Software Requirements Specification

for

Street Fighter

Version 1.1 approved

Prepared by:

Isbah Malik 21L-1843
Maryam Saqib 21L-5164
Shahryar Ahmad 21L-7727
Usman Ali 21L-5405

FAST-NUCES, Lahore

3rd March 2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Street Fighter	3/8/2024	Initial Version	1.0
Street Fighter	5/5/2024	TBD 1 through 6 completed	1.1

1. Introduction

1.1 Purpose

This document details the Software Requirements Specifications for “Street Fighter” for FAST-NUCES, Lahore. Street Fighter is built purely for entertainment purposes to help people re-experience the thrill of their beloved childhood games without requiring emulators. The primary aim of the document is to detail the entire functionality of the game system.

1.2 Document Conventions

The font that is followed in this document is Arial and its size is 11. Special highlighting is done by making the text bold so that the important keywords can be easily differentiated. Every requirement stated in this document has its own unique priority and every functionality is equally important.

1.3 Intended Audience and Reading Suggestions

The document is meant for the evaluation team at FAST-NUCES who are to evaluate this project. It is also made for the people looking to play this game for enjoyment. They are the ones who will state the requirements of the software and their feedback will also be needed to modify the existing requirements

This document shall also serve as the reference document for the development team (Isbah, Maryam , Shahryar and Usman) who will Analyze, Design and Implement the system. They will coordinate every activity that will take place in the Software Engineering process and will be guided by the Evaluators (Ms. Mehroze and Mr.Usama) at FAST-NUCES.

1.4 Product Scope

The Street Fighter Game is a software project aimed at recreating the nostalgic experience of the classic Street Fighter console game for modern systems. This endeavor is driven by the recognition that many beloved games from our childhood face compatibility challenges on contemporary platforms, often necessitating the use of emulators that may detract from the overall enjoyment. In response to this, our project focuses on the development of an exact replica of the iconic Street Fighter game using cutting-edge technologies such as SFML (Simple and Fast Multimedia Library) and C++.

1.5 References

The document does not refer to any external files.

2. Overall Description

2.1 Product Perspective

While "Street Fighter" is a self-contained product, it integrates with the broader legacy of the "Street Fighter" game series. It acts as a bridge between the past and present, ensuring compatibility with modern computing environments. The SRS identifies key interfaces with the original game series, outlining the seamless connectivity that allows users to relive the classic gaming experience without compromising on performance or authenticity. Stakeholders for the "Street Fighter" project encompass both end-users and evaluating entities. The primary stakeholders are the players who seek to engage in the nostalgic gaming experience provided by the product.

2.2 Product Functions

The player is first shown a startup sequence which leads to a menu screen. From the menu screen he can close the game, go to character and stage select, open training mode or view the credits.

If the player opts to view the credits he is shown the names and information of the development team.

At the character and stage selection screen, the player can select which character he wants to play as, as well as the stage (and effectively the enemy) he wants to go up against.

After the selection stage, the player and his enemy are spawned and their timed battle begins.

The training stage allows the player to select a character. It is set for infinite against a dummy enemy to practice all moves of a character. The player can select to return from this screen to the menu screen anytime.

Non-Functional Features:

The system boots up at any engine which meets the minimum requirements* in under 30 seconds. Valid inputs to the system shall produce the resulting move on screen in under 2 seconds. The system should have high reliability and stability.

2.3 User Classes and Characteristics

Users of this system shall be able to play the game, train in-game and view the credits. All users are of the same type: Players. Players will have access to all player-class functions and the ability to select from different options. They should be able to perform the following tasks:

- Select a mode:
    - Normal Play
    - Training Mode
- Select a character (list expected to update by final evaluation)
    - Ryu
    - Zangief
    - Sagat

    - Dhalsim
    - Chun-Li
    - Balrog
    - Guile
    - Ken

- Select an enemy/stage (list expected to update by final evaluation)
    - Ryu
    - Zangief
    - Sagat
    - Dhalsim
    - Chun-Li
    - Balrog
    - Guile
    - Ken

- Perform the following categories of moves from the character's arsenal
    - Basic Movement:
        - Move Left/Right
        - Jump
        - Crouch
    - Basic Hits
        - Punch
        - Kick
    - Combos (if any)
    - Special Moves (if any)
- Practice infinitely in the training mode
- View credits
- Exit the game

Aside from Player Class, a game class handles interaction among in-game assets such as:

- Collision detection
- Round time management
- Setting stage and other environment assets
- Round win/lose management

2.4 Operating Environment

The game is developed as a shippable executable file along with its assets. The Operating system on the PCs/Laptops running this file can be Windows, Linux, Unix or Mac.

The system will be built with the help of the following software:

Sr No.	**System**	**Environment for Development**	**Description**
1	Software used to Develop	C++	C++ (or "C-plus-plus") is a generic programming language for building

			software. It's an object-oriented language. In other words, it emphasizes using data fields with unique attributes (a.k.a. objects) rather than logic or functions.
2		SFML (Simple and Fast Multimedia Library)	Simple and Fast Multimedia Library (SFML) is a cross-platform software development library designed to provide a simple application programming interface (API) to various multimedia components in computers.
3		IDE: Cross Platform	Runs across all supporting compilers having SFML installed.
4		Version Control: Git, GitHub	GIT is a version-control system for tracking changes in computer-files and coordinating work on those files among multiple people.

2.5 Design and Implementation Constraints

1) The use of SFML and C++ is mandated for the development, limiting the development team to these specific technologies.
2) Compatibility with modern systems is a priority, imposing constraints on the use of legacy technologies that might hinder the game's performance on newer platforms.
3) The game should be optimized to work on a wide range of modern hardware configurations, considering factors such as processor speeds, graphics capabilities, and memory constraints.
4) Ensuring that the game runs smoothly without any performance issues on the targeted hardware is crucial.
5) Compatibility with keyboards and screen resolutions must be addressed to cater to a diverse user base.
6) Adherence to legal and regulatory frameworks, especially those related to intellectual property rights and licensing agreements, is crucial.
7) Compliance with any gaming industry standards or regulations applicable to the development and distribution of video games.
8) Adhering to coding standards, design conventions, and best practices to ensure maintainability and scalability of the codebase.
9) Utilizing high-quality sound assets that meet copyright regulations and licensing requirements.
10) Ensuring that the sound design integrates well with the gameplay and contributes to the nostalgic yet modern feel of the game.
11) Adapting the user interface, text, and audio to different languages and cultural contexts.

12) Developing an effective training mode involves addressing user interactions and creating a system that adapts to different skills. This requires additional design considerations and potential constraints.
13) Implementing a robust testing strategy to ensure the game functions as intended on various platforms and configurations.
14) Considering constraints related to testing timeframes, resources, and the availability of testing environments.

2.6 User Documentation

The following document shall be available for end users:

- User Manual

The User Manual shall explain all key bindings and possible moves for all characters as well as their damage.

2.7 Assumptions and Dependencies

1) It is assumed that the primary focus of the game is on a single-player mode, allowing one player to engage in battles against computer-controlled opponents.
2) Assuming that players will predominantly use standard input devices such as keyboards for playing the game. The design does not account for more specialized input devices that might be used by some players.
3) Assuming that the iconic characters from Street Fighter II are available for use in the game.
4) Assuming that players are familiar with the basic gameplay mechanics and conventions associated with traditional fighting games, particularly Street Fighter II.
5) The project is dependent on the compatibility and proper functioning of SFML and C++ for graphics, game logic, and overall game development. Any changes or updates to these technologies might impact the project.
6) Dependency on the availability of character animation resources that accurately replicate the iconic moves and combos from Street Fighter II. Collaboration with animation libraries may be necessary.
7) Dependency on the effective design and implementation of the user interface, ensuring clarity in presenting game information, character status, and controls.
8) The project is dependent on thorough quality assurance and testing processes to identify and rectify any bugs, glitches, or performance issues that may arise during development.
9) Dependency on the availability of player feedback and the ability to incorporate it into the development process to improve gameplay, address concerns, and enhance overall user satisfaction.

3. External Interface Requirements

3.1 User Interfaces

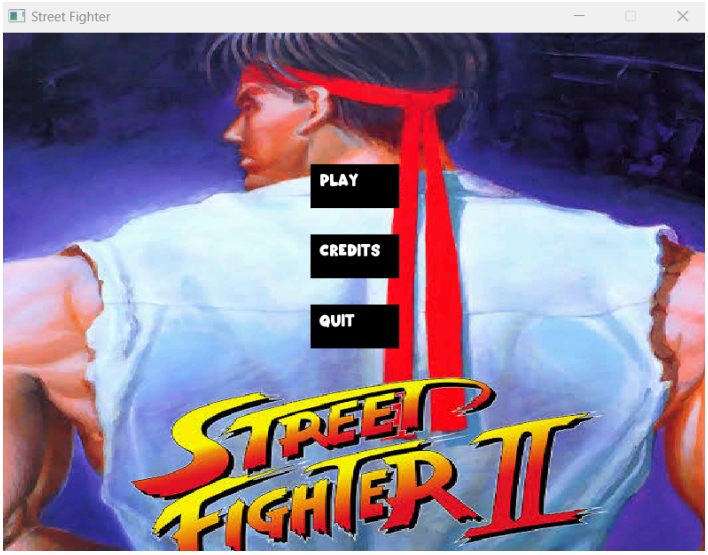
The User Interface section defines the way the various stakeholders interact with the System. All

screens will be developed to work on a PC/Laptop. Buttons will be used to make the navigation simpler.

3.1.1 Startup Screen

Plays intro video and music and displays options. (WIP)

- Allows user to start game by pressing a key
- Play Button: Allows user to play the game
- Credits Button: Allows user to navigate to credits screen
- Quit Button: Exits from the game



3.1.2 Stage Selection Menu

Allows the user to choose a stage to play from a menu of multiple stages.

3.1.3 Regular Fight Mode Screen

Displays characters, their moves and the health bar status.

- Use different keyboard inputs to execute moves
- Health bar status will change when a move results in collision

3.1.4 Training Mode Screen

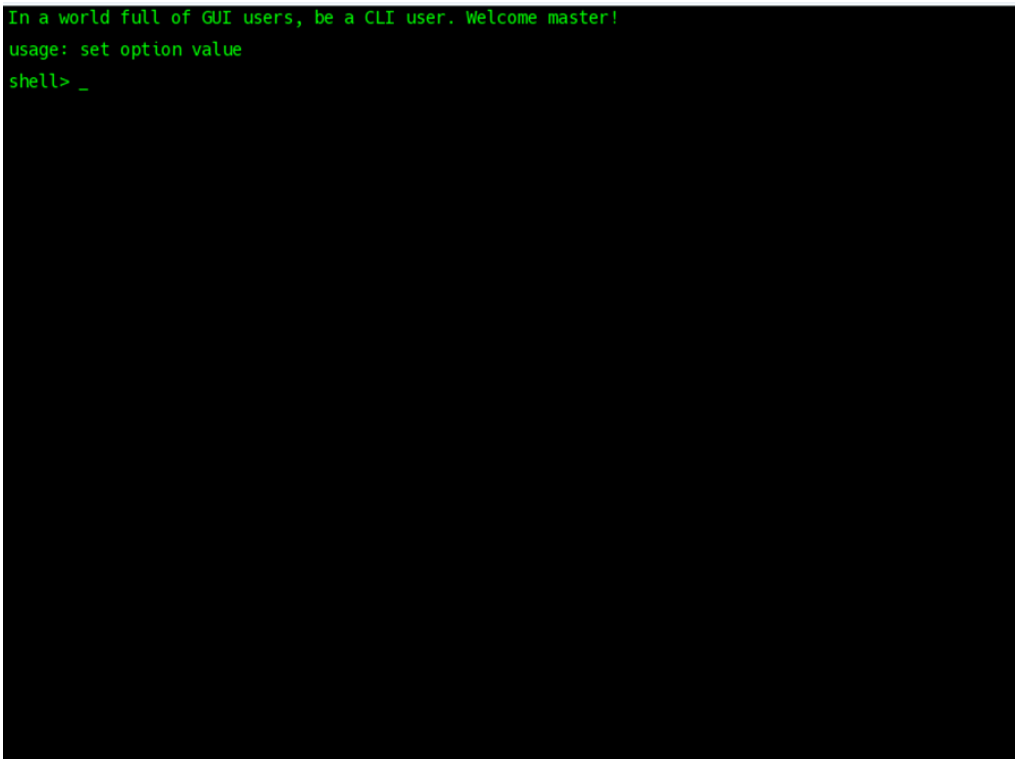
Displays characters and only the user's character executes movements.

3.1.5 Credits Screen

Displays Credits.

3.1.6 Settings Terminal

Users can change volume, fps and enable or disable training mode here.



3.2 Hardware Interfaces

1) The game exclusively supports keyboard input for user interaction.
2) Recognizes keyboard input commands for character movement, attacks, and special moves.
3) Graphics output displayed on the user's monitor or screen.
4) Audio output delivered through the user's speakers or headphones.
5) Storage devices include hard drives, solid-state drives, or other storage media.
6) Interacts with the operating system for tasks such as file operations, window management, and keyboard input recognition.

3.3 Software Interfaces

1) Visual Studio (version 2019+): Developed using the IDE for coding, debugging, and building the project.
2) SFML (Simple and Fast Multimedia Library): Utilizing SFML for graphics rendering, providing smooth visuals, animations, and graphical elements.
3) Windows, Linux: Designed to run on multiple operating systems, involving interaction with the OS for file operations and window management.
4) Git: Utilizing Git for version control, enabling collaborative development, change tracking, and code version management.
5) Visual Studio Debugger: Leveraging the built-in debugger for identifying and resolving software bugs during development.
6) C++: Game coded in C++, utilizing its features and libraries for implementing game logic.
7) CMake (version 3.28+): Utilizing CMake for configuring the build process and generating platform-specific build files.

3.4 Communications Interfaces

No such interfaces required.

4. System Features

4.1 Character Select

4.1.1 Description and Priority

This feature allows the user to select the character he wants to play as.

Priority: High

4.1.2 Stimulus/Response Sequences

- The system presents a character select screen to the user.
- The user then selects his character from the screen

- The user presses confirm to lock in the character

4.1.3 Functional Requirements

REQ-1: The system shall present the character select screen, containing all playable characters, to the user to select a character
REQ-2: The system shall allow the user to select the character of his choice by clicking on the respective button
REQ-3: The system shall update the currently selected character as the user clicks on the selection button
REQ-4: The system shall allow the user to lock in their character by clicking the respective button in the character select screen

4.2 Stage Select

4.2.1 Description and Priority

This feature allows the user to select the stage he wants to play on. (and effectively the enemy he wants to play against)

Priority: High

4.2.2 Stimulus/Response Sequences

- The system presents a stage select screen to the user.
- The user then selects his stage from the screen
- The user presses confirm to lock in the stage

4.2.3 Functional Requirements

REQ-5: The system shall present the character select screen, containing all playable characters, to the user to select a character
REQ-6: The system shall allow the user to select the character of his choice by clicking on the respective button
REQ-7: The system shall update the currently selected character as the user clicks on the selection button
REQ-8: The system shall allow the user to lock in their character by clicking the respective button in the stage select screen

4.3 Player Movements

4.3.1 Description and Priority

This feature is responsible for handling player moves in the game.

4.3.2 Stimulus/Response Sequences

- This feature is available in normal play and training mode stages only.
- The system loads the player character on the screen
- The user can press any key/combination of keys.
- The system checks which of these keys (if any) invoke any change in state of the player character.
- The system updates the state of the player character and plays the corresponding animation.

4.3.3 Functional Requirements

REQ-9: The system shall detect the user input and produce its results on screen (if any) in <= 0.5 seconds.
REQ-10: The system shall correctly map the input key to change in state for the player and update the player character to the correct state.

4.4 Enemy Movements

4.4.1 Description and Priority

This feature provides the player an enemy to play against.

Priority: High

4.4.2 Stimulus/Response Sequences

- The system sets the stage and music and loads the enemy on screen.
- The system updates the enemy states to play against the player.

4.4.3 Functional Requirements

REQ-11: The system shall provide an automated opponent to the character in the normal play mode.
REQ-12: The system shall provide a stationary enemy with infinite health to players in training mode.

4.5 Collision Detection

4.5.1 Description and Priority

This feature detects collisions between player and enemy and updates their states and health-bars accordingly.

Priority: High

4.5.2 Stimulus/Response Sequences

- The system detects any collision between player and enemy frames.
- The system examines the states of player and enemy at collision.
- The system decides its next course of action depending on the results of the previous examination.
- The system updates the states of enemy and/or player
- The system updates the health-bar of enemy and/or player

4.5.3 Functional Requirements

REQ-13: The system shall correctly detect collision detection between player and enemy frames.
REQ-14: The system shall examine the states of enemy and player and update their states and health-bars in <= 2 seconds.

4.6 Credits

4.6.1 Description and Priority

This feature allows the players to view the credits for the game.

Priority: Low

4.6.2 Stimulus/Response Sequences

- The system presents a screen to the player with all the credits.
- The player can exit from the screen by pressing the back button.

4.6.3 Functional Requirements

REQ-15: The system shall present the credits screen to the player in <= 2 seconds of pressing the credits button on the main menu.

4.7 Exit

4.7.1 Description and Priority

This feature allows the user to exit the game.

Priority: Medium

4.7.2 Stimulus/Response Sequences

- The system closes the application(game) when the exit button is pressed.

4.7.3 Functional Requirements

REQ-16: The system shall close the game in <= 2 seconds of pressing the exit button.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The following processes are critical and must respond as per below

- Key input to player movement translation <= 0.5 seconds
- Player and enemy health updates <= 2 seconds
- Stage setup <= 10 seconds
- Loading character select screen <= 5 seconds
- Loading stage select screen <= 5 seconds

5.2 Safety Requirements

The user interface must be designed with clear, intuitive controls to prevent user confusion during gameplay. Avoid rapid, repetitive flashing or contrasting patterns in the game's visuals.Audio levels should be adjustable to prevent potential hearing damage or discomfort to users. Adhere to safety regulations and standards outlined by relevant gaming industry authorities.

5.3 Security Requirements

Ensure compliance with industry-specific security standards and regulations.

5.4 Software Quality Attributes

1) Usability: The game interface must be intuitive and user-friendly.
2) Reliability: The game must operate consistently without critical failures.
3) Performance: Ensure smooth gameplay with minimal lag and quick response times.
4) Maintainability: Codebase must be well-organized and documented to facilitate future updates and modifications.
5) Robustness: The game should gracefully handle unexpected inputs or situations without crashing.
6) Portability: Ensure the game runs seamlessly on major operating systems (Linux, Windows).

5.5 Business Rules

BR-1: Each player can choose from a list of characters before starting a match. A player cannot select the same character as the chosen enemy character in a single match.

BR-2: Only one player can play at a time.

BR-3: Each character has a set of special moves and combos unique to their fighting style. The execution of special moves and combos is based on specific input commands.

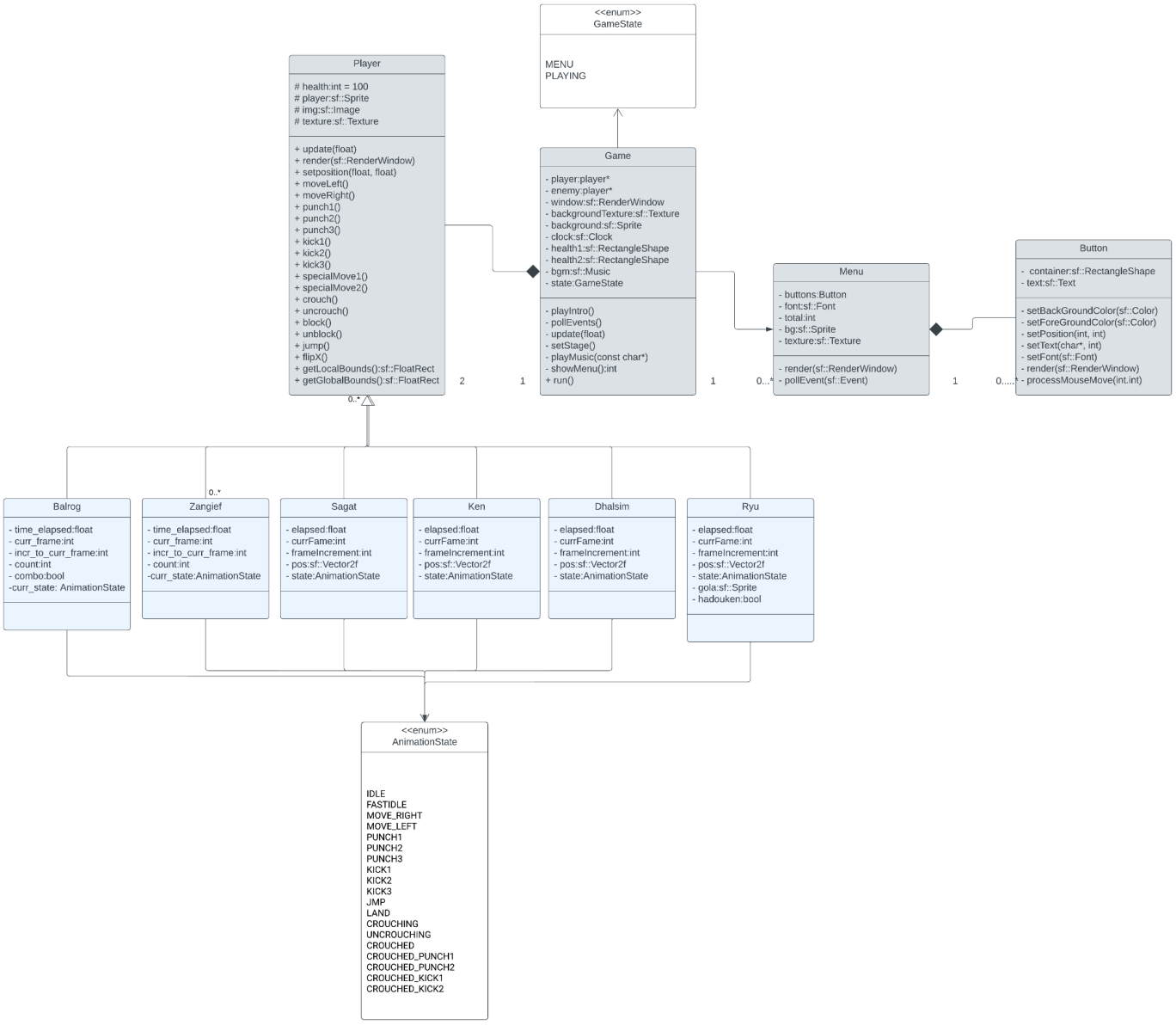
BR-4: A training mode is available for players to practice and improve their skills at their own pace. In training mode, players can freely experiment with character moves without time constraints.

Appendix A: Glossary

1. TBD: To be decided
2. WIP: Work In Progress
3. Collision Detection: determining whether two or more game objects intersect or collide with each other in the game world. This detection is crucial for implementing gameplay mechanics such as character interaction, object interaction, and environmental collisions.
4. Assets: Resources used in the development of a game, including but not limited to graphics, audio files, 3D models, animations, textures, and scripts. Assets contribute to the visual and auditory elements of the game and are essential for creating an immersive gaming experience.
5. OS: Operating System
6. IDE: Integrated Development Environment
7. CMake: Cross-platform Make

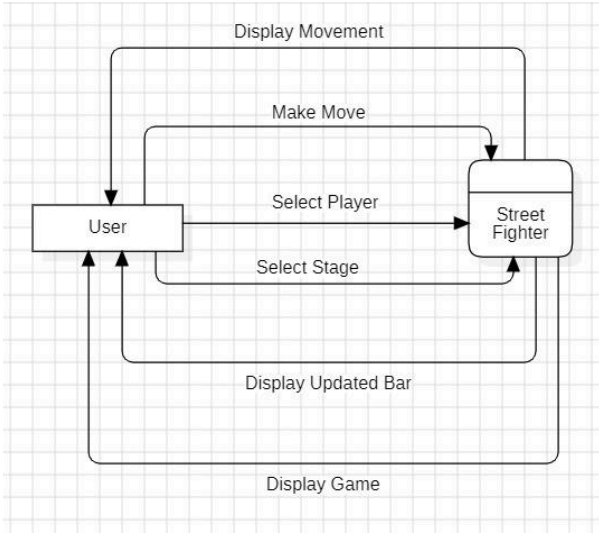
Appendix B: Analysis Models

Class Diagram

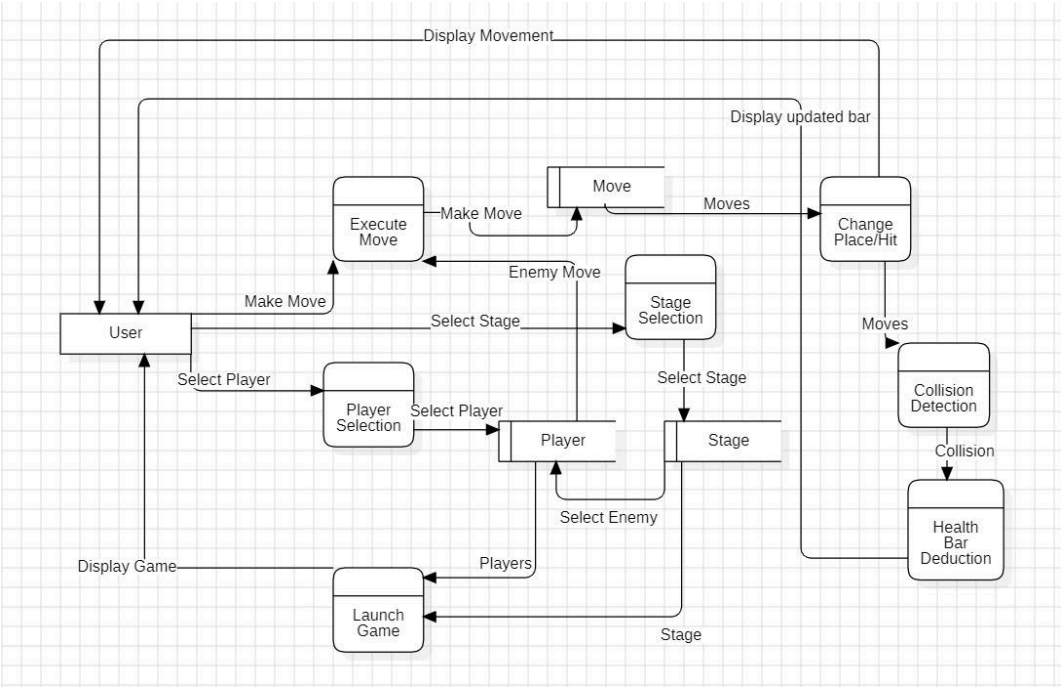


*subject to change till final evaluation

DFD Level 0



DFD Level 1



Appendix C: To Be Determined List

NaN - TBD List Completed