Technical Design Document – Outline

Theme Tussle

Jake Soucy

2024-04-14

Jakennik01@gmail.com

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author(s) | Changes |
| 0.1 | April 13th, 2024 | Jake Soucy | Created and Technical Design Title, History and Table of Contents |
| 0.2 | April 14th, 2024 | Jake Soucy | I finished all the last content |

1. Table of Contents

[1 Document History 2](#_Toc163996343)

[2 Game Summary 3](#_Toc163996344)

[3 Development Environment 4](#_Toc163996345)

[3.1 Development Hardware 4](#_Toc163996346)

[3.2 Programming Languages 4](#_Toc163996347)

[3.3 Development Tools 4](#_Toc163996348)

[3.4 Game Engine 4](#_Toc163996349)

[4 Architectural Analysis 4](#_Toc163996350)

[4.1 Classes 4](#_Toc163996351)

[4.2 Behavioral Analysis 4](#_Toc163996352)

[4.3 Game Loop 5](#_Toc163996353)

[5 Technical Risks 6](#_Toc163996354)

# Game Summary

In this 2D fighting game, the user controls one of four characters they’ve selected to combat against another player or a computer. The goal is to have the opposing character’s health points reach zero by utilizing an array of moves against them. The game is fast-paced and for true fighting game fans.

# Development Environment

## Development Hardware

A GF63 Thin 11UD Laptop (Windows 11) was used to create this project.

## Programming Languages

C++ was used for the engine and scene design, and JSON was used for the animations.

## Development Tools

Visual Studio 2022 was used to make the project whole, Paint.NET was used to create the sprites, and Splice was used to generate music for the game.

## Game Engine

This project was done using the C++ game engine made by Dave Burchill at NBCC. It contains an entity manager, physics, components and anything else necessary to make a game. I didn’t make any significant changes except create new components, which we will see in the future.

# Architectural Analysis

## Classes

|  |  |  |
| --- | --- | --- |
| Class | Responsibilities | Collaborations |
| Scene Menu | Giving the user the option to navigate through all the following scenes | Scene Character Select, Scene Settings, Scene Arcade, Scene Tutorial |
| Scene Settings | It gives the player an option to change the in-game volume | Scene Menu |
| Scene Character Select | It gives the player the ability to select their character | Scene Menu, Scene Tussle |
| Scene Tussle | The main game, where the users duke it out | Scene Win Screen, Scene Menu |
| Scene Arcade | Displays Arcade information | Scene Menu |
| Scene Tutorial | Displays Tutorial information | Scene Menu |
| Scene Win Screen | Displays the winner! |  |

## Behavioral Analysis

A character can move, attack and be attacked. The player's state will change depending on what attack they’re performing, their direction, or if they have been hit. This screenshot explains how exactly a hit is registered in Theme Tussle.A screenshot of a video game

Description automatically generated

## Game Loop

The following will be called after the user has entered the main game out of the main menu.

Theme\_Tussle.cpp

loadLevel(), loadCharacter(), loadEnemy and loadMode() are all called once and then never again; these read config files in the assets folder.

registerActions() is called once and then never again followed by spawnPlayer() and init().  
spawnPlayer() spawns players and registerActions() maps all buttons to movement and attack commands.  
After that, playerMovement() gets called by the sMovement() command. playerMovement() focuses on the two characters' movement, while sMovement() is used for a general movement of every entity and gravity.

sRender() draws all the entities and text on the screen for the player to see, and sDoAction() listens for player input and then passes a state to the player depending on the buttons pressed. If a movement button is pressed, movement will be performed in the playerMovement() function.

sCollisions() handles all collisions we saw earlier, and it also adjusts the player's position in case they try to escape the game’s window.

sUpdate() updates a couple of different tasks, such as if the player is on the right side, give him the CSide() “right.”

Finally, the checkEnemy() and checkPlayer() state functions call another function (statePlayerCheck(), stateEnemyCheck(), stateCheckNoHitBox()(if the state isn’t an attack.) Those functions aim to assign damage, animations, hitboxes, and hurt boxes for the player if the state is equal to the current character state.

# Technical Risks

List all technical risks that could make it difficult or impossible to complete the game. Examples:

|  |  |  |
| --- | --- | --- |
| Risk | Severity | Mitigation (what is to be done to eliminate or minimize this risk) |
| Unsure how to have attacks be slower than others | It makes the game a little clunky, but nothing terrible | As of now, the animation duration is the only thing that matters. |
| Uncertainty of how to implement a computer as of now | Arcade mode cannot be completed without it | Focused on multiplayer for now |