Ars Ensis

Lovagi Kör és Kardvívó Iskola Egyesület

Free Scholler
Project documentation

Modelling and visualising fencing exercises

Written by: Márkus János Dániel

Free Scholler candidate

Datasheet

Year of writing	2018-2019	
Name of the candidate	Márkus János Dániel	
External consultant	Name:	Sándor Zsolt
	Occupation:	Software developer
	Workplace:	
	Address of workplace:	

Internal consultant	Name:	Dr. Miskolczi Mátyás
	Occupation:	
	Workplace:	
	Address of workplace:	

I have reviewed the documentation, it can / can not be submitted.

Date: _____ External consultant: _____

Mentor: _____

Evaluation sheet

Author (candidate): Márkus János Dániel

Title of the work: Modelling and visualising fencing exercises

Judge's name, workplace, occupation:

Forján Valentin, Software Developer

---

1. Choice of topic (max. 5 points)	Given points	**5**
2. Structure, style of work (max. 8 points)	Given points	**7**
3. Usage of literature (max. 10 points)	Given points	**10**
4. Quality of research (max. 20 points)	Given points	**19**
5. Practical applicability of work: (max. 7 points)	Given points	**7**

Total points: **Points:** **48**

Judge's opinion:

A történelmi hagyományok, így a vívástechnika, vagy annak gyakorlati alkalmazása egy, a rendelkekezésre álló technológiák tekintetében, informatikai oldalról méltánytalanul elhanyagolt terület. János dolgozatában eme terület támogatására tett kisérletet, egy olyan adatmodell fejlesztésével, amely az egyes gyakorlatok olyan mód történő általános leírására melyet az informatikai rendszerek tárolni illetve feldolgozni képesek. A modell létrehozása során dicséretes az eszközválasztás, melynek lévén a modell illetve a program platformfüggetlen, ezáltal könnyen implementálható a különféle jellemzőkkel bíró környezetekben, eszközökön. Külön kiemelném a programozási minták és jógyakorlatok szakszerű és helyes használatát, melynek révén a program karbantartása és bővíthetősége támogatásra került.

A feladatot egyértelműen nehezítette, hogy az adatmodell létrehozása során egyszerre kellett a választott fegyver és hagyomány technikáinak elméleti alapjairól mély ismeretekkel rendelkeznie, és a programozási illetve adatmodellezés nehézségeivel, korlátaival számolnia. Az elkészült program által jelenleg megjelenített információk alapján, és Lovag Arlow Gusztáv munkásságának ismerete nélkül – mely tényező alapján elfogulatlan bírálóként tudok nyilatkozni a látottak érthetőségéről - úgy vélem ezeket a nehézségéket János sikeresen leküzdötte.

Habár a modell vizualizációs része jelenleg csak a témában legalább minimális szinten járatos vívók részére hasznos, úgy gondolom az elkészített Mestermunka remek alapja lehet egy jövőbeli ilyen irányú fejlesztésnek, melynek végeredményeként egy formagyakorlatokat leíró és azokat illusztrált módon megjelenítő alkalmazás elkészítése is lehetségessé válik, amely már közvetlenül a különféle történelmi vívással foglalkozó csoportok gyakorlati oktatásai során is felhasználható, akár kezdő vívók részére is.

Mindezek alapján javaslom a Jelöltet Free Scholler díjvívásra bocsátani!

A Mestermunka és a kísérő szakdolgozat elkészítését 48 pontra értékelem.

Date: _____ Judge: _____

Modelling and visualising fencing exercises

Free Scholler
project documentation

Márkus János Dániel

aresius423@gmail.com

Contents

1 Abstract

This documentation explains the thought process behind the creation of the web application titled ”Fencercise”, with some details on the implementation.

The app allows the user to build up and display fencing systems, and exercises. There was a considerable amount of work put into building up a data model that is generic enough to be able to represent any fencing system.

To show the capabilities of the app, the majority of Arlow Gusztáv's book ”Kardvívás” has been processed. The end result is a demonstration of the data model in action, with a clearer picture of what still needs to be done, in order to make a responsive and truly usable application.

The source code of the project is available at:

https://github.com/Aresius423/fencercise

2 Goal

In 2016, the sabre research group of Ars Ensis was formed. Their main goal was to study Lovag Arlow Gusztáv's 1902 book: Kardvívás. During the research phase, it became apparent, that the exercises are not particularly well structured, and are often difficult to understand and carry out.

As the material was starting to get passed on to students, the following issues were encountered:

- Due to the complexity of the system, by the time the exercises were reached, small details crucial to the proper execution were forgotten.
- The author often gave both fencers multiple options, without the possible paths being marked. This made the planning of trainings difficult, as one couldn't simply write down the number of the exercise.
- Handling the entire book, and finding the required information in the midst of a training is difficult.

The aim of this project was to create an aid for recalling the information gained from the book describing the fencing system. While it is possible to cram every single bit of information into the data model, it cannot provide the progression and the overview that the source material does. It does not replace the existing resources, it merely tries to enhance them.

3 Design

During development, modularity and portability were prioritised. The application has to be usable from PCs and phones alike, so it must be presented as a web application. The choice of programming language fell on JavaScript, due to it being a good fit for the assignment. It has a broad range of libraries available for data visualisation, and requires no backend to be present. Python and Scala were also considered, but were deemed less suitable for this sort application.

Modularity is reached by the complete separation of the data model and the view model. Should it be necessary, an entirely different view can be produced by swapping out the drawer module to another one, tailored for this use case.

Being able to model various fencing systems was imperative. The end result had to be able to represent the simple, sequential exchanges of early sabre codices, and the intricate tempo-breaking actions of later systems. It had to support other weapons as well, not only sabre. While these have not been tested in practice, the application should handle anything from the fairly similarly describable bolognese one handed sword, through single-person montante rules, to elaborate longsword plays.

Step-by-step execution of exercises was a crucial feature from the start. Being able to inspect any single moment helps recreating the entire flow accurately.

3.1 Participant model

A person (and by extension, their sword) can be broken down into parts. Every action can be simplified to a few part modifiers, that, when applied to a person in any state, will result in the same state, as a correctly executed action would produce. This is elaborated on on page 6.

During training, the following questions often arise:

- Where is my <body part>?
- Why is it there?

Recording the results of actions in a traceable fashion improves clarity. Inadvertent movements can influence the success of an action, therefore it is important to keep track of everything that did, or was supposed to lead to the participant's current state.

The fencing system determines, which parts should be represented in the model. For example, in sabre fencing, the position of the left hand is static - always behind the fencer's back - therefore it does not need to be represented. In other systems however, the off-hand may be used, possibly with a secondary weapon even, in which case it can and should be added to the list of parts.

3.2 Time model

There are ways to describe the flow of an exercise - the most commonly used one comes from classic Italian rapier manuals. The issue with these systems is that the same exercise can be described in multiple ways, depending on which person's perspective the play is viewed, and what the written or unwritten intentions are behind the action. This in itself is not an issue, however fails to fulfill the following requirements:

- Transcription of exercises must be easy
- The model shall be free of interpretation

Because of this, the time model was simplified to two types of actions:

- Concurrent actions
- Sequential actions

This means that composite actions (e.g. feints) must be described as multiple, separate actions. There are benefits to this: understanding of the exercise is not required for its transcription. The resolution at which the participant model can be observed is increased.

Any additional timing information that cannot be represented with these actions is to be written in the action's notes.

3.3 Action model

The techniques are composed of invocations and assertions.

Invocations are references to other techniques, which are executed, when the action is performed. For example, the end position of a tierce parry is a tierce invito[1], therefore the *tierce parry* action will invoke the *tierce invito* action. Every implication a *tierce invito* has, applies to the action invoking it. Invocations aren't necessarily performed concurrently - the exact method of execution should be contained in the notes, or the source.

The extra information that is the difference between performing a tierce parry and assuming the tierce invito position can come from the name (presuming that the reader is familiar with the execution of the technique), the note attached to the action, or the composition of invocation. For example, the end state of a tierce parry and a contre-tierce is the same. In the latter's representation, however, not only the tierce invito is invoked, but also the contre-parade technique. Even though it doesn't influence the fencer's state, it provides additional information for the execution.

Assertions are the aforementioned modifications to the fencer's state. They set the part's state to a value. For example, the *lunge* technique will set the fencer's foot attribute to *lunge*, while the *recovery from lunge* sets it back to its default value.

[1] (Arlow 1902, p. 46)

A technique without invocations is consistent - multiple assertions for the same part cannot be made. However, when multiple techniques are invoked, which make different assertions for the same part, a conflict arises. Assertions in the technique override those found in its invocations, thus being able to solve conflicts.

Example

The following two techniques are defined:

A thrust:

- Footwork: lunge
- Sword hand: extended

A deep reverse lunge:

- Footwork: reverse lunge
- Off hand: supporting weight on floor

The *under stop-thrust*[2] is an action that combines a thrust with a reverse lunge. Since both techniques influence the 'footwork' part, the result will be conflicting.

To avoid the issue altogether, an action can be created, that accurately describes the under stop-thrust. This is not recommended, when the technique is not used repeatedly, therefore needing to be accessible with a single reference.

To resolve the conflict, an additional assertion can be declared, which explicitly sets the *footwork* part to *reverse lunge*. This has the additional advantage that it preserves the actions invoked by both sub-actions.

[2]The technique presented (Hutton 1889, p. 98) is described differently, for the sake of demonstration.

3.4 Exercise model

An exercise consists of a set-up, and an action flow.

Initially, both participants start from an undefined state - unless its state is described in the source, it is up for interpretation. If the starting position is described, a list of set-up actions can be provided, which will result in the correct state.

The action flow is a sequence of composite actions. A flow step can also be a multi-participant step, when it contains more than one of these composites. This is typically used when the two participants perform an action truly simultaneously. Reactions should be described in a separate step, even if the movements overlap in time.

A flow step has similar attributes as a technique - it may have a list of invocations, assertions, and a note. The participant who performs the step must be marked.

The way the invocations are interpreted is different. While a technique is considered to be a singular action, its timing information not being interpreted by the application, an exercise does carry such information through the existence of flow steps. If two actions are performed concurrently, they shall be invoked in the same step, but if they are performed sequentially, they should be in separate steps.

4 Architecture

As mentioned previously, the application is built up from separate units. These conform to the MVC (Model-View-Controller) architectural pattern.

- **Model:** The model contains the data structure that is being manipulated. How this data is interpreted, and how an exercise is simulated, is up to the algorithms contained in the model.
- **Controller:** The user interacts with the controller - in this case, the UI elements (e.g. advancing the exercise by one step) activate a method in the controller, which notifies the model of the requested operation.
- **View:** The module that creates the page the user sees, based on the data extracted from the model.

To make the data model fully independent from the view drawer module, the observer design pattern was used, with a push model. The model is implemented as a subscribable: when it decides that its properties have changed enough to warrant a re-draw, it notifies its subscribers that a change has occurred. The current state of the data model is pushed to the subscribers in its entirety, and the view drawer module is expected to be able to parse and display the data without requiring any additional information.

5 Data model

5.1 Data structure

The data gathered from a single source material is expected to be contained in a single JSON file. The structure of the data is as follows:

- system
  - title:String = the title of the source
  - weapon:String
  - bodyparts:List[String] = the fencers' attributes
  - participants:List[String] = the names of the exercises' participants
- techniques:List[Technique]
- additional-notes:List[AdditionalNote]
- exercises:List[Exercise]

5.1.1 Technique

a technique is a JSON object with the following fields:

- name:String
- invokes:List[String] - techniques that this one is composed of
- assertions:Dictionary - key-pair values, where the key is the part, and the value is their desired state
- note:String - additional notes
- images:List[String] - path to the image files to be displayed alongside the technique
- alias:List[String] - synonyms for this technique

This is the basic building block of the exercises.

5.1.2 AdditionalNote

These additional notes contain information that can be referenced and displayed. It is a JSON object with the following fields:

- name:String
- note:String

5.1.3 Exercise

A variant of an exercise, or a single exercise without decisions that would influence the actions. It is a JSON object with the following fields:

- name:String
- init:
  - <participant 1>:List[String] - invocations for the initial set-up for participant 1
  - <participant 2>:List[String] - invocations for the initial set-up for participant 2
  - note:List[String] - notes to be displayed
- flow:List[Steps]

A single participant step:

- actor:String - the actor who executes the step
- actions:List[String] - invocations
- assertions:Dictionary - assertions
- note:String - additional notes

A Step object can either be a single participant step, or an array of them.

5.2 Model internals

The model class can be instantiated with a JSON file containing the previously discussed data structure. The contents are loaded into memory after validation. References to non-existent techniques are logged, but are not considered fatal errors.

The model keeps track of the selected exercise, and the step, at which it is to be displayed. The instructions and notes to be displayed are stored separately, and the participant part states are re-evaluated and stored at each step.

To answer both questions posed in section 3.1, the model keeps track of the changes made in the current step. If multiple actions set the state of a body part, all of the traces will be stored.

If the body part's state is inconsistent, i.e. set to different values by two or more actions, the part is marked as invalid. A consistent change (part set to the same value by one or more actions) will result in the part being marked as updated, as to aid in displaying the part differences between steps. Assertions set the status of the part to 'assertion' - thus marking it as valid.

The following interfaces can trigger changes in the model:

- loadExercise - the exercise with the index (provided as a parameter) from the array of exercises is loaded.
- stepExercise - the currently selected exercise is advanced by one step
- unstepExercise - the previous step of the currently selected exercise is made active
- setExerciseStage - the exercise is set to the step requested - the number of the step must be provided as a parameter.

As a subscribable, the following interfaces are provided:

- subscribe - the JS object (provided as a parameter) is added to the list of subscribers. The object must implement the update method, which takes a SystemData object as its sole parameter. This is called when the model's state changes.
- unsubscribe - removes the JS object (provided as a parameter) from the list of subscribers.

6 View

The view drawer formats and displays the data available to the user. Discussing the details of the currently existing implementation is not in the scope of this project documentation, as it is currently in development.

The primary view the user encounters is the landing page. Choosing the fencing system and exercise to load happens here.

The responsibilities of a view drawer are:

- Store a reference to a controller, which is received at instantiation
- Display controls with which the user can navigate the steps of an exercise
- Bind the controls to the appropriate requests in the controller
- Display the participants and their parts
- Implement the update(DataModel) method, which updates:
  - The body parts (see: DataModel.exerciseTrace)
  - The body parts' status (valid values can be retrieved from DataModel.partStates)
  - The body parts' most recent trace (see: DataModel.exerciseTrace)
  - The current instructions (see: DataModel.activeInstructions)
  - The current note (see: DataModel.activeNote)
- Display
  - notes, images, etc. of techniques
  - additional notes

  on request, and provide a way to access them

7 Further development

The user interface is only suitable for demonstration. A responsive UI must be created to make the app fully usable.

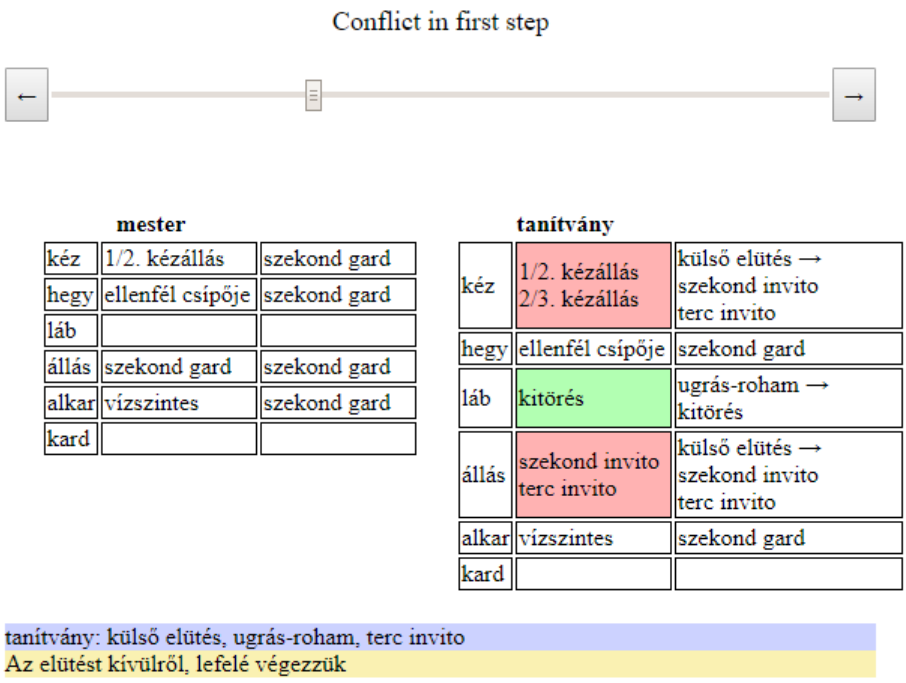
Certain properties of the system are present, but not displayed.

Currently, systems and exercises must be put together manually in a JSON file. An interface through which the user can ”click together” an exercise, or enter a system is to be created.

A feature which generates plantUML sequence diagrams from the entered exercises can improve offline usability.

8 Figures

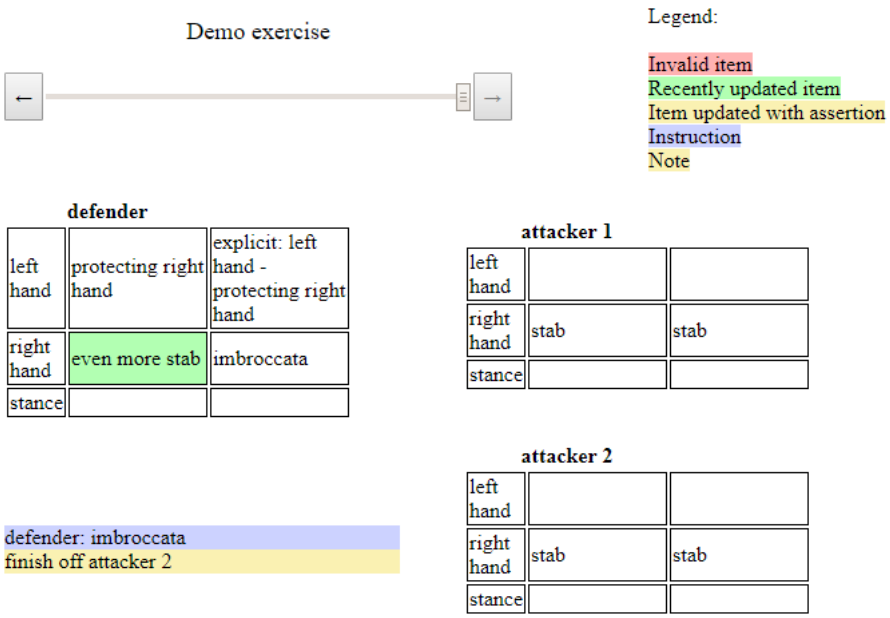
A simple demonstration of an exercise being displayed



A demonstration of conflicting steps

The depicted step consists of three concurrent actions, two of which conflict.

The end position of a *külső elütés* is a szekond invito, which affects the same body parts as the provided step *terc invito*, but (as can be seen from the trace fields), the result is not consistent.



A demo exercise showing the capability to work with different body parts, and more than two participants