‘The Seas Will Bury Us’ TDD

Lathraia Games

Justin Green, Ethan Dawkins

# Contents

[2 Contents 1](#_Toc177324205)

[3 Team Members & Roles 2](#_Toc177324206)

[3.1 Design 2](#_Toc177324207)

[3.2 Programing 2](#_Toc177324208)

[3.3 Art 2](#_Toc177324209)

[4 Revision History 2](#_Toc177324210)

[5 Development Environment 2](#_Toc177324211)

[5.1 Game Engine 2](#_Toc177324212)

[5.2 IDE 2](#_Toc177324213)

[5.3 Plugins 2](#_Toc177324214)

[5.4 Other Software 2](#_Toc177324215)

[5.5 Source Control 3](#_Toc177324216)

[5.5.1 Source Control Style Guide 3](#_Toc177324217)

[5.5.2 Source Control Procedures 3](#_Toc177324218)

[6 Controls & Perspective 4](#_Toc177324219)

[6.1 Perspective 4](#_Toc177324220)

[6.2 Control Schemes 4](#_Toc177324221)

[6.2.1 Keyboard & Mouse 4](#_Toc177324222)

[7 AI / Behavior Trees 5](#_Toc177324223)

[7.1 Main Enemy 5](#_Toc177324224)

[8 Technical Goals 5](#_Toc177324225)

[8.1 Main Goal - Immersion 5](#_Toc177324226)

[9 File & Programming Naming Standards 6](#_Toc177324227)

[9.1 Directory structure 6](#_Toc177324228)

[9.2 Coding Conventions 6](#_Toc177324229)

# Team Members & Roles

|  |  |  |
| --- | --- | --- |
| Design | Programing | Art |
| Yash Verma | Justin Green  Ethan Dawkins | Valentina Sequeira  Micki Huang |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Editor** | **Additions, Changes and Removals** |
| 0.0 | 13/09/2024 | Justin | Initial Document Setup – 1.0, 2.0, 3.0, 4.0,5.0 |
| 0.1 | 14/09/2024 | Ethan | Added 8.0, 9.0 |
| 0.2 | 15/09/2024 | Jusitn | Added 6.0,7.0,10.0 |

# Development Environment

## Game Engine

This Game will be created in the Unreal Engine (**Unreal 5.4)**. This engine is an industry standard 3D game development software and was chose because of the team’s prior experience with the engine and its free upfront cost. It is important to note that **royalties will apply after $1 million USD (*$148,590.00 AUD*) in annual gross product revenue** as stated on the [Unreal’s download page](https://www.unrealengine.com/en-US/download).

## IDE

The IDE chosen for this product is Visual Studio Community 2022 (**VSC22 17.11.3)**. This is because it is Unreal’s default and preferred IDE and provides Benefits such as quicker symbol loading.

## Plugins

The only plugin used in this project at this version is the **Visual Studio Integration Tools.** This enables better integration with the IDE.

## Other Software

|  |  |
| --- | --- |
| **Software** | **Purpose** |
| Word | Documentation |
|  |  |

## Source Control

The version control system used for this project is Git and the version control client is GitHub

### Source Control Style Guide

To promote clarity and collaboration, a source control style guide will be put in place. I will be using a modified version of an existing [GitHub Style guide by git user Shinjith](https://gist.github.com/shinjith-dev/1c95763fe3aa0df8e2af233ae4f36686#commit-message-conventions). Commit messages should go as follows:

type(s)**(**scope**):** short description

optional body

#### **Type Scope and Message Body**

A type prefix in the subject line can be used to represent type of the changes included in the commit. Some of the commonly used types are:

* Add: To summarize a new addition in the project.
* Mod: To summerize a modification made to a previous addition.
* Fix: To address a fix to the project.

**Mod/Fix(**TutorialLevel**):** Changed level layout and fixed clipping issues

types can also be combined using the / symbol to provide further context. For example:

A scope is added to a commit's type to offer additional contextual information, and it is enclosed in parenthesis, for example:

Add(**BP\_Player**): Created Player BP.

Mod(**BP\_Player & Locomotion**): Using blueprints player now has locomotion.

A body can be added to the message to include detailed explanations in the commit.

### Source Control Procedures

The Git repository will be hosted through GitHub and will consist of 2 major branches being an origin/main branch, and a development branch, and any feature branches. Any big features will be developed separately within their own feature branch which is a snapshot of the development branch and will then be safely merged into development branch when completed, the branch will also be deleted once a feature has been finished and merged into development branch.

In the diagram below the development branch is a sub-branch of the origin / main meaning it is a modified version of the origin. Other feature branches will then be a snapshot of the development branch.

Having our Git architecture setup like this means that development will always be playable in the editor, and the origin / main will always be a playable build of the game. This will also reduce work being overwritten as ideally there should be minimal to no overlap of actively worked on assets.

Also as a sidenote, working builds will be made every 1-2 weeks to allow for a mostly up-to-date build to show off at all times.

#### Source Control Procedure Diagram

# Controls & Perspective

## Perspective

This game will be in first person. The reasoning for this is immersion, which plays a big part in a horror game, the first person perspective accentuates the claustrophobic feeling we want to mimic.

## Control Schemes

### Keyboard & Mouse

|  |  |
| --- | --- |
| **Input** | **Behaviour** |
| Mouse X,Y | Rotates player camera accordingly. |
| WASD | Controls player’s forward, left, back and right movements |
| Space | Allows player to jump |
| Ctrl | Allows player to crouch |
| Shift | Increases the player movement speed |
| E | Allows player to interact with interactable |

# Custom Systems & Game Mechanics

## Core Mechanics

|  |  |
| --- | --- |
| Movement | Gravity affected 3D directional movement, including features such as crouch, jump and sprint. |
| Interaction | Player will be able to interact with certain interactable such as doors or pages that will trigger a UI element. |
| Crafting | Player can find alchemical components in the map and can use these components to craft a different flame for their lantern which apply different gameplay effects. |
| Use Lantern | Player will be able to use their lantern and depending on the colour affects the world and enemies differently |

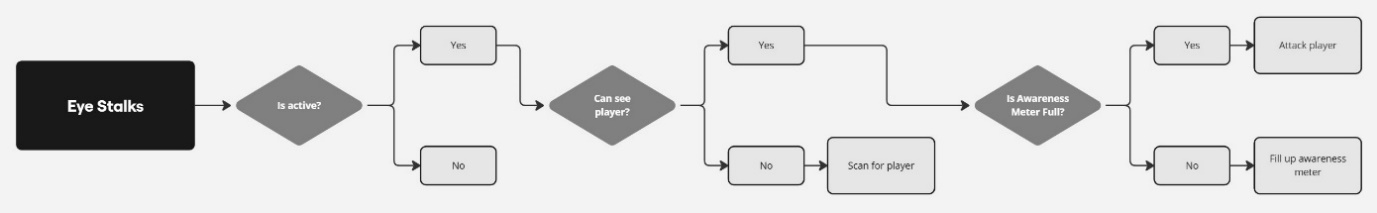
## Custom Game Systems

### Alchemical Crafting

The player will encounter obstacles that require certain alchemical creations to surpass. This creation requires components that are found in a level and is shown as a coloured flame depending on the components used during creation.

# AI / Behavior Trees

## Main Enemy

The main enemy/s will be in the form of ever-present eye stalks. These eye stalks will be at set points within each level, and act as a sort of CCTV camera. When the player is within the view cone of the eye, an awareness meter shall go up, when this awareness meter is full it will attack the player. Each individual eye stalk will have two states; open or closed, when closed the player can sneak past them and can not be seen, but when the eye is open it will run the vision check.

# Technical Goals

## Main Goal - Immersion

Our main goal for this project is to achieve immersion through AI and player interaction. We plan on doing this by having the main enemy ‘adapt’ to the player as they traverse through the level. The enemy will adapt to the player by switching its’ active eye stalks, it will do this when either the player interacts with a tendril, or stuns an active eye stalk.

# File & Programming Naming Standards

This project’s file and naming standards come from two sources. All code naming conventions will come from the official [Epic Coding Standard](https://dev.epicgames.com/documentation/en-us/unreal-engine/epic-cplusplus-coding-standard-for-unreal-engine)s which is considered the standard and best practice for writing maintainable code. All other structure and style will adhere to an unofficial but very succinct [UE5 Style Guide](https://dev.epicgames.com/documentation/en-us/unreal-engine/epic-cplusplus-coding-standard-for-unreal-engine), sponsored by Git User Michel Allar.

## Directory structure

|-- **Content**

|-- **GenericShooter**

|-- **Art**

| |-- Industrial

| | |-- Ambient

| | |-- Pipes

| |-- Nature

| | |-- Ambient

| | |-- Foliage

| | |-- Trees

**|-- Characters**

| |-- Bob

| |-- Common

| | |-- Animations

| | |-- Audio

| |-- Jack

**|-- Core**

| |-- Characters

| |-- Engine

| |-- GameModes

| |-- Interactables

| |-- Pickups

| |-- Weapons

**|-- Effects**

| |-- Fire

| |-- Weather

**|-- Maps**

| |-- Campaign1

**|-- MaterialLibrary**

| |-- Debug

| |-- Metal

| |-- Paint

| |-- Utility

| |-- Weathering

**|-- Placeables**

| |-- Pickups

**|-- Weapons**

|-- Common

|-- Pistols

| |-- DesertEagle

| |-- RocketPistol

|-- Rifles

To lay out content, the following approach relies on the search and filtering capabilities of Unreal’s content browser for working with assets of a specific type.

For an in depth explanation of the file structure click [here](https://github.com/Allar/ue5-style-guide/tree/v2?tab=readme-ov-file#2-content-directory-structure).

## Coding Conventions

Coding naming conventions used will be mainly PascalCase alongside using prefixes depending on parent class, the only exception to this rule will be Booleans which are prefixed with a lowercase ‘b’, for example ‘bTimerFinished’.

Comments will be used sparingly throughout the code as good code should not require extensive comments to be readable and understandable. As a rule of thumb, “Do not comment bad code, rewrite it.” This will ensure that all comments are communicating vital information to anyone who reads over it.

Code formatting conventions will be always using braces on a new line for an if/else statement to make the code more readable. Tabs and indenting will be used to separate execution blocks, for example in a switch statement each case will be indented to convey that each code block is separate.

Asset naming prefixes will depend on the main type of the file and whether it is a subtype or not, i.e. material and material instances. This will help us to manage which assets serve which purpose or functionality. As for materials and their material instances these conventions will mitigate any issues with editing the original material and affecting every other instance of the material in the project. The other naming conventions are aligned with Unreals’ recommended asset naming conventions as per the Unreal 5.4 Documentation.