BREFHÄMER

Technical Design Document

Not the Game Design Doc

Written by Jason Conger-Kallas

Version 1.00

Thursday, June 22, 2017

Table of Contents

1. Technical Overview..... 4

1.1 Description.....

1.2 Technical Goals.....

1.3 Platform.....

1.4 Frameworks and API.....

1.5 Secondary Software.....

2. Development Tools..... 5

2.1 Engine.....

2.2 Coding Languages.....

2.3 Software Compatibility.....

2.3.1 System Requirements.....

3. Pipeline Implementation..... 6

3.1 Libraries.....

3.1.1 Graphics.....

3.1.2 Physics.....

3.1.3 Audio.....

3.1.4 Video.....

3.1.5 Rendering.....

3.2 Assets.....

3.2.1 Textures.....

3.2.2 Shaders.....

3.2.3 Models.....

3.3 Effects.....

3.3.1 Collision Detection.....

3.3.2 Fluid Dynamics.....

3.3.2 Particle Systems.....

3.3.2 Draw Distance.....

3.3.2 Fluid Dynamics.....

4. Architecture..... 7

4.1 Class Names.....

4.2 Entities.....

4.3. Scripting.....

4.3.1 Instances.....

4.3.2 Modules.....

4.3.3 Flow.....

5. A.I..... 8
5.1 Pathing.....
5.2 Behavior Trees.....
5.2.1 Enemies.....
5.2.2 NPCs.....

6. API..... 9
6.1 Methods.....
6.2 Classes.....

7. Modification Guides..... 10
7.1 How to Navigate Code.....
7.2 Documentation Guide.....
7.3 Continued Support.....
7.3.1 Patches.....
7.3.2 DLC.....
7.3.3 Modding.....

8. Risk Assessment..... 11
8.1 Scope.....
8.2 Mechanics.....

1. Technical Overview

1.1 Description

The game may not revolutionize game mechanics, graphics, or technical innovation, but it will implement many different types of gameplay mechanics. The trick is going to be managing all the assets and programming systems with a small team.

1.2 Technical Goals

The production goals for indie projects should not aim to push next-gen graphics or the level of interactivity with immersive VR game environments. The game needs to use a clever combat mechanics system to make the gameplay interesting

It would be nice to develop an advanced A.I. library to draw from so that NPCs, enemies, and characters don't constantly repeat the same actions and phrases.

If the programming and art teams collectively decide they want to challenge themselves by building in other advanced features or new technology, then that will be up to them to decide.

Early implementation and testing of a minimum viable product prototype is recommended.

1.3 Platform

The game will need to be optimized for PC, Mac, and Linux. The target platform is Steam, so Open GL, Directx, and Steam API formatting standards will need to apply.

1.4 Frameworks and API

The art style and budget of the game will be a major factor in the choice of game engine software. If 2D, Gamemaker 2.0 would be the best option. If 3D or a 2D/3D hybrid, Unity is the top choice.

Middleware options or Unreal may also be considered depending on the preferences of the technical director. The platform will be determined before the crowdfunding campaign.

At this point in time, attempting to create a custom proprietary engine is not advisable.

1.5 Secondary Software

Installer.ini
Level Editor

2. Development Tools

2.1 Engine
Unity

2.2 Coding Languages
C#
JavaScript
C++
Python
C

2.3 Software Compatibility
Preferably the game will be able to run on most laptops and computers without high-end graphics cards or drivers. If viable, portability to consoles might be considered.

2.3.1 System Requirements
Minimum Specs:

1.2 GHz Intel processor
250 MB RAM
Windows 7, 8, 10
Mac
Linux
PS4, Switch, Xbox, Other Next Gen Platforms?

Settings will also need to consider factors such as resolution, framerate, and resolution.

3. Pipeline Implementation

3.1 Libraries

3.1.1 Graphics
3.1.2 Physics
3.1.3 Audio
3.1.4 Video
3.1.5 Rendering

3.2 Assets

3.2.1 Textures
3.2.2 Shaders
3.2.3 Models

3.3 Effects

3.3.1 Collision Detection
3.3.2 Fluid Dynamics
3.3.2 Particle Systems
3.3.2 Draw Distance
3.3.2 Fluid Dynamics

4. Architecture

4.1 Class Names

N/A

4.2 Entities

N/A

4.3. Scripting

4.3.1 Instances

4.3.2 Modules

4.3.3 Flow

5. A.I.

5.1 Pathing

The goal is to create algorithms that will make enemies behave responsively to certain actions.

5.2 Behavior Trees

5.2.1 Enemies

It will be necessary to reuse animations, character models, and types of stats. Creating a library of core stats and A.I. patterns that can be mixed to create different enemy and player move sets and abilities will add more variety to otherwise generic grind enemies.

5.2.2 NPCs

NPCs will follow preprogrammed routes. Preferably implementing a system that does not make them repeat the same two lines over and over again.

6. API

6.1 Methods
N/A

6.2 Classes
N/A

7. Modification Guides

7.1 How to Navigate Code
N/A

7.2 Documentation Guide
N/A

7.3 Continued Support
7.3.1 Patches
Any major bugs discovered in playtesting may need to be patched with a subsequent patch

7.3.2 DLC
DLC is not anticipated to be viable at this time, but it could be used to patch issues, add content, and include things that didn't make it into the original release

7.3.3 Modding
A creator tools kit is not required for the project scope, but it might be nice to make certain assets available for creative commons use.

8. Risk Assessment

8.1 Scope

The biggest challenges are going to be Animation, AI, and Assets.

The game needs to be approached from a minimalist stance. Anything that is unnecessary, even if it is cool, needs to be cut. If it doesn't serve the story or core gameplay, cut it.

Reducing the quality of graphics will be necessary from a practical point of view. Physics, texturing, and 3D lighting are not things an indie team want to deal with.

The art style and medium need to be established before programming anything, or the project will stall out.

8.2 Mechanics

To make the scope of the project manageable, much of the combat will likely need to be based on stats rather than animated combat. This could end up resembling an old school RPG rather than 3D open world action games.

Some combat may need to be substituted for puzzles, since there is no way all the variety of planned enemies will make it into the game.