Software Requirements Specification

for

3D-HnK
(3D Hack n Slash)

Version 1.0 approved

Prepared by C. Hodge

TCPHijinks

2020-01-28

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Initial Draft	2020-01-28	Initial draft release.	v.1.0

1. Introduction

1.1 Purpose

This Software Requirement Specification (SRS) describes the functional and nonfunctional requirements for the "Ranma Project's" 3D Hack and Slash combat system subsystem. Including the accompanying movement system that facilitates the Hack & Slash system itself.

This is the first draft revision and is therefore subject to change.

1.2 Document Conventions

There are no document conventions currently.

1.3 Intended Audience and Reading Suggestions

The intended audience of this SRS is the developer of this specific system to provide clarity as to the exact required system functionality from a high-level perspective. The secondary intended audience is GitHub.com users with an interest in this specific release.

This document should be read sequentially in-order if the reader is not familiar with the SRS and the described requirements. Developers of this release should be aware that this SRS is subject to change. Therefore, Developers responsible for this release should take note of changes in the revision log. Reviewing these changes and read any made changes to avoid misunderstanding the intended system.

1.4 Project Scope

The purpose of the Hack n Slash system and accompanying movement system is the facilitation of a basic gameplay prototype. This basic gameplay prototype will demonstrate to interested parties the intended style of gameplay and the credibility of the project by providing a tangible prototype. Therefore, while the scope of this subsystem is small. It is the foundation of many future subsystems and the core component of gameplay. So it's imperative that this release be developed to a high standard; as this system is intended to be used in a complete video game system.

1.5 References

There are no references at this time.

2. Overall Description

2.1 Product Perspective

This Hack n Slash system is a part of a much larger video game system. The sum of this project and other following subsystems will be a completed 3D Hack n Slash RPG Video Game. This video game will include this combat system, movement, advanced animation system, useable items, useable gear, an advanced node dialog system, a node-based quest subsystem, save/load functionality, a music system, 3D graphics, general UI and more to be decided on.

Therefore it is essential that this product release be completely modular and optimized for use in a much larger interconnected system which will be composed of several advanced components.

2.2 Product Features

This product contains two major components and some accompanying smaller complementary features to facilitate testing.

- *Input system for controller gamepad input solution.*
- *8-Directional 3D movement system.*
- *An effector (i.e. weapon) and effect (e.g. do X damage to Y) system.*
- *An Character component health system that destroys the parent Character on "death".*
- *Combat system where a Character X can input an attack, damage and "kill" Character Y.*
- *A basic camera renderer (feature of Unity3D).*
- *A basic test scene.*

2.3 User Classes and Characteristics

Users of this system should be able to move in all 8 directions (N, S, E, W, NE, NW, SE, SW) using a video game gamepad input solution. Users should also be able to attack "Enemy" Characters by being within attack range, looking at the enemy and pressing an attack input button on the gamepad input solution.

Users with access to the Unity3D inspector and editor should be able to change the "Weapon" effector to another with another effect of any type by editing a special Character component that exposes modifiable properties to an effect subclass
(i.e. effector fists with 5 damage effect to an enemy hit).

2.4 Operating Environment

The operating environment includes:

- *Unity3D game engine environment for development.*
- *Unity3D built executable for testing.*
- *Operating System: Windows 10.*

2.5 Design and Implementation Constraints

- *The system must be built with performance in mind for less powerful computer hardware.*
- *The system must be very modular and built with the component design-pattern in mind.*
- *The system must be built in Unity3D & therefore is subject to engine limitations.*
- *The system must be designed with K.I.S.S and D.R.Y conventions in mind.*

2.6 User Documentation

There will be no user documentation delivered alongside this deliverable release. The only documentation will exist in the form of the systems themselves and accompanying comments and method summaries for the developer(s). As this is only the first subsystem for the game, no documentation will be provided for game testers either as the system should be intuitive.

2.7 Assumptions and Dependencies

This SRS is written under the assumption that Unity3D will be used and that all systems are designed and developed with modularity and best-practice in mind. Thereby enabling the rest of the subsequent game subsystems to utilize this release with minimum (if any) refactoring.

3. System Features

The "Player" character will have the ability to move in the 8 cardinal directions in a 3D test environment. The direction and speed moved will be determined from player input using any gamepad input device. The character movement will be physics-driven. The basic camera script will follow the player from above and at a moderate distance within the 3D test environment. The player will be able to attack, damage and destroy "Enemy" characters in the 3D environment. The system will be designed so any number of custom effects and effectors (i.e. weapons) can be created.

3.1 Player Gamepad Input Detection & Interpretation.

3.1.1 Description and Priority

Detect if the player is using a connected gamepad. – Must have
Read gamepad raw input. – Must have
Interpret specific raw input as game commands. – Must have.

This is a high priority item that is the most fundamental aspect of a video game system.

3.1.2 Stimulus/Response Sequences

Moving the gamepads left-thumbstick in a cardinal direction will be interpreted as a direction in the 3D video game environment. Pressing any of the standard four action buttons will be interpreted as an action.

3.1.3 Functional Requirements

- *Must use Unity 2019's new experimental input handling solution.*
- *Must correctly read input from the left-thumbstick.*
- *Left thumbstick must have a dead zone (min & max possible readable values).*
- *Must correctly read the four standard gamepad action buttons.*

3.2 8-Directional Movement

The system is reliant on 3.1 Input Detection & interpretation (for player testing). Using the input as, the character can move in the 8 cardinal directions (N, S, E, W, NE, NW, SE, SW). Future NPC (non-player-character) subsystems will generate artificial Vector2 thumbstick inputs to move with this system.

3.1.1 Description and Priority

Must be able to move in 8 directions.
Must rotate character to look in direction moving to.
The magnitude of Vector2 move direction must help dictate movement speed.
Must use Unity's Rigidbody system to enable physics-driven movement.
The system of movement must be universal for all characters. Not just the player.
Should be no transition when rotating.

3.1.2 Stimulus/Response Sequences

Given a Vector2 directional input (X, Y). The movement system will convert it to a Vector3 where Y becomes the Z-axis movement. The Y of the Vector3 should always be 0. Using this Vector3 the character will calculate the magnitude as the movement speed to be applied to the direction Vector3. The Vector3 will be then converted to the nearest integers away from zero so movement direction & rotation is always an exact cardinal direction. For instance, Vector3(.2,0,-.8) becomes Vector3(1,0-1).

3.1.3 Functional Requirements

- *Must move in an exact cardinal direction.*
- *The magnitude of the given direction Vector2 must help determine speed.*
- *Must be a physics-driven movement.*

3.3 Character Health System.

3.1.1 Description and Priority

Must have a maximum health capacity.
Must destroy character if health becomes less or equal to 0.
Health must NOT be mutated directly. Must be mutated via a method.
Current remaining health must be exposed to Read.

3.1.2 Stimulus/Response Sequences

Health will start by default at the maximum. Classes calling the damage method will pass an amount, the Health System will subtract that and check if it needs to destroy itself and the character.

3.1.3 Functional Requirements

- *Start at full (maximum) health.*
- *It must be completely self-sufficient and modular.*
- *Must destroy parent character when health below or equal to 0.*
- *It must mutate health using a method/*
- *It must expose the remaining health to Read.*

3.4 Character Modifier Properties Component.

3.1.1 Description and Priority

Characters must have a component with modifiers to affect character.
Character modifier component must allow exposure of the health system.
Character modifier component must expose the character animator system.

3.1.2 Stimulus/Response Sequences

The contained modifiers will be accessed by the character superclass and subclasses as modifiers when performing actions. For instance, modifying a speed percentile modifier to -.5 would incur a -50% speed penalty when the character is trying to move.

3.1.3 Functional Requirements

- *Character properties to facilitate effects*
- *Must not access the character itself. Character must access it.*
- *Must expose the health system.*
- *Must expose a primitive animation system.*

3.5 Effector and Effects System.

These systems are reliant on the existence of 3.4's Character Modifiable Properties component and subsequently 3.3's Character Health System.

3.1.1 Description and Priority

Must have an abstract Effector superclass to be extended by effectors.
Effectors must be able to easily have multiple effects.
Characters must have a component with modifiers that effects can affect.
Must have an abstract Effect superclass to be extended by effects.
Effects should do one simple thing. Effectors use many to do complex things.
Effects must-do changes to characters using their modifier component.

3.1.2 Stimulus/Response Sequences

Effector subclasses will be responsible for determining "How" the target Character is selected and "When" to apply all effects to the target's Character Modifier properties. For instance, a weapon effector that deals X damage to "Enemies" inside of an attack Trigger when an attack animation is playing.

3.1.3 Functional Requirements

- *A working effects system.*
- *A working effector system.*

3.6 Player Camera Follow.

3.1.1 Description and Priority

Using the player position, the camera always follows it. The position and rotation can be changed using offset variables. The camera otherwise will not rotate in this primitive iteration.

3.1.2 Stimulus/Response Sequences

The camera takes the player transform at the end of each frame, calculates where it must move to given the cameras offset variables and then moves there with some basic smoothing.

3.1.3 Functional Requirements

- *The camera must follow the given gameobject.*
- *The camera must have settings to offset position and rotation.*
- *The camera must have basic smoothing.*
- *The camera must update its position at the end of the frame.*

4. External Interface Requirements

4.1 User Interfaces

There is no UI in this release beyond what is rendered by the game engine. Any messages to the tester during testing will be delivered through the Unity3D editor's debugger.

4.2 Hardware Interfaces

A computer running the Windows operating system (Windows 10 preferred) is needed. A gamepad controller is also needed as an input device.

4.3 Software Interfaces

There are no known software interfaces relevant to what is being developed beyond the existence of the Unity3D engine itself and a basic acknowledgment of how it works independently to run the application behind the scenes.

4.4 Communications Interfaces

There are no communication interfaces in use.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

There are no performance requirements beyond using best practice, design-patterns and to be mindful of performance at this stage. Performance optimization will happen during a later release.

5.2 Safety Requirements

No safety requirements.

5.3 Security Requirements

No security requirements.

5.4 Software Quality Attributes

Software should be simple to read and follow the D.R.Y (Don't Repeat Yourself) principle. Ensuring that all systems are easy to understand, concise and very modular. Software design patterns should be used wherever possible to better ensure a product of a satisfactory quality.

6. Other Requirements

None.

Appendix A: Glossary

SRS *– Software Requirement Specification*
D.R.Y *– Don't repeat yourself.*
K.I.S.S *– Keep it simple stupid.*
Design Patterns *– the best solution to common design problems (i.e. components, singleton, etc.)*
Best Practice – General guidelines for generating good and legible code.

Appendix B: Analysis Models

None.

Appendix C: Issues List

Nothing at this time.