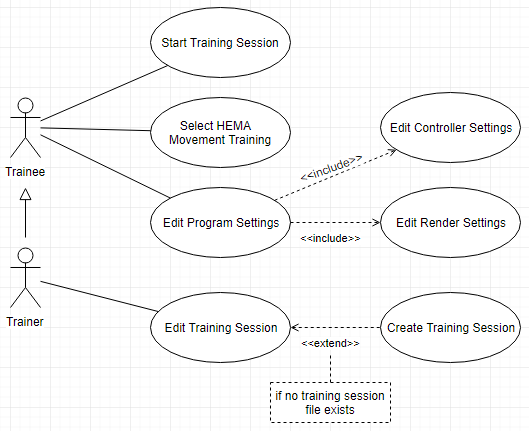
HEMA Simulator System Overview

System Specifications

This system is for the training of HEMA, specifically Fechtschulen, a school of German fencing. The program will allow the user to participate in specific pre-made training scenarios, or to freely practice specific combat movements. The user interacts with the software through a custom wand-style (*currently Arduino*) movement input device, designed to mimic a training sword. Users must also be able to alter relevant system/controller settings, and ideally customize training sessions as well.

Figure 1: 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.

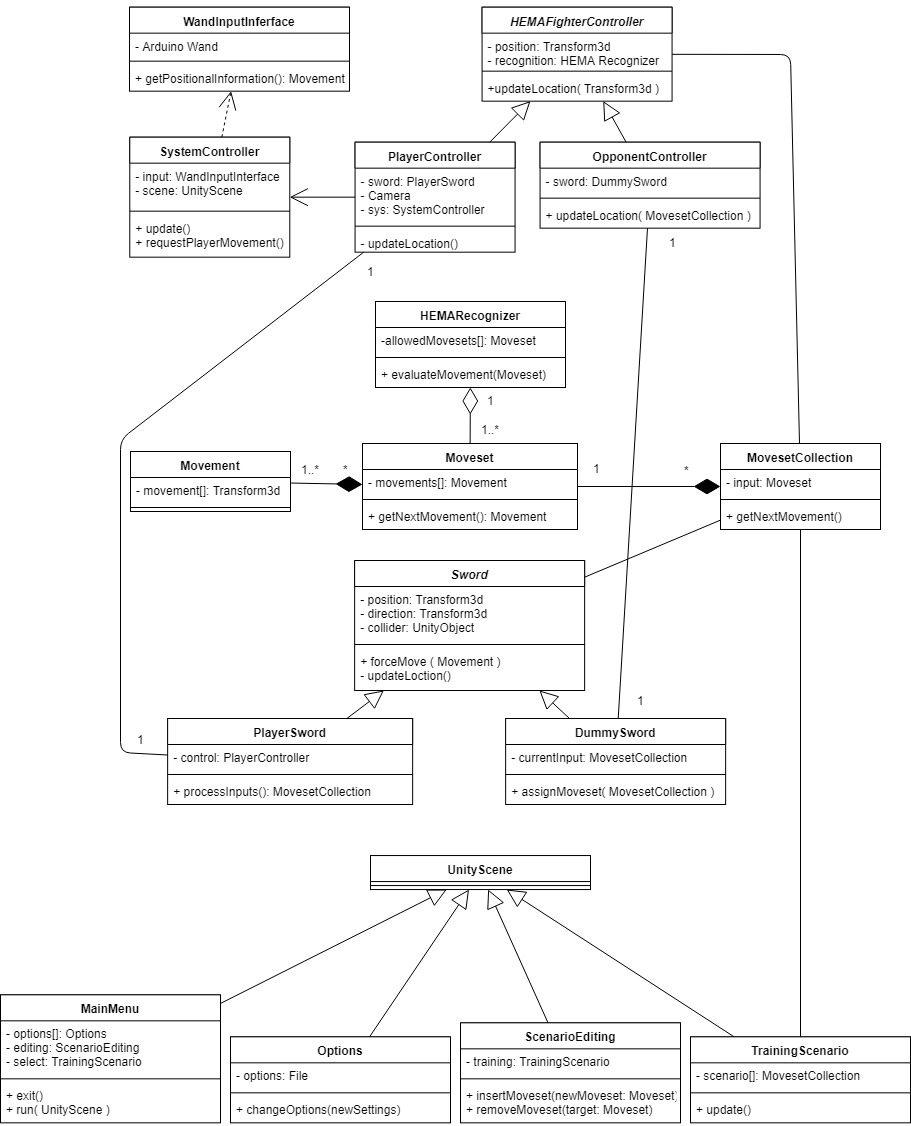
**Edit Render Settings**: Access to any Unity or program-specific render settings.

**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.

This diagram starts to clearly indicate several independent system objects: a “training session”, a “HEMA movement”, and the Arduino Wand, and by extension it’s in game avatar. The specifications also indicate several different user interfaces: one for training, one for editing system/controller settings, one 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 functionality together, we get the following initial class diagram.

Figure 2: HEMA System 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.

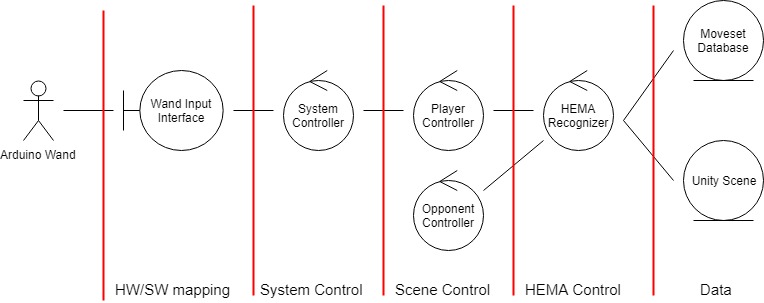
**HEMAFighterController**: An abstract controller that encapsulates all functionality common between PlayerController and OpponentController.

**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.

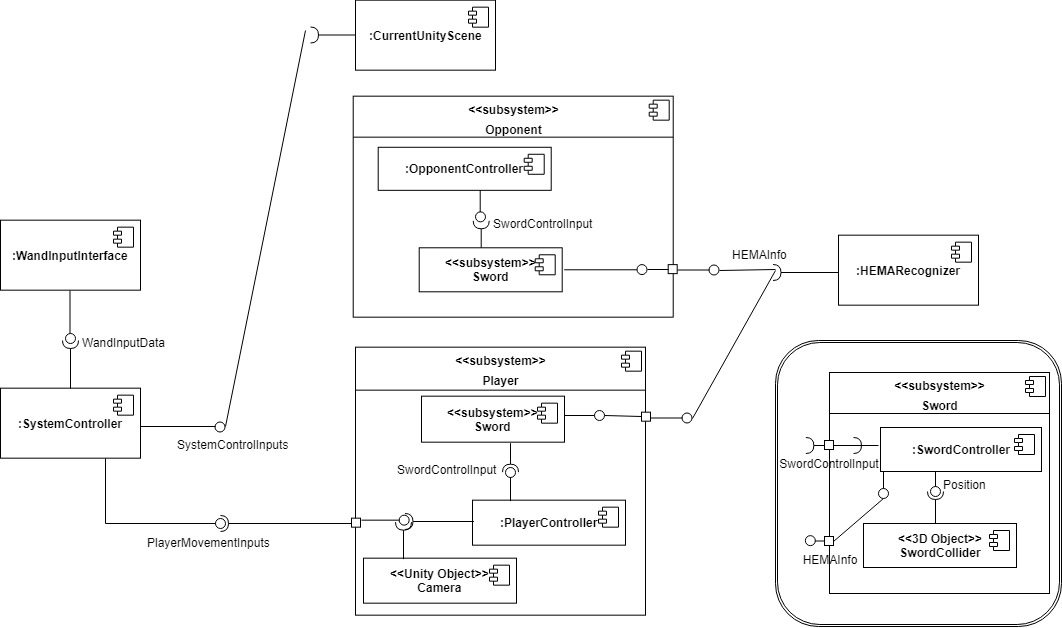
When analyzing the design goals of this project, the highest priority is clearly *usability*, as if the user cannot successfully understand their training scenario, they won’t learn, rendering the system useless. Another clear design goal is *responsiveness* since the player must know when they are or are not in proper form, so input data losses or system errors will result in an effective reduction of teaching capacity. Note that responsiveness must only exceed player input capacity, any increases beyond that are redundant. To that end, the system’s flow of information has been mapped in the following 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



Using these design decisions, we can start to decompose the system into distinct semi-autonomous units. Starting within the HEMA simulation, there will need to be two separate entities, the player and their opponent (if included), each with their own controller class to facilitate data exchange. Next is the sword, which is “contained” by the player but also needs its own controller to handle player inputs and exchange data with the HEMA recognizer. Zooming out from the training simulation, the entire system will also need one interface to the Arduino wand, 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). The once the inputs have updated the current system instance, they will need to be fed to our HEMA database for verification.

Figure 4: System 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

Glossary (of terms otherwise undefined)

**Avatar**: All components that represent an in-engine “physical” presence.

**Training** **Session**: Training scenario; a set of HEMA actions to be executing in sequence by the user, and the accompanying Unity scene.

**HEMA Movement**: A physical motion that is recognized as a valid HEMA action, ex. proper sword grip, follow-through, etc.

**Arduino Wand**: The hardware control system, as devised and implemented by Eric Reesor (presently in progress).