HEMA Training Simulator Progress Report

Authors: Eric Reesor | 100970401
Conlan LaFreniere | 100971291

Supervisor: Dr. Samuel Ajila

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

December 3, 2017

Abstract

Learning Historical European Martial Arts (HEMA) require a large amount of practice and the best practice is against an opponent because they force you to react and repeated reaction builds habits. Trying to practice alone by going through the motions will only get you so far and places to spar in and trained opponents to spar against are rare.

The solution to this is to simulate a sparring match with a skilled opponent in a way that can be done solo. Virtual reality (VR) simulations will allow for the immersive feeling of actually fighting while an artificial intelligence (AI) opponent will push you to learn and grow. This is our goal.

As of December 6th, 2017, the input system has been assembled and completed. The basic user interface (UI) has been implemented and the basic simulation model has been rendered. The current project portions in progress are the utilization of the input data and increasing the UI's user friendliness. The portions remaining are the implementation of the AI opponent and the system through which the AI will recognize proper HEMA guards and stances.

Table of Contents

1.0 – Introduction
    1.1 – Background
    1.2 – Motivation
    1.3 – Problem Statement
    1.4 – Proposed Solution
2.0 – Technical Information
    2.1 – Technical Background and Terminology
    2.2 – Project Technical Details
        2.2.1 – Software Details
        2.2.2 – Hardware Details
3.0 – Conclusion
4.0 – References
5.0 – Appendices

List of Figures

Figure 1: Application Use Case Diagram
Figure 2: High Level Class Diagram
Figure 3: Input Flow Diagram (Left to Right)
Figure 4: Application Component Diagram

1.0 - **Introduction**

1.1 – *Project Background*

The goal of this project is to design and develop a reliable and accurate HEMA training simulator, which will allow the user to learn and practice the set of stances and actions which make up the art of the German longsword. More specifically, to teach and reinforce instinctual reactions one will require to spar and fight safely within the Liechtenauer school of fencing. This will be done though sequential drills and reaction based training. The user interacts with the system through a video output device (screen, VR headset, etc.) and an Arduino wand-style motion input device, to mimic holding a sword.

1.2 – *Motivation*

This project has a firm starting conditions in application and hardware development. It must contain an application, capable of communicating with the customized wand-style input device. It must also be capable of interpreting a player's movements, and comparing those interpretations with valid HEMA form. All this information must then be processed and presented to the user in the simplest way possible, since any teaching tool must focus as much as possible on the subject matter, and not it's "game" aspects.

1.3 – *Problem Statement*

In order to complete this project, there are several problems in need of a solution. The wand-style input device must communicate successfully with a PC in order to collect user inputs. There also needs to be an application front-end capable of displaying all necessary input information to the designer, or user. Once these are achieved, a standardized 3d movement input format can be created. This will allow the manual programming of different HEMA styles. After these are finished, there must also be real-time recognition software capable of comparing the user's inputs to valid HEMA forms.

1.4 – *Proposed Solution*

The proposed solution to these problems is a Unity based application paired with an Arduino based movement capture device.

1.5 – *Accomplishments*

As of December 1, 2017, the following describes all completed and to be completed components of the project:

Complete
- Wand style input device
- Application frontend and 3d sword model

In Progress
- Conversion of input data to useable standardized format
- Fleshing out UI and system control to clarify usability

To Be Completed
- Program valid HEMA movements using standardized input format
- Design movement recognition software to determine HEMA validity

2.0 – **Technical Information**

2.1 – *Technical Background and Terminology*

HEMA: Historical European Martial Arts
Liechtenauer: A German fencing master of the 14th century, whose teachings this project is centered around
Federschwert: A thin fencing training sword used to train pupils of the German school of fencing

2.2 – *Project Technical Details*

2.2.1 – *Software Details*

The front-end design process began by formulating all the “activities” that a user would do in this application. The program must allow the user to participate in specific pre-made training scenarios, or to freely practice specific HEMA combat movements. The user interacts with the software through a custom wand-style movement input device, designed to mimic a training sword, so one must be able to alter relevant system/controller settings. Ideally, the user could customize their training sessions as well.

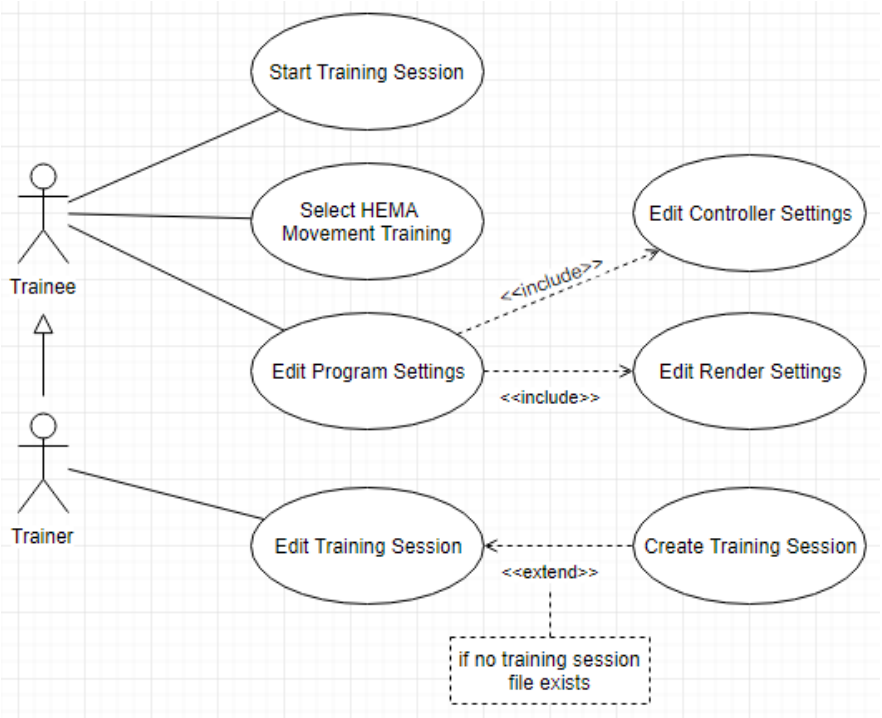

Figure 1: Application Use Case Diagram

Use Case Descriptions

Start Training Session: User selects a training session to load, then proceeds through said training session using the Arduino Wand. This encapsulates the primary function of the system.

Select HEMA Movement Training: Select a specific HEMA movement to practice, as opposed selecting an entire training session.

Edit Program Settings: Screen allowing the user access to and privilege over alterable system options
Edit Controller Settings: Access to any Arduino Wand settings exposed to the user.
Edit Render Settings: Access to any Unity or program-specific render settings.
Edit Training Session: Allows user to customize which movements will appear in their training session and in what order, as well as any other session specific settings.
Create Training Session: Extending the "Edit Training Session" use case with the necessary training session creation tools. Only necessary on creation of a new training session.

From this diagram, one can derive several different system objects: a "training session", a "HEMA movement", and the Arduino Wand, and by extension it's 3d avatar. The use cases also indicate a need for different user interfaces: for training, for editing system/controller settings, for editing, and presumable a "main menu" to tie them all together. Finally, the system will need some internal mechanism to represent player inputs, since HEMA has specific acceptable "forms" similar to other martial arts like karate. Pulling all this information together, the initial class diagram is derived.

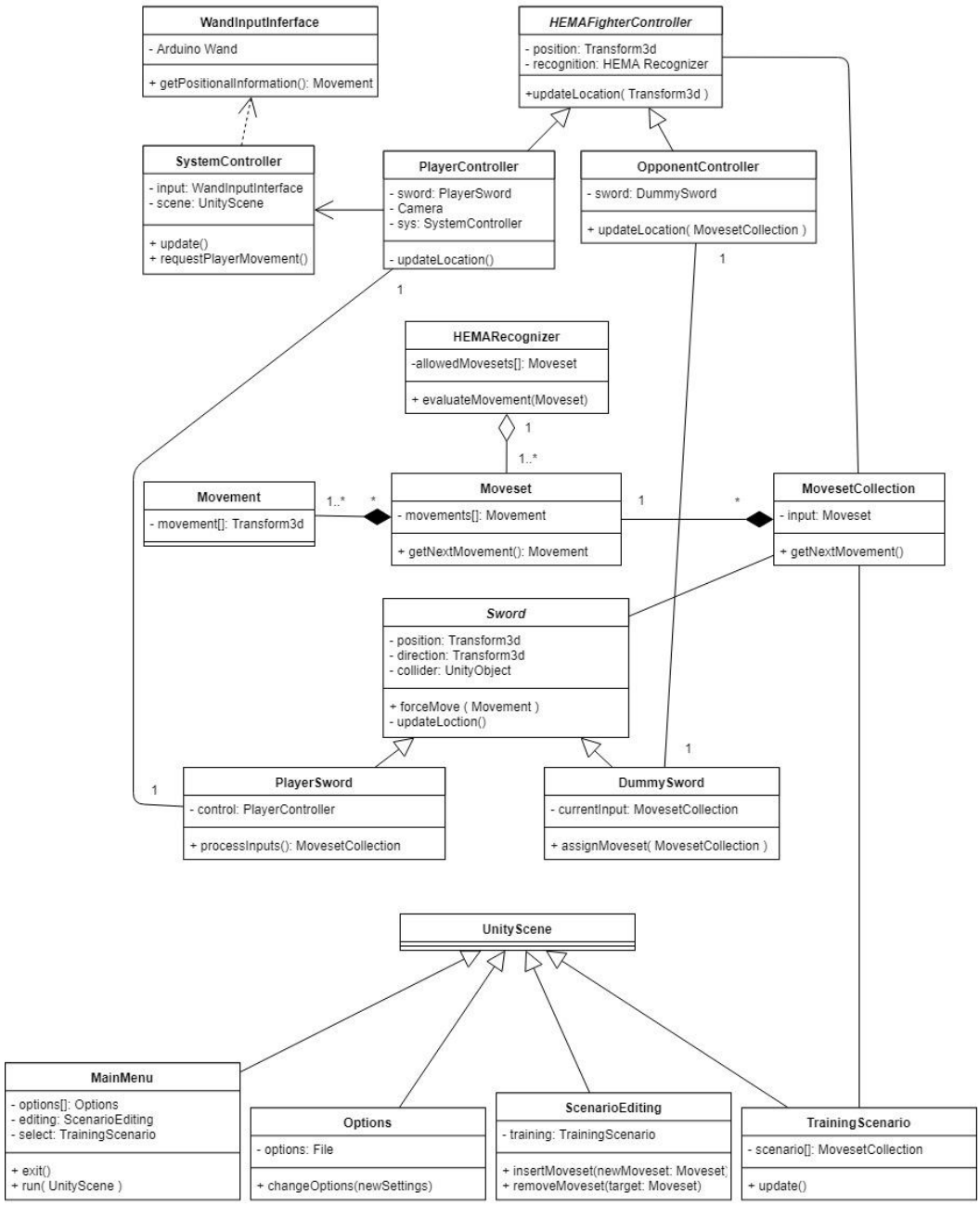

Figure 2: High Level Class Diagram

Class Descriptions

UnityScene: Unity has an internal structure called a “scene”, which is a method of encapsulating individual “screens” of your program. It will be used as the foundation for each of our views.

MainMenu: The start-up view, supplies user access to all the other views.

Options: A view wherein the user can alter all available program settings.

ScenarioEditing: Opens a TrainingScenario in edit mode, allowing for the injection and removal of HEMA movements from said scenario.

TrainingScenario: The view through which an available training session can be loaded and used.

MovesetCollection: An ordered collection of Movesets that combiner to describe a training session.

Moveset: An ordered collection of Movements that describes a particular, valid HEMA move.

Movement: Data representing a specific, indivisible movement, used as a building block by Moveset.

HEMARecognizer: A processing unit capable of interpreting player movement and determining HEMA validity based on its collection of validated Movesets.

Sword: An object describing all common functionality between the player-controlled sword and opponent-controlled swords.

PlayerSword: An extension of Sword, specific to the Player avatar, controlled by the PlayerController.

DummySword: An extension of Sword, specific to non-player avatars, controlled by an OpponentController.

WandInputInterface: The interface between the Arduino Wand and Unity, responsible for processing of raw input and conversion to forms usable by the Unity program.

SystemController: The main program controller, responsible for handling all user input, and passing along necessary input to the HEMAFighterControllers.

HEMAFighterController: An abstract controller that encapsulates all functionality common between PlayerController and OpponentController.

PlayerController: The player avatar controller. Accepts input from the system controller and forwards the necessary information to the HEMARecognizer.

OpponentController: The opponent avatar controller. Accepts input from program-defined inputs, and forwards the necessary information to the HEMARecognizer.

When analyzing the design goals of this project, the highest priority is clearly usability, since if the user cannot understand their training scenario, they won't learn, rendering the system useless. Another clear design goal is responsiveness, since the player must know when they are or are not in proper form and must receive direct feedback from their inputs, like they are holding a sword. This means input data losses or system timing errors will result in an effective reduction of teaching capacity. This only applies to a finite time granularity, since system responsiveness must only exceed controller input rate. To that end, the flow of information through the system has been mapped to a Boundary-Entity-Control diagram. Note that each red line marks a decrease in information volume (from left to right) due to a narrowing of focus within the program.

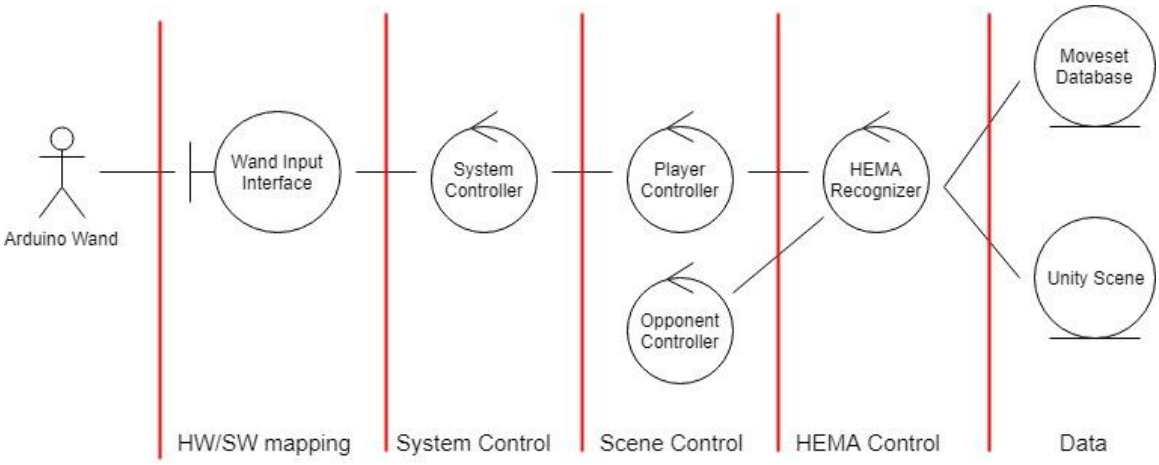

Figure 3: Input Flow Diagram (Left to Right)

Using these design decisions, the system can start being decomposed into distinct semi-autonomous units. Starting within the HEMA simulation, there will need to be two separate entities, the player and their opponent, each with a controller class to facilitate data exchange. Next, the sword, which is "owned" by the player, but designed separate with its own controller class to handle player inputs and exchange data with the HEMA

recognizer. Zooming out from the training simulation, the entire system will also need a point of contact with the Arduino wand, another controller class, responsible for processing raw Arduino inputs. These are then fed to the system controller to handle program-wide inputs, such as main menu control, then divided up and conveyed to relevant subsystem (i.e. supplying PlayerController with sword movements) as per Figure 3. The once the inputs have updated the current system instance, they will need to be fed to our HEMA database for verification.

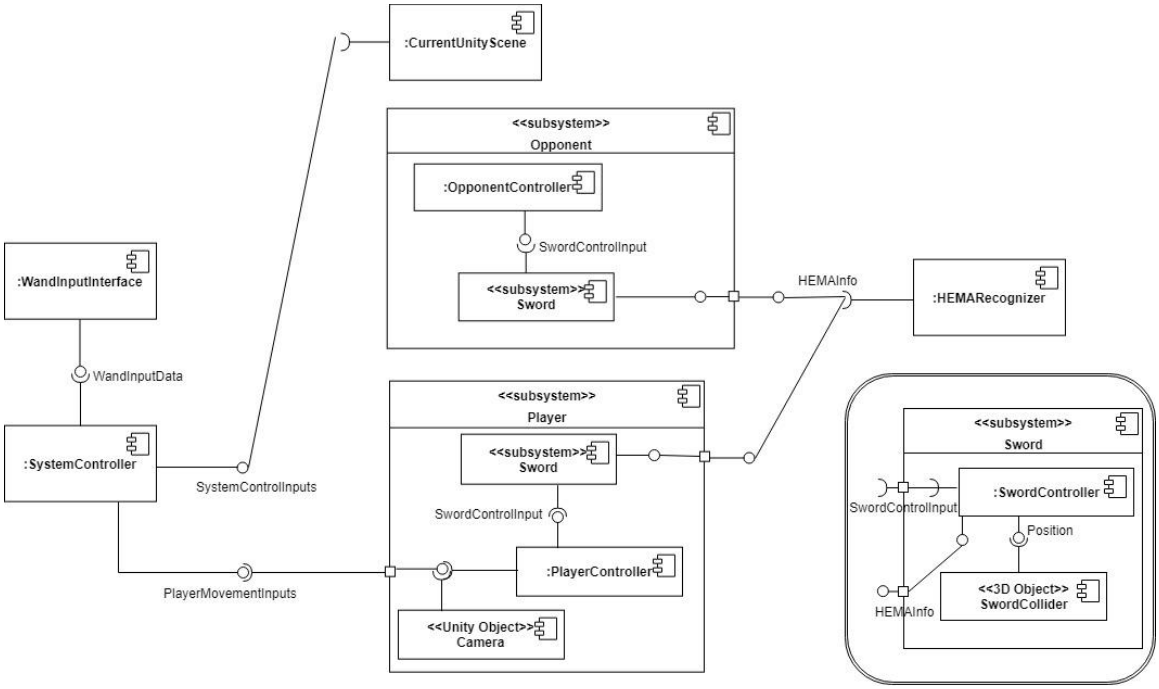

Figure 4: Application Component Diagram

Component Descriptions

WandInputInterface: Responsible for accepting input data from the Arduino wand and converting it into a form useable by SystemController

SystemController: Responsible for processing inputs from WandInputInterface and forwarding them to whatever system they are bound for

CurrentUnityScene: Less an actual component than representation of Unity's overlap with our design. Unity has a built in "scene" object that handles much of the UI and skeletal system control we would otherwise implement ourselves. System controller will feed relevant control inputs to the "CurrentUnityScene"

Player: Encapsulates the user's presence in a training simulation. Accepts player inputs from the system controller and sends HEMA data to the HEMARecognizer

PlayerController: Responsible for processing player inputs, generating SwordControlInputs for coordination between the player's in-game avatar and sword

Camera: Another prefab Unity object, handles rendering the internal 3d scene model to the display

Sword: Encapsulates sword functionality excluding autonomy, as both the player and opponent may need swords

SwordController: Processes SwordControlInputs and updates the SwordCollider accordingly

SwordCollider: The 3d model that visually represents a sword, and it's collision polygon.

Opponent: Responsible for any "AI" the player must face-off against, capable of generating SwordControlInputs based on acceptable HEMA forms

OpponentController: Responsible for processing opponent SwordControlInputs and coordinating between the player's in-game avatar and sword

Sword: See Player->Sword

HEMARecognizer: Contains DB of acceptable HEMA forms, responsible for processing the scene to check for HEMA form violations

2.2.2 – *Hardware Details*

The hardware of the HEMA Simulator system has three components. They are the input system, the simulator system and the output system. The input system initially was a Nintendo Wiimote Plus, however this system was inadequate to fully capture movement in 3D space. As an alternative, an Arduino Uno with an HC-05 Bluetooth Adapter and a 10DOF IMU sensor was assembled. The IMU sensor collects data from a 3-axis accelerometer, 3-axis gyroscope and a 3-axis compass to fully capture a sword's acceleration, position and rotation. The HC-05 allows wireless communication with the simulator system. The simulator system was originally going to be the same as the output

system as the Unity application was to be deployed on an Apple iPhone 5s. There was a compatibility issue with the HC-05 because it is not included in the MFi program that the iPhone supports with little else being supported. To solve this, the simulation system has changed to a Windows PC with the output system being a separate device. Ideally, the incompatibility will be circumvented by using the PC as a midpoint which will have both the input and output systems connected wirelessly. The PC will process the input data, make a choice and send it to the output system to display.

3.0 – **Conclusion**

This project's intention was to design and develop an application capable of training users in the Liechtenauer style of Historical European Martial Arts. The project has mostly kept up with the proposal's vision, but there have been setbacks, the largest of which was the Wii Remote's inability to capture sufficient data to emulate a sword. This had a non-trivial impact on the timetable, but progress is still being made, and the project is still within reach of an April completion deadline. Design and development will continue into the new year.