

Grim Harvest / Player

Architecture/Design Document

Table of Contents

1	Introduction..	3
2	Design Goals..	3
3	System Behavior..	3
4	Logical View...	4
4.1	High-Level Design (Architecture)	4
4.2	Mid-Level Design.	5
4.3	Detailed Class Design.	6
5	Process View...	7
6	Use Case View...	9

Change History

Version: 0.1

Modifier: Austin Morris

Date: 01/31/23

Description of Change: Module Design Document started.

Version: 0.2

Modifier: Austin Morris

Date: 02/24/23

Description of Change: Mid Level Design & High Level Design completed.

Version: 1.0

Modifier: Austin Morris

Date: 03/15/23

Description of Change: UML Diagram & Process View completed.

Version: 1.1

Modifier: Austin Morris

Date: 04/12/23

Description of Change: Changes made to account for alpha 2 submission.

1 Introduction

This document describes the architecture and design for Grim Harvest, a game being developed by Jest Games. Grim Harvest is a singleplayer gothic action roguelike which focuses on the use of strategy with well-timed I-framed dashes, and proper ability usage to defeat constant waves of enemies.

The purpose of this document is to describe the architecture and design of the Player Module application in a way that addresses the interests and concerns of all major stakeholders. For this application the major stakeholders are:

- Developers – they want an architecture that will minimize complexity and development effort.
- Project Manager – the project manager is responsible for assigning tasks and coordinating development work. He or she wants an architecture that divides the system into components of roughly equal size and complexity that can be developed simultaneously with minimal dependencies. For this to happen, the modules need well-defined interfaces. Also, because most individuals specialize in a particular skill or technology, modules should be designed around specific expertise. For example, all UI logic might be encapsulated in one module. Another might have all game logic.
- Maintenance Programmers – they want assurance that the system will be easy to evolve and maintain on into the future.

2 Design Goals

The design priorities for the Player system are:

- The design should minimize complexity and development effort.
- The design should allow Designers to easily have access to the properties of the weapon, player, and player movement.

3 System Behavior

The Player Module is built from the ACharacter base. The player has 3 main components for main gameplay functionality, health, damage, and death. These components allow the player to deal damage, receive damage, and die. The player should be able to use abilities based on the weapon currently equipped, which is handled by the Weapon Manager. The player should be able to dash in a directional input.

4 Logical View

The logical view describes the main functional components of the system. This includes modules, the static relationships between modules, and their dynamic patterns of interaction.

In this section the modules of the system are first expressed in terms of high level components (architecture) and progressively refined into more detailed components and eventually classes with specific attributes and operations.

4.1 High-Level Design (Architecture of the Entire system)

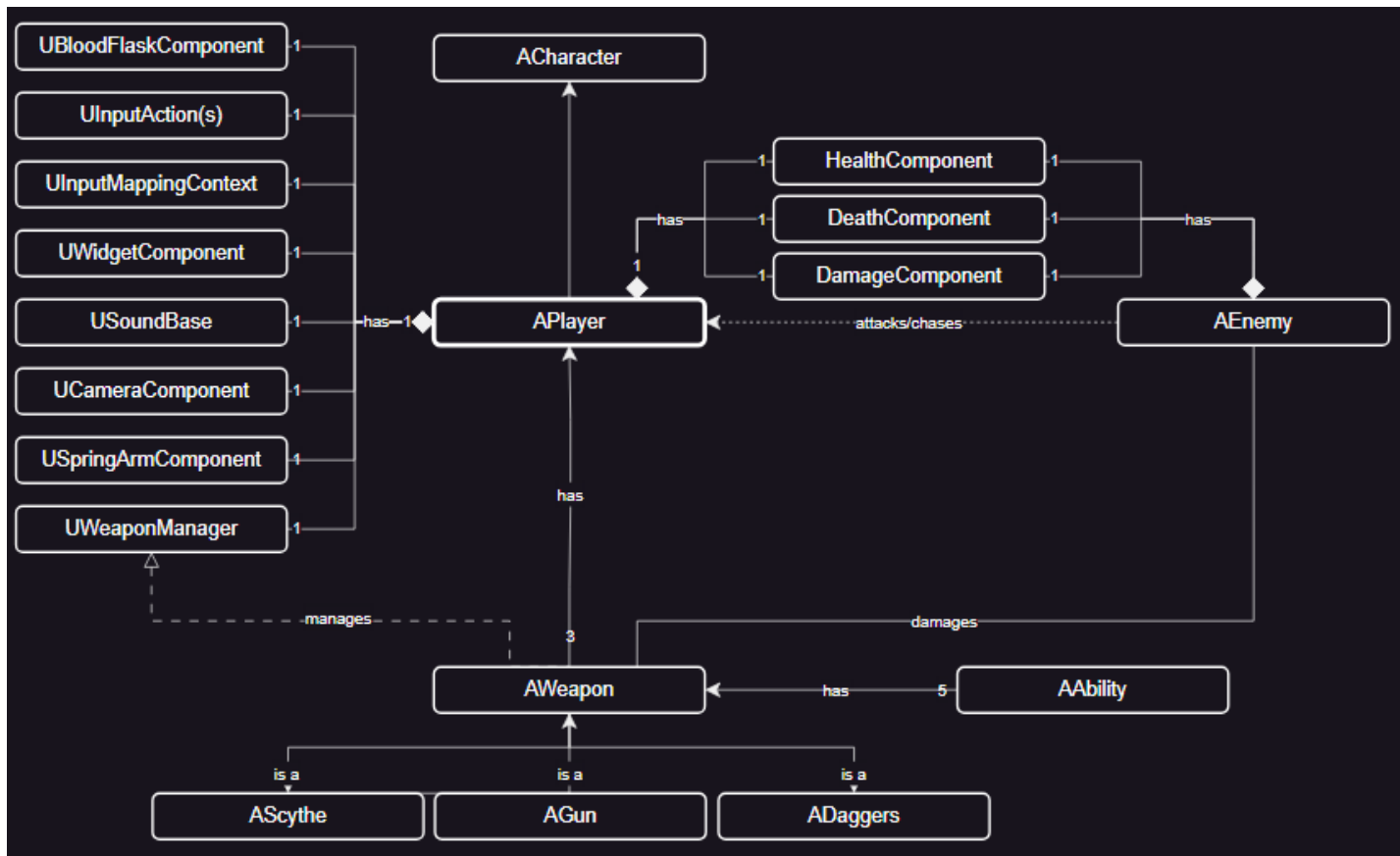
The high-level view consists of 4 major components, with 3 assisting sub components.



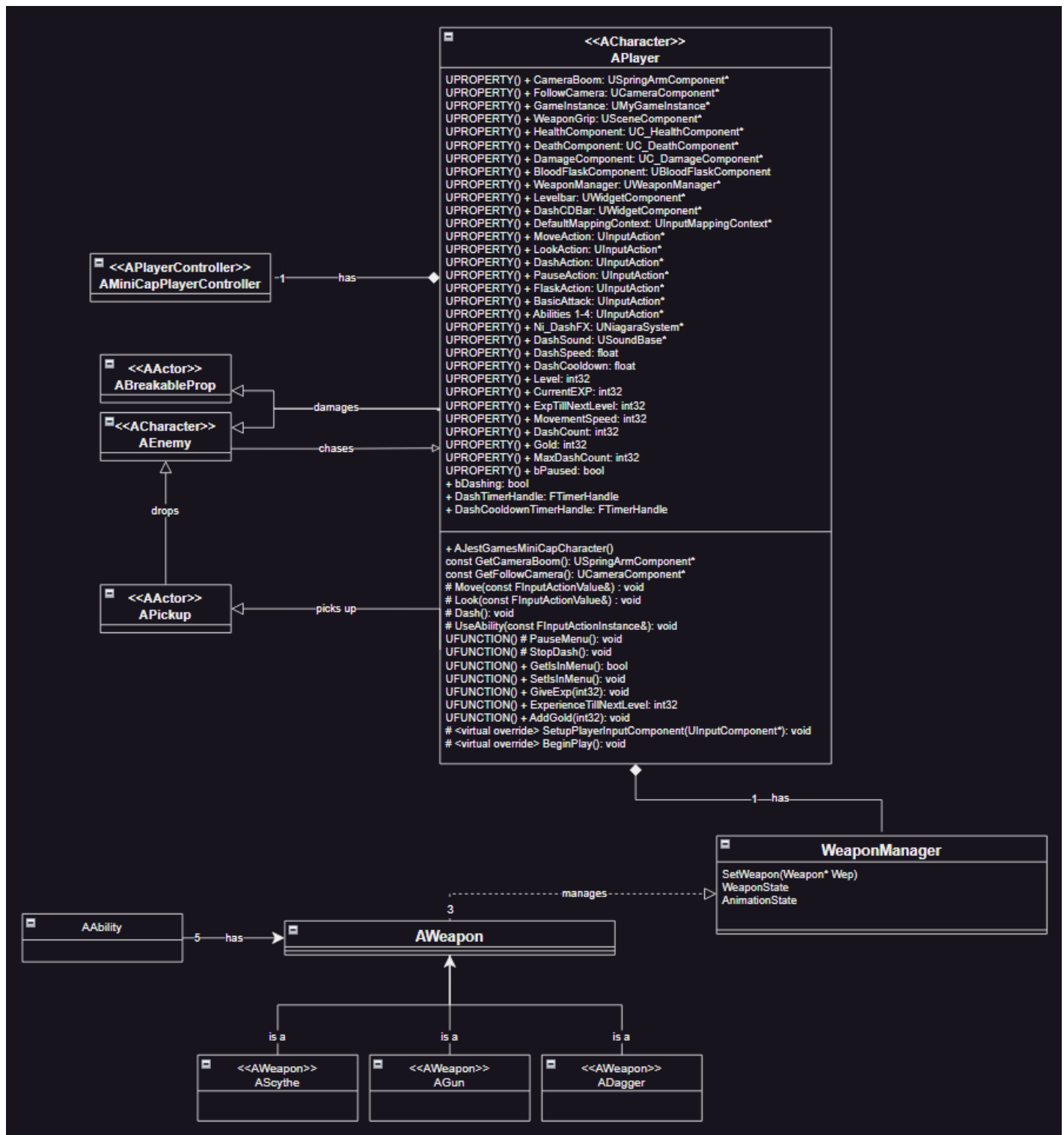
- Player System is the main system which has a character that takes input.
- Enemy System consists of multiple enemy types being spawned by an enemy spawn controller.
- AI system is used for enemy swarming & attacking behavior.
- Weapons System consist of 3 weapons that are handled by a weapon manager given to the player. Weapons can use abilities to damage enemies.
- Abilities handles the logic for all abilities.
- Upgrades is the upgrade system used to effect the player and abilities stats and variables using the GameInstance.
- Blood Moon system is the mechanic that occurs during gameplay which effects enemies for a short duration.

- The stats components consist of Health, Damage, and Death, and are given to the Enemy & Player class which handles all logic regarding the general combat.

4.2 Mid-Level Design of the Player Module



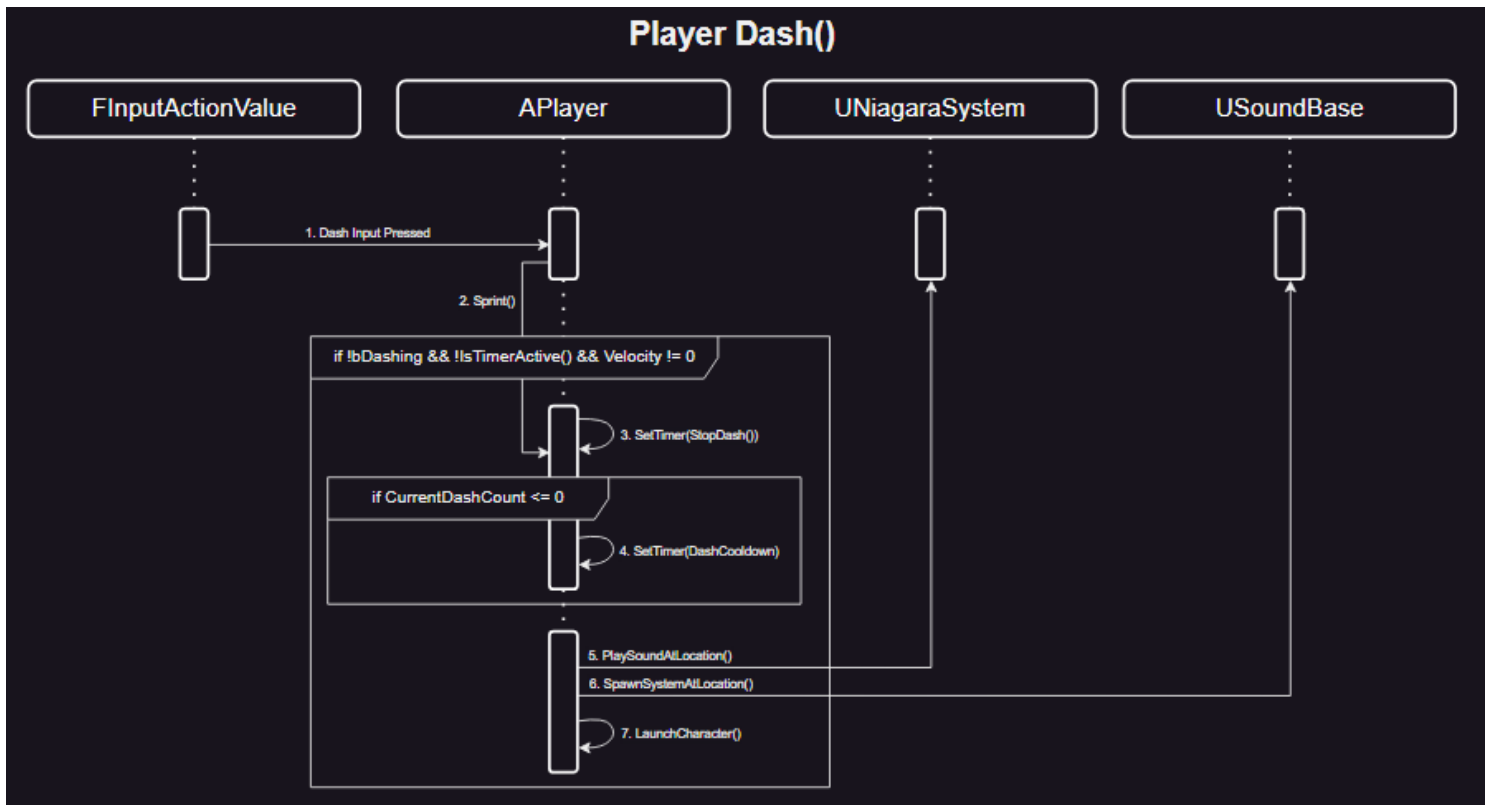
4.3 Detailed Class Design of the Player Module



5 Process View of the Player Module

The player is able to perform a few actions that can assist them during gameplay, including dashing, using abilities, and (in the future) using flasks. The Weapon module will most likely go into depth on the use of abilities, however there will be a small example here of how player input interacts with the abilities.

Player Dash



The dash sequence begins when the player presses the dash key (L-Shift on keyboard or L-Shoulder on controller). There are several checks the player must pass before using a dash. 1. The player can't be currently dashing. 2. The player's velocity cannot be 0, and 3. The player's dash cooldown timer cannot be active.

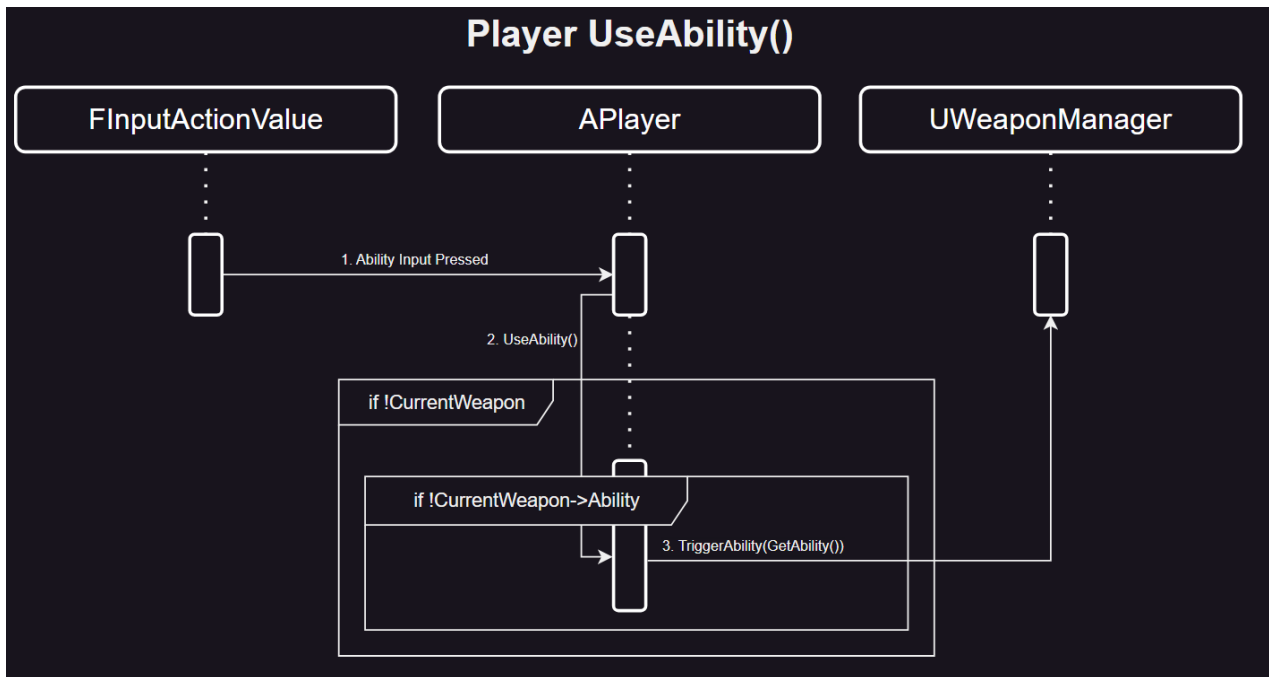
Once the player passes these checks, the `bDashing` boolean is set to true, `CurrentDashCount` is subtracted by 1, and the `StopDash()` timer is called, which stops the player's momentum based on a timer.

After that, there is another check to see if the `CurrentDashCount` is `<= 0`, which means the player is out of dashes. A dash cooldown timer is set, and it resets the `CurrentDashCount` to the `MaxDashCount`.

Finally, the effects are played, and the character is launched.

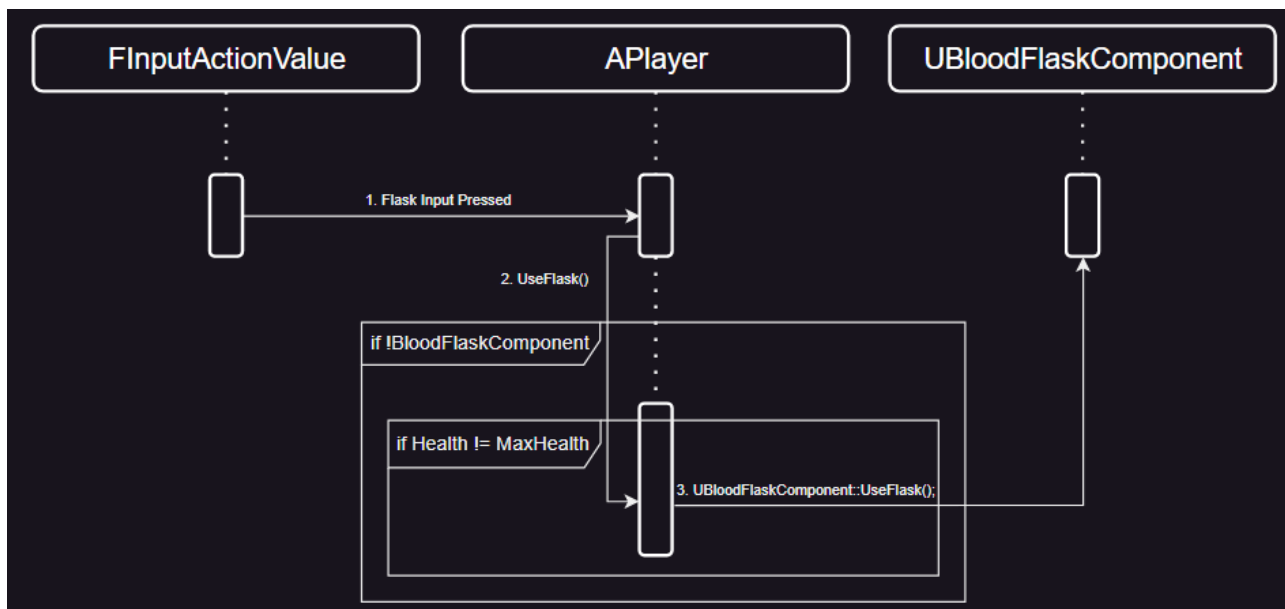
Player UseAbility

The player is able to use abilities, which change depending on the weapon currently selected. TriggerAbility uses a system which condenses this code into one statement. GetAbility gets a number for the Ability Array based on the name of the Ability used which you get from the Input Action Instance. The ability triggered will always be the right one based on input. (See Weapon Module for more information on abilities)



Player UseFlask

The player can use a flask using F to heal. This function uses the Blood Flask Component which is attached to the player. First, there's a null check, and then a simple check if the player's health isn't currently full. As long as it isn't full, the flask can be used.



6 Use Case View

Dashing









There are a few variables which can be edited to allow for more unique dashing.



The player can currently upgrade their dash count (default 10 gold for every first upgrade) by 1. This allows the player to dash twice before the cooldown triggering.

▼ Dash	
Dash Speed	1000.0
Dash Cooldown	1.0
Current Dash Count	1
Max Dash Count	1

There are several variables which can be edited in the editor, including DashSpeed, DashCooldown, CurrentDashCount, and MaxDashCount. Changing these will directly change the amount of times you can upgrade your dash, since the menu uses the MaxDashCount variable to determine how many upgrades you can purchase.

▼ Effects	
Ni Dash FX	 E   
Dash Sound	 Floating_UI_Close   

There are also a sound effect and Niagara particle effect which can be easily changed for the dash.

Player Stats

There are several stats that directly affect the player, which include Movement Speed, Dash Count, and Gold Gain (NOTE: Increased Gold Gain is not currently implemented in this version)



These variables are affected by the GameInstance versions, to make sure that player upgraded stats are kept throughout multiple menus/level loads.

There are several other stats which can be purchased, but they directly affect abilities so refer to Weapon Module doc for more information.