Platformer Game Proposal

Ryan Kenfield & Jared Keefer
Date: 2/17/20

Table of Contents

Project Description:

We plan on making a locally hosted Metroid-Vania style platformer. In this game, the main character is in a dream where they are traveling though levels themed around different music genres. The core of the game will be traversing the environment avoiding/killing enemies and bosses to acquire new skills. Our planned game mechanics include;

- General platforming(i.e. Jumping, dodging, general mobility options for increasing the entertainment.)
- Multiple physical and projectile attacks for the player to use and avoid
- Damage
- AI for enemies

Technologies Used:

We will develop this game using the Unreal Engine 4 game development engine. This framework will provide us with the basic requirements to make a functioning game such as; access to the GPU for displaying graphical content, linking files and libraries, included objects and specifications for interfacing with computer firmware and other software elements, predefined libraries that provide basic game engine requirements.

The coding, by necessity, will be written in C++ using the Visual Studio IDE. This is the only Development environment supported by Unreal Engine 4. It will also provide a robust debugging platform that will allow us to identify scripting issues faster. This will, in turn, aid in the speed of our development, and allow us to design a more complete and interesting playing experience.

Version control will be handled through a public GitHub repository. This will allow us to work simultaneously on the project to increase project efficiency. In addition, this method will provide a safety net for the scenario that our game code becomes broken; we can revert to the previous running version. Again this will aid in overall project efficiency and availability of the finished product.

The game soundtrack will be composed, recorded, and mixed/mastered by Jared using Pro Tools 11, Reaper, and a multitude of vst plugins. The audio will be rendered using offline render included in the Pro Tools DAW which will then be transferred to the Unreal Engine project in the .wav file format. Utilizing the .wav format ensures the highest quality audio; but it does come with a significant space overhead. This overhead will be managed, if need be, though selective compression and alteration after the audio has been integrated into the Unreal Engine Project.

Since we are planning to source our artwork from more experienced animators and creators, outside modeling and art creation software, potentially unknown to us, will be used in the creation of our in-game models. Though it is ideal to be aware of everything that has/is being used in the production of our project, the artists we plan on working with are close friends of Ryan and we should have a easy enough time opening a dialogue with them should any conflicts arrise.

Essential Project Components:

- Unreal Engine 4 must be properly formatted and configured for our project to compile and be designed properly
    - If this engine fails or proves inefficient, we will look into using the Unity game engine. This engine is roughly the same but will require existing code to be ported into C#.
- Our project won't work if GitHub doesn’t interface well with the Unreal Engine. It is common for game development engines to keep meta files and logs of changes that have been made in hidden files within the project. This has been shown to cause some issues when attempting to use version control methods. However, there is extensive documentation about how to configure, and/or troubleshoot, issues such as this.
    - If, however, Git/GitHub proves to be insufficient, we plan on using Tortoise svn. Ryan has used this version control before with Unreal Engine with minimal hassle. GitHub was chosen as it is considered somewhat of a computer science industry standard; and an attempt to further include class material in our individual project.
- As explained earlier there may be an issue with the integration and/or space constraints when using our planned .wav audio file format.
    - If this issue arises, we plan to render the audio to the more common, lightweight, file format of .mp3. This would lead to decreased audio quality but may allow for faster integration into the game project and a smaller final executable
- If there are any issues with the art files than We will either have our outside resources fix them or find a free-to-use alternative;
    - Basic geometry placeholders
    - Free-use downloadable animation and environments

Outside Resources:

- Brandon Colley & Jacob Levine (Art/Animation/render providers)
    - Both are art students attending SCAD in Georgia
- The libraries and classes provided in the game engine
- VST plugins and audio recording software both included in the DAWs Pro Tools and Reaper.

Design Patterns:

- Flyweight
    - Within our proposed game there will be a multitude of enemies and environment variables that must be rendered and managed at any given time. Therefore, to have each of these defined completely unique would cause a large memory and GPU overhead. So to combat this we will take a more efficient approach of identifying the similarities in these game objects/characters and defining them within their own class. This will remove redundant information from memory, and allow each object to have a reference to this single memory. In turn, each object may still be unique, but the lumping of similar information into a single memory instance will reduce the game's space and GPU overhead.
- Prototype
    - Since we are planning on having multiple different enemy types in each level, multiple levels, and boss enemies our project must have a way of spawning these enemies into the game. The naive approach would be to make a class that spawns a given enemy type for all enemies. This would lead to very cluttered code that would be unnecessarily difficult to navigate/debug. To improve on this, we will use the principle of prototyping by making the parent class of all enemies have a function to clone an enemy. With this we can then re-define this method in each child class to clone the desired enemy in whichever way we see fit. This will also decrease the overall memory requirement as there is now only a single spawner that must exist in system memory when the game is running.

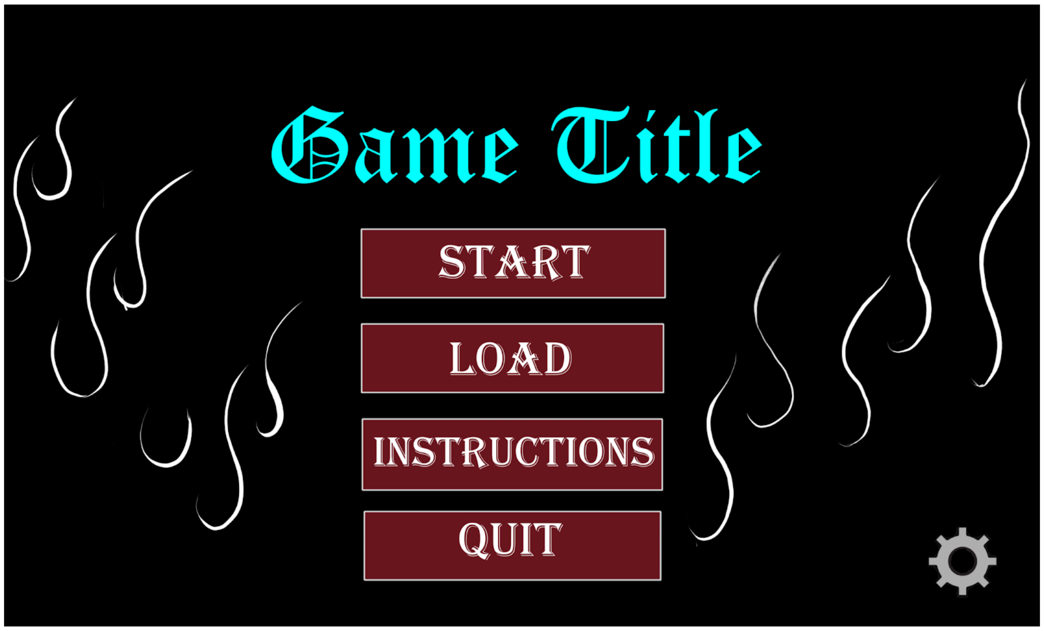
Program Architecture:

Objects and inheritance architecture:

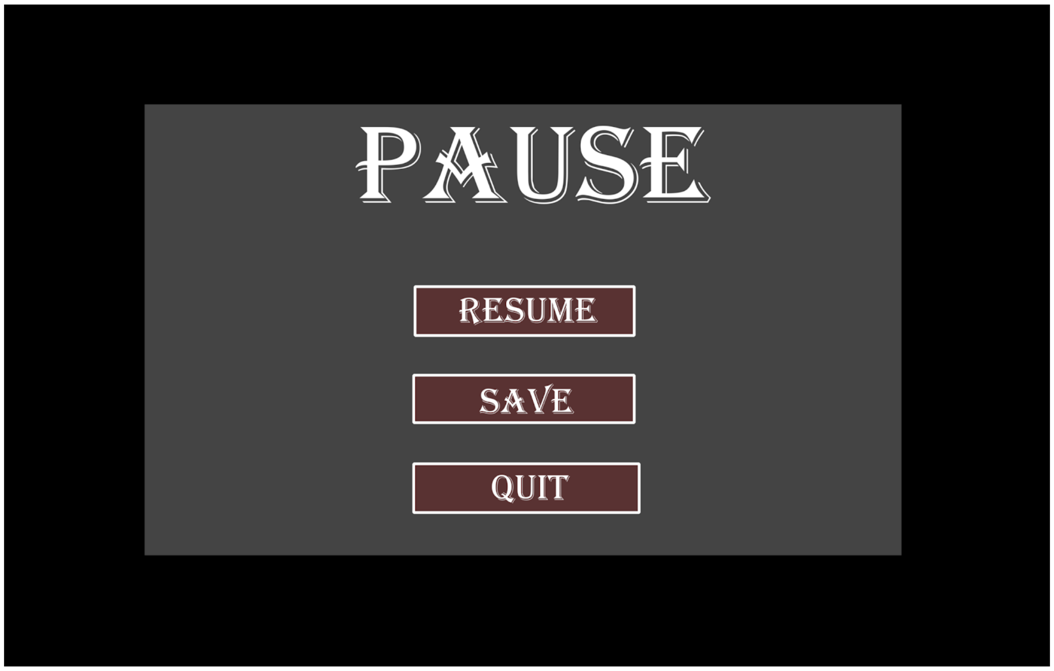
- Characters:
  - Player character
    - If time allows, we plan on having a character selection mechanism in which the aesthetics and base move-set of the character would depend on this choice.
      - This would be done with a class Player that would inherit from the parent Characters.
  - Enemies:
    - Multiple types of enemies drawing from the enemy class
      - Inherits basic physics and mechanics from the parent Character class
    - Boss enemies will be a class that inherits from the Enemies class.
      - This inheritance will provide; basic physics, collision with the player, and basic ai methods
      - Health, speed, bullet type etc…, That they also inherited from Enemies/Character parents.
- Projectiles:
  - Enemy bullets
  - Boss bullets
  - Player bullets
    - All of which can derive from a projectile class and can have their own changeable values.
- Actors in level
  - Traps to avoid
  - Possible mechanic additions (e.g. grappling hook locations)
  - Moving level pieces or any actors in level that would have code attached(e.g. Fans, bouncy tiles, accelerating pads, etc..)
    - All of which can draw from a class that shares gameplay options(e.g. Speed, collision, etc…)

GUI Design:

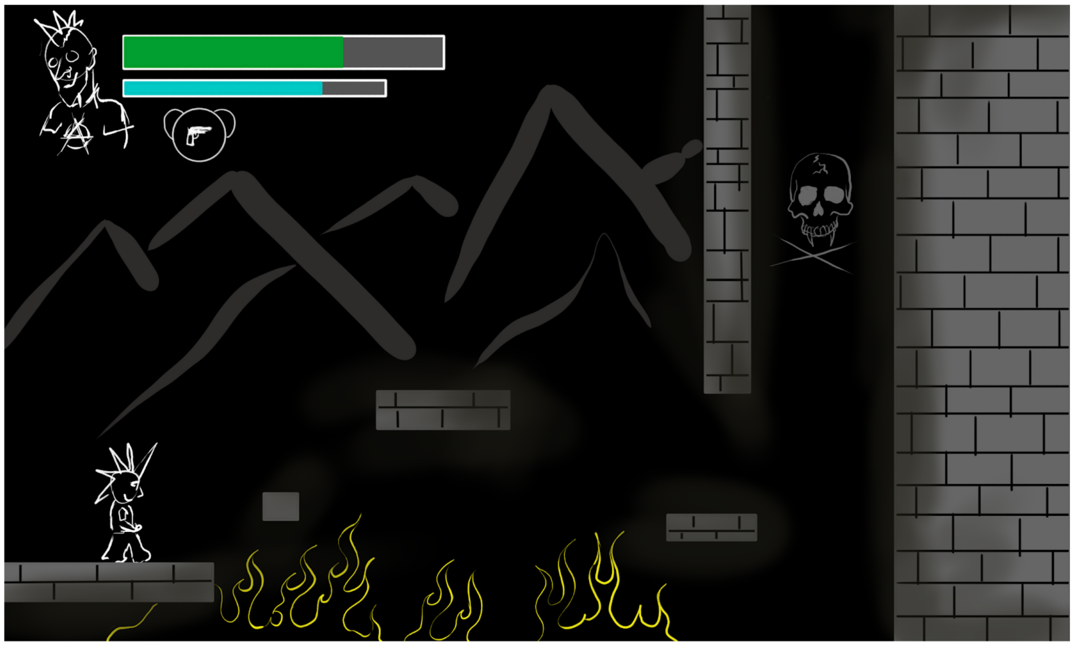
- Main Menu



- Pause Menu



- In-Game UI



Detailed Development Plan:

Our overall design approach will be collaborative in nature. We plan to have weekly meetings in which we discuss the current state of our project, any issues we have encountered, and generate a plan for what must be done over the course of the following week. Currently our intended distribution of labor is as follows;

- UE4 project layout and management: Ryan
- Level design: Jared & Ryan
- Basic character movement: Ryan
- Advanced player mobility: Jared
- Game physics: Jared & Ryan
- Collision detection: Ryan
- Game UI and navigation: Jared
- Enemy type definition and spawning: Jared
- Enemy AI: Jared & Ryan
- Game memory and code optimization: Jared

With this distribution of labor, we plan to adhere to the following development schedule;

hw2pt3) Get project set up, repository made and distributed, file structure sorted. Aim for general mechanics to be working, this includes player movement, jumping, firing. Controls for the player character should be finished and mapped. Also, basic UI for players and menus.

hw3) Attempt to get a basic enemy ai working. Aim for a basic layout of the first level, not including all of the traps/ obstacles, include the enemy and make a winnable first level. After we have this done we will aim to make some obstacles and level additions to make the level more challenging and fun. Assuming the previous components are in working order, we will add more AI. Player stats and enemy stats as well as enemy an player bullet interactions.

NOTE: All levels will be hard coded into the game source. this will allow for found bugs to be recreated more reliably. Also, this allows for more robust testing of each obstacle, enemy, checkpoint, etc...

hw4) Aim to add a possible boss for the first level and work on second and third levels(not including bosses for those). We should have Ai that we can use for these levels and can use the obstacles and added gameplay aspects to make these levels more fun. Also aim to integrate new weapon gained from killing the boss. Depending on progress with this we can add bosses for these latter levels and repeat the same process for adding new guns.

- At this point the game should be mostly made up of geometric art, excluding the main character which we can aim to have the art done for by now

hw5) After three levels in a working state, we will begin user testing. In this we will recruit a small sample of people who will receive the beta version of the game for a couple of days on the pretenses that they provide detailed notes about their experience and any bugs/flaws they see in its current form. During this testing period, we will begin adding the polish of the game in the form of; music, additional art and lighting, particle effects, etc… Also, we will begin optimizing the game code for best possible performance and smallest memory requirement.

hw5) Utilize the data collected during user testing to fix any found issues and implement quality of life/fun factor changes requested. Add more user-friendly features such as settings options, saving, etc... Add story elements and general increased interactivity.

How We Will Stay Engaged In Class:

This class is structured around improving our, as students, ability to develop and understand object-oriented programs. The core components of game development are rooted in class design, creation, management, inheritance, and memory management. Therefore, the material covered in class will provide us with methods for creating more robust game code.

Though we will not be using some of the tools that will be covered in class, the principles and good practice guidelines of their usage, provided through class material, can be used to improve corresponding game elements. For example, the class syllabus suggests that we will be covering Qt as a method to create GUIs. Since Qt uses C++, and Unreal Engine also uses C++ there will be a significant amount of crossover from class material to us developing the many interfaces in our game. In addition, the principles of GUI design covered in this module will help us to make our interfaces more intuitive to the player.

Finally, this class is taught in C++, which we plan on using for the entirety of our game development. So, the new concepts introduced in class pertaining to how to best achieve a task within C++ will provide us with new, and more effective, ways to implement the many tasks of creating a fully functioning interactive experience.