Escape Velocity

Technical Design Document

Technical Description

A 3D platforming game where the direction of gravity is constantly changing when near a gravity platform. Complete platforming puzzles to reach other planets and hit switches to unlock areas blocked by a force field.

Core Game Loop

Find a way to get off of each planet and propel towards new planets to reach the end of the solar system. Each planet will have it's own gravitational pull to keep the player grounded, and a platforming puzzle to solve before making progress. The last area has force fields around them that need to be deactivated by finding pressure plates scattered across the planet. Throughout the game, the player can use their propulsion tool to launch themselves high into the air, move around in open space, and destroy obstacles.

Core Mechanics

- GravityP,latforms
- Propulsion Tool

Supporting Mechanics

- Collectibles
- Propulsion Charging
- Propulsion aiming by turning the player around
- Pressure plates
- Obstacles (Force Fields)
- Smooth Gravity Transitioning
- Gravity Strength by Float Curve
- Oxygen system
- Destruction
- Launch pads (Jellyfish)

Risks

- Puzzle Complexity

- Game Length
- Gravity Movement polish
- Making navigating outside of space less confusing

MVP

A simple platformer game where you platform out of planets and towards new ones to get to the rare material at the end of the level!

Asset Style Guidelines

Unreal Asset Naming Rules

- Abbreviate the name of the parent class of an asset and include it in the name of the child asset as a Prefix or Postfix. Refer to Unreal Asset Naming Conventions.

Code Naming Rules

- Methods and Variables should be named with PascalCase (i.e. **FooBar()**)
- Events should be prefixed with On (i.e. **OnTakeDamage**)

Properties

- All properties meant to be accessed within Blueprint should be tagged with appropriate UPROPERTY() specifiers.
- Prefer concise, descriptive names to abbreviations or sentences (i.e. **ProjectileSpawnLocation** rather than **SpawnLoc** or **PlaceWhereTheProjectileIsSpawnedFrom**)

Conditionals

```
// Prefer conditionals to be formatted
if (condition) {}
// Rather than
if (condition == true) {}


// Prefer guard clauses to large nested if statements
if (!condition) return;
// Rather than
if (condition)
```

```
{
    // Large block of code
}


// Prefer long conditionals to be on multiple lines
if (condition1 ||
    condition2 ||
    condition3 ||
    etc) { }
```

C++ Classes and Structs

- Prefer structs to be used more for data storage and classes for functionality
- Prefer classes to always have a default, zero argument constructor (or constructor where all arguments have default values) in addition to any others such that all classes can be instantiated without passing arguments unless strictly necessary

Blueprints

- When making a blueprint first create a C++ class as a base, even if it'll be empty (In case it needs to be refactored for performance).
- In project settings blueprint tick by default will be turned off.
- Use redirect nodes to have property connections run around nodes.
- Redirect all underlapping property connection wires in all Blueprints.
- Straighten all property connection wires to be straight. (select two nodes and press SHIFT + Q to auto-align them if they are connected)

`

Commit Policy

- [Conventional Commits]
- All commit messages should be descriptive as if they were going directly into a changelog
- All commits should be prefixed with a noun describing the commit's purpose (i.e. **fix:**, **feat:**, **chore:**)
- Example:
    - **fix: Airship movement issue when hitting maximum allowed pitch**
    - **feat: Added auto projectile despawn based on distance from spawn location**
    - **chore: Forgot to save the scene file again**
    - **refactor: Converted BP_Cannon_Standard from Blueprint to C++**
    - **docs: Added documentation entry for StateMachine**