BarreBuild: Language Specification

Michelle Wang Serah Park

May 20, 2024

1. Introduction

Video presentation: `https://drive.google.com/file/d/1OfAAns5iu7UpG7sXPT-wenUON1XNY5hK/view?usp=sharing`

Our language is called BarreBuild. It solves the problem of communicating a barre combination (combos) for ballet classes by transforming given instructions (such as from a ballet instructor) into a visual animation (for ballet students to watch).

Barres are used extensively in ballet training and warm up exercises, where such exercises are commonly referred to as barre work. Barre work/barre combos are an integral part of every ballet class, and combos are often repeated every class. A ballet class typically begins with barre combos roughly in this order:
1. Pliés
2. Tendus
3. Dégagés
4. Ronds de Jambe
5. Frappés
6. Fondus
7. Grands Battements
8. Adagio
9. Stretch

We believe that this problem should have its own programming language due to the fact that barre combos occur in almost every ballet class, but the creation and teaching of these combos relies on the ballet teacher's memory and demonstration. Since these barre combos are typically very formulaic and have a clear structure, our programming language allows ballet teachers to create barre combinations prior to class and be able to show the generated animation to students instead of relying solely on their memory to recall and teach the combos. This also makes teaching barre combos more accessible, as it removes the need for a teacher to physically demonstrate.

This programming language could also be used by individuals other than ballet teachers, such as a ballet student who wants to transcribe a barre combo in an easy, digestible way for future reference (without actually recording themselves doing it).

2. Design Principles

Aesthetically, our programming language is meant to be simple and readable so that dancers (or anyone really) are able to use it without extensive coding knowledge. With a simple list format that resembles how one would typically deliver a ballet barre combo verbally, our language consolidates all of the information and outputs it into a visual video format.

Technically, we rely on the application and composition of functions to piece together combinations of dance moves with specified details from a set of simple, elementary moves. Our programming language generates a series of JPG frames (pictures of stick figures doing the moves) combined into an MP4 video that displays an animation of how the given barre combo is supposed to look like.

3. Examples

Three examples of sample programs are shown below. The examples are stored in the 'examples' directory.

1. `dotnet run "tendu first front 1"`
This program should create (or replace) the Combo.mp4 file with a video of a stick figure doing one tendu to the front from first position. The command line text for this example is in example1.txt.

1. `dotnet run "tendu first side 1 degage first side 1 battement first side 1 retire first na 1"`
This program should create (or replace) the Combo.mp4 file with a video of a stick figure doing a tendu, degage, and battement to the side from first, and then a retire from first. The command line text for this example is in example2.txt.

1. `dotnet run "plie fifth na 3 eleve fifth na 1 plie fifth na 3 eleve fifth na 1"`
This program should create (or replace) the Combo.mp4 file with a video of a stick figure doing three plies in fifth and then an eleve in fifth, then three more plies in fifth and another eleve in fifth. The command line text for this example is in example3.txt

4. Language Concepts

To write a program in this language, the user will have to understand the the basic components of a barre combo. Barre combos are made up of a series of moves, each of which is a step done in a certain direction (sometimes) and from a certain position. These moves can also be repeated a certain amount of times. A complete program can be as simple as just one move with a specified step, direction, position, and number of repetitions, or it can be a series of many moves.

The steps available for use in our current language include: plie, tendu, degage, battement, eleve, retire
There are three positions: first, second, fifth.
There are three directions: front, side, back.

Two important rules to note:
1. Certain steps (plie, eleve, retire) do not take a direction, so these should have 'na' for its direction.
2. Certain steps cannot be done from second (tendu, degage, battement, retire).
Combos written in violation of these rules will be invalid.
To save a combo, rename the output file, Combo.mp4, to something different. That way, the next time a new program is run, the old video will not be replaced.

5. Formal Syntax

A complete BNF for programs written in our language.

```
<move> ::= {<step>, <pos>, <dir>}
<combo> ::= (<move>, <count>)+
<step> ::= Plie | Tendu | Degage | Battement | Eleve | Retire
<pos>::= First | Second | Fifth
<dir> ::= Front | Side | Back | NA
<count> ::= n in Z+
```

6. Semantics

Our primitive values are steps (strings), positions (strings), directions (strings), and count (ints). Each combo can be created from the given steps, positions, and directions in our BNF grammar.Our combining forms are moves that take a step, a position, a direction, and combos of either one sequence of moves (a move and a count) or a series of sequences. The output is a combo of move sequences in the order specific by the user. Evaluating a program will produce a video output that gives a visual representation of the dance combination.

Syntax	Abstract Syntax	Prec./Assoc.	Meaning
`<count>`	`int`	n/a	*count* is a primitive corresponding to the number of times a given move is repeated. We represent integers using the 32-bit F# integer data type (Int32).
`<dir>`	`Direction`	n/a	*dir* is a primitive corresponding to the direction of the body a given move is performed. It is a Direction type value from Front, Side, Back.
`<pos>`	`Position`	n/a	*pos* is a primitive corresponding to the position of the feet in which a given move is being performed. It is a Position type value from First, Second, Fifth.
`<step>`	`Step`	n/a	*step* is a primitive corresponding to the actual dance move being performed. It is a Step type value from Plie, Tendu, Degage, Battement, Eleve, Retire.
`<move>`	`Record-type Move of Step * Position * Direction`	n/a	*move* is a record type that describes a complete move, with the step, position, and direction.
`<combo>`	`Sequence of Move * Int / Series of Combo list`	n/a	*combo* is a combining form that keeps track of a list of move types and their corresponding counts. combo represents the series of moves that might make up an entire exercise at the barre.

7. Remaining Work

Further enhancements of our language includes:

- Adding more moves beyond the library of moves we currently have. This can include moves listed above in our Introduction that are typically seen in at barre, such as Ronds de Jambes, Frappés, and Fondus. We purposefully did not include these moves because they are harder to animate – we cannot fully capture the movement in two frames like we did with the moves currently in our language.
- More advanced animation with more than two frames.
- A way to change/specify arm positions. Our language currently only allows for arms in second position, as that is a very typical arm position used in barre combos.
- Showing the combination from two different angles (i.e. one from the front and one from the side).
- Extending our language to ballet combos in other contexts, such as floor or across the floor combos.
- Implementing a feature that allows users to pick a MIDI file of ballet music that moves are synced up with and which plays with the output video.

The addition of these enhancements would allow users to make more complex combos for class and get more advanced ballet combo animations.