Product Name: RePoste

Client: Terry Yoo

Team Name: Goobernauts

Team Member Names:

Stephen Goodridge, Logan Geiser, David Geng, Ryan Giles, Heath Miller

Date: 11/4/2024

RePoste Fencing Replay System
System Design Document

Table of Contents

1. Introduction

This capstone project is for the development of a video replay software system for the sport of fencing. This project is for Dr. Terry Yoo, in partial fulfillment of the Computer Science BS degree for the University of Maine. Our team will also be working closely with the University of Maine's fencing club, the Blade Society, when developing this product.

The goal of this project is to create open-source video replay software that can run on a laptop using MacOS or Windows. The USA National Fencing requires a video replay system to help referees review calls during fencing matches. Currently the replay system used is outdated, expensive, closed-sourced and needs to be updated. This product aims to give a solution to clubs and competitions by offering an open-source solution that can increase accessibility to video replay systems.

1.1 Purpose of This Document

The purpose of this Software Design Document is to provide a guide for the design and implementation of the replay system. This document captures the system architecture, data design, and requirements. By detailing architectural decisions, module decomposition, and data handling, this SDD ensures that team members have consistent understanding and execution across development.

This document will serve as the primary reference throughout development, ensuring that the design aligns with requirements and stakeholder expectations. Additionally, the appendix outlines review and agreement protocols.

1.2 References

SRS:
https://docs.google.com/document/d/1ghOX66CNCSBitvlBNi53V00oroY5A7_cwFiM-EpjHns/edit?usp=sharing

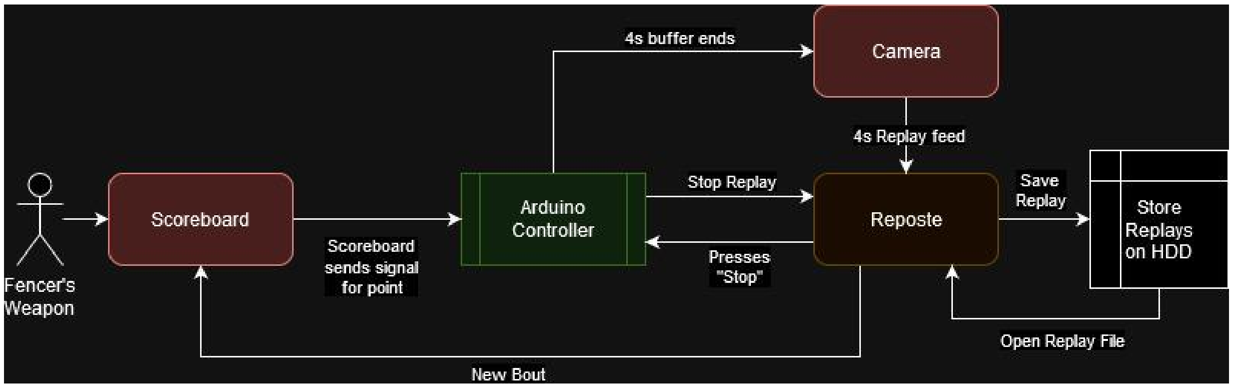
Knowledge Acquisition:
https://cmake.org/
https://cmake.org/documentation/
https://cmake.org/cmake/help/book/mastering-cmake/
https://github.com/marketplace/actions/cmake-quality
https://opencv.org/

Client Supplied:
https://superfencingsystem.com/
http://escrime.koovbroadcasting.com/download/RefereeVideoReplayUserManual.pdf
https://static.fie.org/uploads/3/18851-Handbook_of_specification_Video%20Refereeing.pdf
https://www.fencing.ab.ca/wp-content/uploads/2016/11/Video-Refereeing-Information-Sheet.pdf
https://superfencingsystem.com/SFS-Link_Manual_V1.1.pdf

2 System Architecture

This section covers the architectural design diagram, hardware, software utilization, and decomposition of the system. The architecture design chosen is the Event-Driven style as the system activates when a trigger occurs and saves the current buffer from the camera for further review.

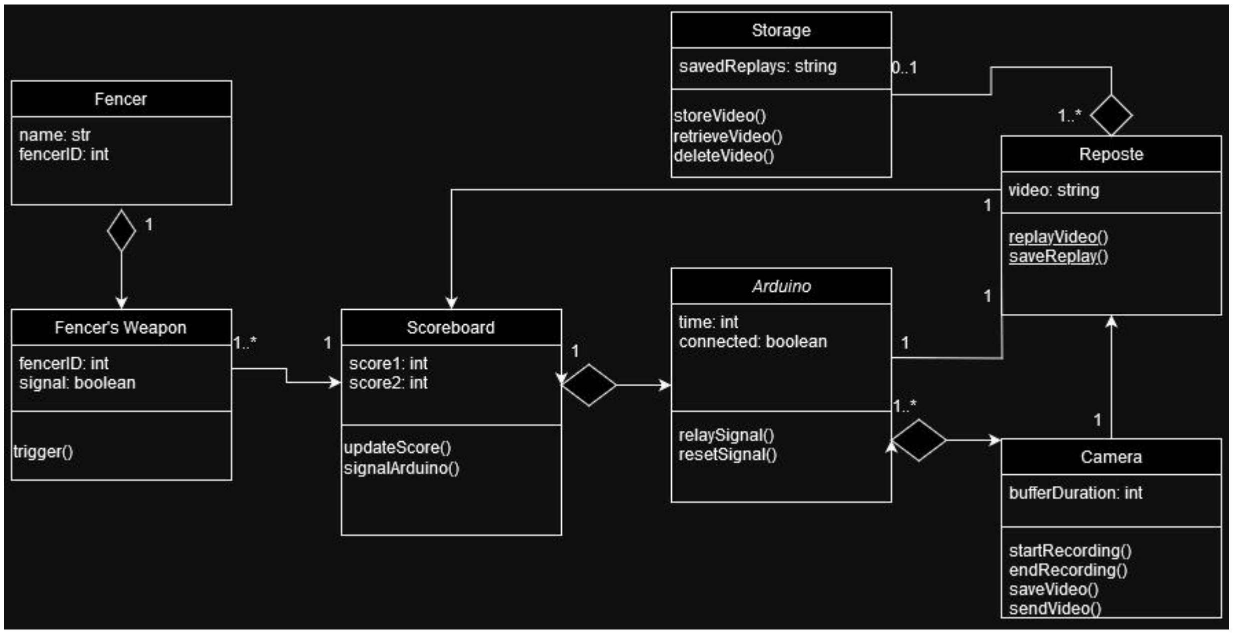
2.1 Architectural Design



The diagram above showcases an event-driven architecture. The triggering event is when the fencer weapon triggers the scoreboard, or when manually triggered through the GUI . When the event occurs, the scoreboard signals the Arduino controller, which sends a signal to both the camera and the system to stop recording and save the last four seconds in the buffer. The four second video is then visible for review in the GUI and then is able to be saved on a hard drive. The files saved on the hard drive are able to be reviewed at a later time through GUI.

Our system will utilize the laptop webcam and an Arduino that is built to work with the fencing scoring machine used by the Blade Society (Favero FA-07) to detect when fencers score a point. When the Arduino triggers or when manually activated, it will signal the system to save the recording from the webcam of the last four seconds where it can be manually reviewed. The file can be later retrieved from the harddrive to be reviewed in the GUI. For software, our project will utilize CMake, Visual Studio Code, and OpenCV for development, ensuring its usability across operating systems, and webcam usage.

2.2 Decomposition Description



The diagram shows the object-oriented decomposition of our Reposte replay system. This diagram focuses on how each component interacts in order to manage the video playback system. The diagram shows an event-driven architecture where actions like points being scored trigger the rest of the components, making sure that the referee will be able to review. Reposte in this diagram acts as a controller that manages the user interactions and provides an interface. While the other devices like the camera provide a model by capturing the footage.

Each one of the components is representing a class. Each has their own attributes and methods that the class will use. Like for example, the Fencer and Fencing Sabre class both are there to represent the physical inputs that will trigger the rest of the system. The scoreboard receives this signal first, once receiving the signal it then will send a message to the arduino which is used to trigger our camera. Once the camera receives that signal, it begins to record the last few seconds and save that video and send it to the Reposte system. While the Reposte system may interact with several things, it can interact with other classes like the camera and arduino, but is primarily used to access our storage device holding all the videos.

3 Persistent Data Design

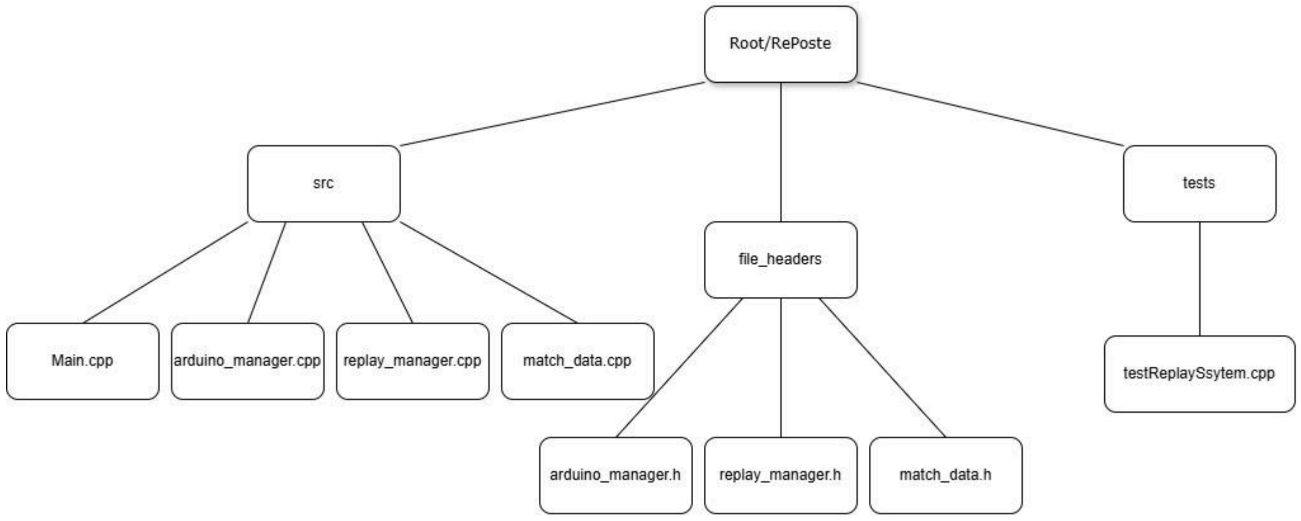
The fencing replay system RePoste will store replays to a local hard drive when a replay action is triggered. The replays will help aid referees with calling correct fouls or points that were hard to judge in real time. Each replay will provide the fencers participating, as well as match data like match time when the replay was triggered along with the current score to help sort and find necessary replays when they are required. The RePoste team is still debating whether to provide cloud storage for replays in addition to local storage. This will be decided during the next meeting with the client.

3.1 Database Descriptions

N/A within the scope of this project. Incorporating the ability to upload replay footage to a cloud database is something that the team will assess at a later date as a stretch goal.

3.2 File Descriptions

RePoste will have a concise file format for C++. A src folder will be a child of the root RePoste to help keep everything organized. 'Main.cpp' will include most of the work as it will handle interacting with the arduino, managing replays, and saving replays locally. There will be three other .cpp files, 'arduino_manager.cpp', 'replay_manager.cpp', and 'match_data.cpp' that will handle logic and operations for the arduino, replays, and match information. File_Headers will be another child of the RePoste root and will include the header files for 'arduino_manager.cpp', 'replay_manager.cpp', and 'match_data.cpp' named 'arduino_manager.h', 'replay_manager.h', and 'match_data.h' respectively. These header files will provide interfaces for their .cpp files to manage data and functionality. Tests will be another child of the root RePoste and will include test files 'testReplaySystem.cpp' to run automated tests.



Main.cpp:

- **matchData**: DataType = string/double
  - Contains match details
- **arduinoManager**: DataType = string/double
  - Handle arduino interactions

- **isRecording**: DataType = bool
    - Check if app is recording
- **replayCount**: DataType = double
    - Increment replay number for new save file

Arduino_manager.cpp:

- **arduinoConnection**: DataType = bool
    - Check if arduino is hooked up

Replay_system.cpp:

- **replayFile**: DataType = string(filePath)
    - File path for replays

Match_data.cpp:

- **matchID**: DataType = int
    - ID for current match
- **playerScore**: DataType = int
    - current score of each fencer
- **matchDuration**: DataType = double
    - Match timer
- **matchDate**: DataType = string
    - Date and time of match

4 Requirements Matrix

This section covers the requirements section of the SDD lining the functional requirements with their associated use cases and the corresponding system component that will satisfy or test the requirement. Some requirements do not have a direct component that would make sense to check so are instead represented in other explanations.

Functional Requirements	System Component
1) System shall record real time video at 60 frames/second so that review can be accurate. (**USE CASE #1**)	int fps60_check() function
2) System shall be able to replay specific moments immediately after they occur so the referee can make accurate judgements. (**USE CASE #1**)	void replay() function
3) System shall record audio along with the video so the audio cues will be captured as well. (****USE CASE**#2**)	audio_recording method

4) System shall work on both MacOS and Windows, and eventually Linux, so it can be used regardless of operating system **(USE CASE #4)**	GitHub Action Build: Runs-on: [ubuntu-latest, macos-latest, windows-latest]
5) System shall save the recorded video and audio with event markers, so that it can be reviewed and shared after a match. **(USE CASE #3)**	void save_file() void event_marker()
6) System shall interface with an Arduino device to capture signals from fencing equipment and log those events during video capture. **(USE CASE #5)**	Boolean signal_receive() function Void log_event() function
7) System shall interface with USB, USB-C, and HDMI inputs. (**USE CASE #6**)	boolean received_input()
8) System shall change the replay speed so that the referee can better understand what actions occurred. (**USE CASE #7**)	Replay speed can be modified (Default 100%): float replay_speed = 1; change_replay_speed() method (modifies replay by +/- .1)
9) System shall save recorded videos with a naming convention provided so the referee/official can find saved replays easier. (**USE CASE #8**)	User will input a string before the match: string bout_name = input; save_bout(bout_name, file) function
10) System shall have the ability to function in fullscreen mode so the referee has the best view of the replay. (**USE CASE #9**)	aspect_ratio() function

Appendix A – Agreement Between Customer and Contractor

This agreement confirms that the customer and the development team have reviewed and approved the contents of this Software Design Document(SDD). By signing this document, both parties agree to the project scope, specifications, and responsibilities as outlined. The team commits to delivering the software according to the requirements, timeline, and quality standards defined. The customer commits to supporting the team by providing necessary feedback and resources in a timely manner.

In the event that future changes to the document are required, both the customer and the development team agree to follow a structured change management process. This process will include documenting proposed changes, assessing their impact on the project timeline and resources, and obtaining mutual written consent from both parties before any adjustments are made to the original agreement.

Team Member Name	Signature	Date	Comments
Stephen Goodridge	Stephen Goodridge	11/3/2024	
David Geng	David Geng	11/3/2024	
Heath Miller	Heath Miller	11/3/2024	
Ryan Giles	Ryan Giles	11/3/2024	
Logan Geiser	Logan Geiser	11/3/24	

Client Name	Signature	Date	Comments

Appendix B – Team Review Sign-off

All members of the team have thoroughly reviewed this Software Design Document (SDD) and agree on its content and format. Each team member has provided their signature and the date below as a confirmation of their agreement. The comment area is available for any minor points that team members may want to note. There are no major points of contention, and all team members are aligned with the approach outlined in this document.

Team Member Name	Signature	Date	Comments
Stephen Goodridge	Stephen Goodridge	11/3/2024	
David Geng	David Geng	11/3/2024	
Heath Miller	Heath Miller	11/3/2024	
Ryan Giles	Ryan Giles	11/3/2024	
Logan Geiser	Logan Geiser	11/3/24	

Appendix C – Document Contributions

Identify how each member contributed to the creation of this document. Include what sections each member worked on and an estimate of the percentage of work they contributed. Remember that each team member <u>must</u> contribute to the writing (includes diagrams) for each document produced.

David: Section 2, 2.1, 4
Stephen: Section 3, 3.1, 3.2
Heath: Section 1, 1.1, Appendix A-B
Ryan: Section 2, 2.1 Diagram, 4
Logan: Section 2.2, Section 2.2 Diagram