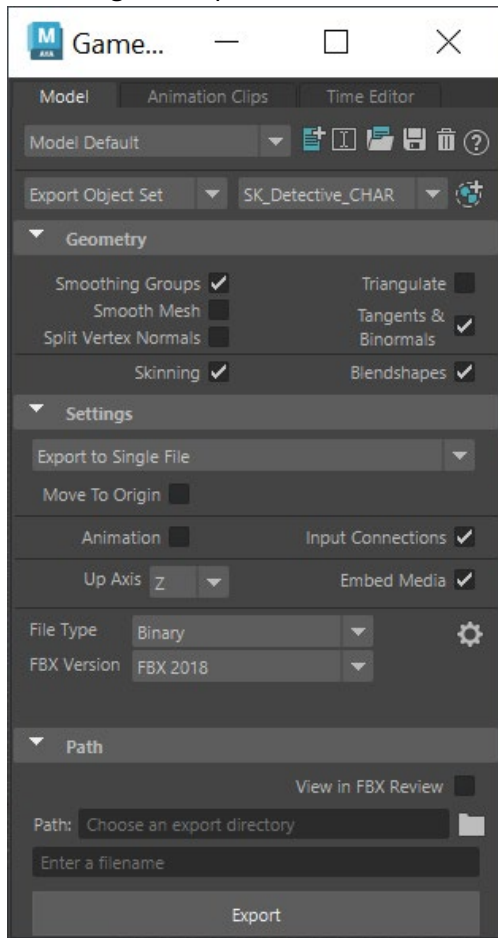
The framerate for animations will be 30 fps.

Export settings:

For exporting **static meshes with no rig** from Maya:



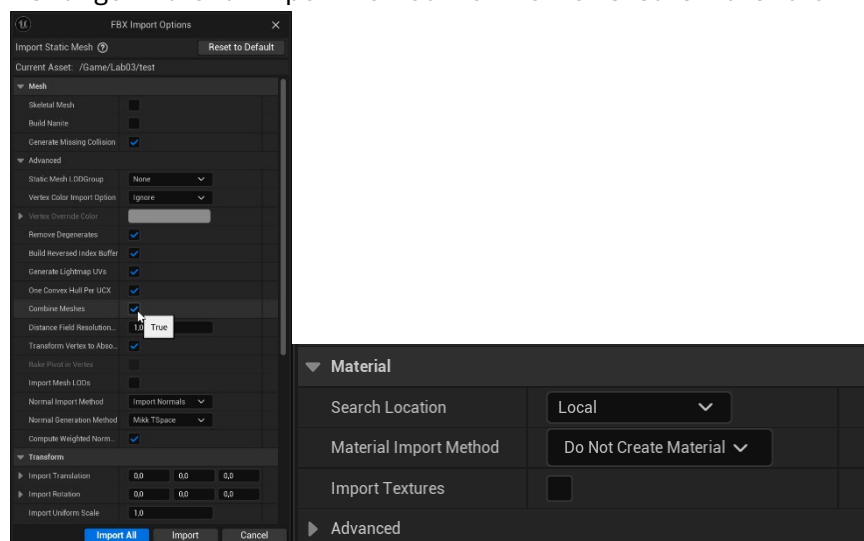
When exporting **rigged meshes** such as characters and enemies from maya:
Use the game exporter with the following settings:



Make sure the **object set name** has the right naming convention,
and pay attention to the **up axis**.
Check the **animation** box when exporting animations.

When importing into Unreal:

- Check the **“Combine Meshes”** box in the Advanced tab.
- Change **“Material Import Method”** to **“Do Not Create Materials”** in the Materials tab

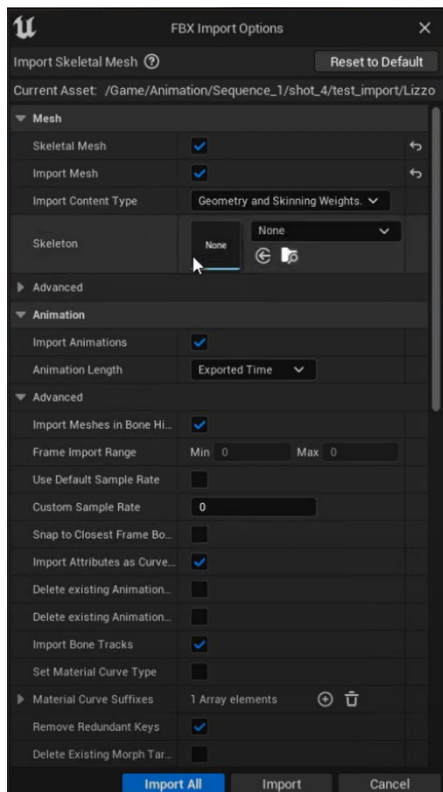


For skeletal meshes:

- Check the **”Skeletal Mesh”** option and leave the rest of the settings to the default values.



For animations:



Texture settings:

Textures will be saved as **.tga (targa)** files.

Texture resolution will be **1024x1024 (1K)** as much as possible. If extra resolution is really needed, a texture can be **2048x2048 (2K)** but **only in select cases** on important assets, like a main character for example.

Texture maps used will be:

Base Color

Normal Map

ORM Map (Opacity, Roughness, Metalness. Packed into R, G, and B channels respectively.)

Emissive Map

Where possible, small, simple props should share textures to improve performance.

Target Platform

The game is intended to be a console exclusive game, like PlayStation and Xbox, as the game is intended to be played with a minimum of 2 controllers. We plan the game to be a free playable demo.

Target Audience

Our target will be people over the age of 15, who enjoy playing games together with their friends. They enjoy chaotic party games, and games that focus on teamwork and communication.

Examples of games they might like:

Overcooked, Unrained, Pico Park, Pummel Party, Moving Out.

Example profiles:

Name: Alex

Age: 24

Occupation: Marketing Assistant

Interests: Video games, board games, spending time with friends

Bio:

Alex is a 24-year-old marketing assistant who has always had a passion for gaming. Whether it's after a long day at work or during a weekend hangout with friends, Alex loves to unwind by diving into chaotic party games that encourage teamwork and communication. Growing up playing games like Mario Party and later discovering different indie gems, Alex thrives in the fast-paced, cooperative nature of these titles. As someone who values social interactions and creating lasting memories, Alex is always on the lookout for new multiplayer experiences to enjoy with friends. With a competitive streak tempered by a love for collaboration, Alex is always seeking out opportunities to laugh, strategize, and triumph alongside his gaming companions. Outside of video games, Alex also enjoys different kinds of board games, further showcasing his love for cooperative and imaginative gameplay experiences.

Name: Jamie

Age: 30

Occupation: Teacher

Bio:

Jamie is a social and fun-loving individual who enjoys spending time with friends. As a teacher, they value teamwork and communication, both in and out of the classroom. Jamie loves playing chaotic party games with friends, finding joy in teamwork and shared laughter. He gravitates towards games like Overcooked and Pummel Party for their fast-paced, chaotic gameplay, and with a knack for problem-solving, Jamie can quickly adapt to changing situations, whether in games or real life. Outside of gaming, Jamie enjoys outdoor activities like picnics in the park and weekend hikes. Jamie's primary goal is to have a good time and create lasting memories with friends through gaming.

Bonding: Through gaming, Jamie aims to strengthen friendships by fostering teamwork and communication.

Target Context

As 'Go to Hell' is a co-op game, it is meant to be played together with friends, and cooperation and communication will be quite important. We imagine the game will be played together on a big screen in the living room, with the players all working together. While the game has some horror elements, it's still meant to be somewhat lighthearted, with a cartoony style. Because of this the game is well suited to be played as a party game in a more casual setting.

Alternatively, if multiplayer support is added, players can play together online, while talking to each other through voice chat in Discord for example.

Programming guidelines

The programming guidelines of this project are based upon the coding standards of Unreal Engine 5 (<https://docs.unrealengine.com/5.0/en-US/epic-cplusplus-coding-standard-for-unreal-engine/>). The Object-oriented programming model (OOP) will be used in this project to make use of its principles: encapsulation, abstraction, inheritance, polymorphism. All code should be written in English, as this is the main language for the entire project.

Name Conventions:

- Add prefix `m_` to member variables.
- All **variable names** (except for static variables) will use **lowerCamelCase** and **no underscore between words**, for example `healthComponent` and is correct and `Health_component` is incorrect.
 - **Static variables** must be in **all caps**.
- Function names use **UpperCamelCase** and must be **clear** and **descriptive** about the functions purpose and/or return value. Comments should be added to make this clearer but should not be relied upon.
- **Enums** start with a **capital letter**

The following prefixes are added to **classes inherited** from **default Unreal classes**, as well as to certain variable types.

type	prefix	Example
UObject	U	class UActorComponent
AActor	A	class AActor
Template classes	T	class TAttribute
SWidget	S	class SCompoundWidget
Abstract classes	I	class IAnalyticsProvider

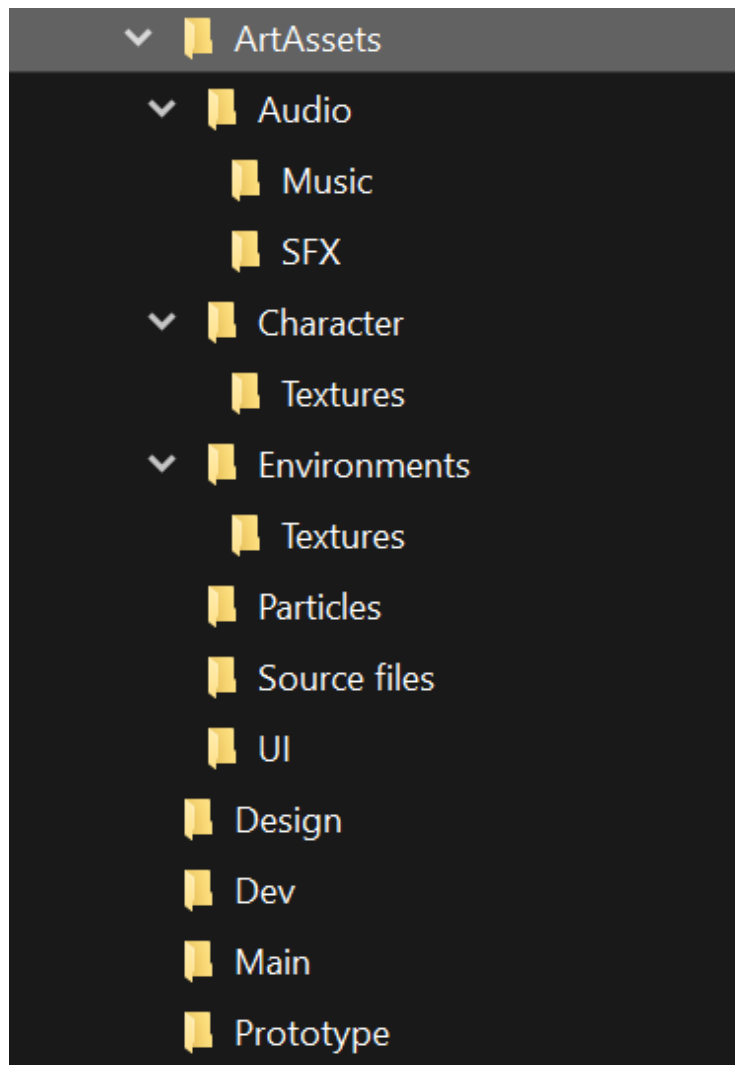
Practices:

- Each variable should be declared on a new line.
- If a function returns a variable its name should indicate what value it returns.
 - Don't do: `bool GetAstralProjecting()`.
 - Do: `bool IsAstralProjecting()`.
- Avoid nested if statements by grouping conditions together.
- Add **comments** at the **top** of each **header** to state the intent of the class.
- Comments can be used to place todos or remove code for testing but should be removed when done.
 - Example: `//todo: add way for player to hand item to another player.`
- Clear variable names that give meaning to your variable.
- Only expose values if they are needed by the artists or for general balancing.

- Keep your code readable and digestible by splitting it into functions where necessary.
- All code must be **const correct**.
- COMMENTS
 - Code should be understandable for non-programmers.
 - Comments can be used to split up large parts of code and give them more explanation.
- Clear and clean blueprints
 - **Comment code in blueprints** to make their **purpose clear** and **easy to move**.
 - Use sequences when needed to keep code readable.

Folder structure

Perforce:



Unreal:

The top materials folder (_Materials) must start with an underscore, so it will always be displayed at the top in the unreal file browser. The folders highlighted in red are just example folders for this document. In the project these would be created for the different objects, characters etc.

If we take the “Candles” folder as an example:

In the candles folder itself there will be static meshes, with materials applied, ready to be added to the level.

The source folder will contain all the source files like textures, material instances, etc. that are needed for the candle props.



Naming conventions

Both working files and exports will be in **UpperCamelCase**, starting with a capital letter.

Keep the name **Short and descriptive**.

The prefix/suffix will always be in **capital letters**.

Working files will use prefixes and exports will use suffixes. (see examples in tables below)

If there are multiple similar files, for example multiple texture psd's for different pillars, the names can be the same, but they will be numbered, starting with 01 and counting up (there will be an underscore between the name and the number, just like with prefix/suffix).

There will be an underscore between the prefix and the name. For example:

T_ObjectName.psd

T_Pillar_04.psd

Working files:

Type	Prefix	Examples
Environment	ENV_	ENV_WallPieces.ma
Character	CHAR_	CHAR_Detective.ztl CHAR_Monster.ma CHAR_Monster.psd
Props	PROP_	PROP_Candle_01.ma
RFX (Particle)	FX_	FX_FlameSprite.psd
Animation	ANIM_	ANIM_DetectiveWalk.ma
Audio files	AUD_	AUD_BackgroundMusic

Exports:

Exports will be using suffixes, this is so that in the engine files will stay together based on the asset name, and not the type. For example: Monster_E, monster_N, Monster_BC will all be grouped when sorted alphabetically, if there would be a suffix different emission maps will all be together, making it harder to find all maps for a certain prop or character.

Also, prefixes can be added in Unreal, while still keeping the suffixes from the export files.

Type	Suffix	Examples
Environment 3D	_ENV	Pillar_01_ENV.fbx
Character 3D	_CHAR	Detective_CHAR.fbx
Base color texture	_BC	Monster_01_BC.png
Packed texture with Opacity, Roughness and Metalness	_ORM	Monster_01_ORM.png
Normal map texture	_N	Monster_01_N.png
Emissive texture	_E	Monster_01_E.png

In engine:

Type	Prefix	Examples
Material	M_	M_BasicMaterial
Material Instance	MI_	MI_Detective
Static Mesh	SM_	SM_Pillar_01_ENV
Skeletal Mesh	SK_	SK_Detective_CHAR
Texture	T_	T_Monster_01_N T_Detective_BC
Levels	LV_	LV_Dungeon
Blueprints	BP_	BP_Player
Niagara Emitter	E_	E_Flame
Niagara System	NS_	NS_Explosion