# eMuse

Rapid Screenplay Prototyping Language

Mark Ayzenshtat
Elena Filatova
Kristina Holst
Vladislav Shchogolev

Table of Contents

# 1 Introduction

"*A picture is worth a thousand words.*"
-- Proverb

Stage and film productions, masterpieces and mediocrities alike, are born as ideas. The ability of writers and directors to develop these ideas and bring them to life depends greatly on their ability to mentally stage action and dialogue—the crucial mechanics that make or break creative works. However, throughout the history of theater and film, the development of these mechanics has been an entirely human process and, therefore, not a very consistent one. Productions can succeed on paper but fail miserably on stage. Sometimes the only reliable way to determine if a production succeeds is to stage it, often at a high cost.

Fortunately, directors and writers can use software tools to preview their productions before they are physically staged. However, although a wealth of animation languages and platforms exist, they may rely on mathematical models or assume substantial previous programming experience. As such, they are too complex to be used by all but the most tech-savvy creative people or those who can afford to hire technical teams.

The eMuse language and run-time environment offers a solution, allowing even laymen writers and directors to virtually stage their productions from start to finish. Instead of requiring script files written in a potentially complex third-party language, eMuse operates on a slightly doctored version of the source screenplay itself. Creative people can, for the first time, exercise complete control as they preview and refine their work.

## 1.1 Structure of an eMuse Script

The overall layout of a program written in eMuse resembles a screenplay. We tried to keep this layout as close as possible to the format in which free-form screenplays are written. Programs written in eMuse include the three following parts:

- identification
- definition
- exposition

The identifcation section contains information about the author(s) and title of the screenplay. The definition section contains all character, prop, and setting descriptions. The exposition section, meanwhile, contains the actual performance: what characters say and do. It is comprised of one or more scenes, each of which can carry the action to a new location.

## 1.2 Action and Animation

Stage actions are presented in the following format:

```
(verb [direct_object] [preposition indirect_object] [adverb])
```

Aside from the verb, any of these attributes can be omitted when appropriate, although a preposition may not appear without an indirect object and vice versa. For example, the following action line contains all possible attributes:

```
Hamlet: (puts sword on table slowly)
```

This line, however, omits unnecessary attributes:

```
Ophelia: (walks)
```

Although this format is slightly more restrictive than pure natural language, it allows us to unify the syntax of dialogue and stage actions, thereby avoiding the many complexities of natural language processing while still allowing more or less free-form English sentences.

eMuse presents the scriptwriter with a very high-level view of animation. Coordinates, angles, frames, and other lower-level animation constructs are completely buried beneath several layers of abstraction. The highest level, the one exposed to the script-writer, is the action interface outlined above.

Actions are not part of the eMuse language itself. Instead, they are implemented as plug-ins in a secondary language like Java and can be seamlessly referenced from any eMuse script. The eMuse run-time environment comes with a standard set of basic actions, analogous to the C standard library. A developer working on the scriptwriter's behalf can implement any custom actions needed to fully realize the artistic vision.

Screenplays are brought to life with an animation library that is part of the eMuse run-time. This library implements all action and dialogue using the Java 2D API, a well-documented standard.

eMuse is not the panacea of animation—it cannot serve as a complete replacement for more general animation packages. However, through its plug-in mechanism, it offers sufficient flexibility without sacrificing ease of use.

## 1.3 The Type System

There are three major types in eMuse: characters, props, and settings. These types are similar to structures in C; each consists of a particular set of attributes. Types are defined, appropriately enough, in the types block of the definition section.

```
Types

CharacterTypes
      castleGuard
            head: guardhead.jpg
            arm: guardarm.gif
            leg: guardleg.gif
            torso: guardtorso.gif
            body: tall, medium, male

PropTypes
      guardSword
            image: sword.jpg
            scale: medium
```

After types are defined, it becomes quite simple to introduce characters, props, and settings into the screenplay. For instance, to create two characters, Francisco and Bernardo, both of whom appear as castle guards, we would merely include the following character definition block:

```
Characters
      Francisco: castleGuard
      Bernardo: castleGuard
```

Similar definition blocks exist for props and settings. With this simple system, directors can quickly populate a rich scene with a variety of actors and props, and move on to more interesting things like action and dialogue.

Characters can move around the stage and speak...

```
Francisco: (enters left)
Bernardo: (enters right) Who's there?
```

...manipulate props...

```
Francisco: (takes sword)
```

...and act in concert with other characters in groups.

Settings, complete with background scenery and ambient music, are also easy to incorporate into the screenplay at the director's discretion.

## 1.4 Script to Stage

Here is an example to demonstrate the compilation and execution of an eMuse script:

The following line in the script:

```
Fransisco: Nay, answer me (takes sword)
```

Java-ish pseudo code:

```
character = objects.getCharacter (“Fransisco”);
prop = objects.getProp (“sword”);
character.perform(“speaks”, {string});
character.perform(“takes, {attributes}”);
```

Though eMuse has a small predefined set of supported actions (appears, disappears, drops, jumps, puts, shouts, sings, speaks, takes, turns, walks) we want to make this list easily extensible. The “speaks” action is the implicit in all dialogue and does not require explicit mentioning in the program text.

The eMuse compiler first translates the source script into equivalent Java commands (see above). Then, it invokes the Java compiler, which compiles the Java code against the rendering library and third-party action libraries, generating byte code that is ready to be executed.

## 1.5 Fundamental Advantages

### 1.5.1 Simplicity

eMuse is designed to make stage and film prototyping as simple and straightforward as possible. GUI libraries alone are insufficient for animation, and traditional animation platforms and packages assume a degree of technical sophistication that many creative people lack. eMuse takes most of the tedium and all of the uncertainty out of preparing complicated creative productions.

### 1.5.2 Ease of Implementation

eMuse programs closely resemble free-form screenplays, so translating any screenplay into an eMuse script is quick and easy.

### 1.5.3 Portability

eMuse scripts compile into Java language source code. Therefore, they can be presented on any platform with a suitable Java Virtual Machine (JVM).

### 1.5.4 Power

eMuse permits maximum expressiveness with minimal syntax. It intelligently chooses the correct defaults for various display parameters and other details not given in the original screenplay (e.g. character size, motion speed).

### 1.5.5 Extensibility

Any functionality not present in the standard action library can easily be written as a third party plug-in and incorporated into the screenplay.

# 2 Tutorial

Transforming a traditional script into an eMuse program is simple. Here we will take you through the process of turning an excerpt from the first scene of *Hamlet* into an executable eMuse file. Here is the original Shakespearean text:

> SCENE I. Elsinore. A platform before the castle.
>
> *Enter FRANCISCO and BERNARDO*
>
> BERNARDO Who's there?
> FRANCISCO Nay, answer me: stand, and unfold yourself.
> BERNARDO Long live the king!
> FRANCISCO Bernardo?
> BERNARDO He.
> FRANCISCO You come most carefully upon your hour.
> BERNARDO 'Tis now struck twelve; get thee to bed, Francisco.
> FRANCISCO For this relief much thanks: 'tis bitter cold, and I am sick at heart.
> BERNARDO Have you had quiet guard?
> FRANCISCO Not a mouse stirring.
> BERNARDO Well, good night. If you do meet Horatio and Marcellus, the rivals of my watch, bid them make haste.
> FRANCISCO I think I hear them. Stand, ho! Who's there?
>
> *FRANCISCO draws sword*

## 2.1 Writing the eMuse Script

The box below contains the final eMuse script on the left side, with comments on the right side to explain the different sections of code:

| | |
|---|---|
| `Title: Hamlet`<br>`Author: William Shakespeare` | Identify the script |
| `Types` | |
| `CharTypes`<br>`  castleGuard`<br>`  head: head.gif`<br>`  arm: arm.gif`<br>`  leg: leg.gif`<br>`  torso: torso.gif`<br>`  body: tall, medium, male` | Define the character type(s), specifying what image to use for the head and general characteristics about the body |
| `PropTypes`<br>`  guardSword`<br>`  image: sword.jpg`<br>`  scale: medium` | Define the prop type(s), specifying what image to use and what size it should be |
| `Settings`<br>`  castlePlatform`<br>`  image: castle.jpg` | Define the setting, specifying the background image |

| | |
|---|---|
| Characters<br>Francisco: castleGuard<br>Bernardo: castleGuard | Create the characters and give them each a character type |
| Props<br>sword: guardSword | Create the prop and give it a prop type |
| SCENE 1: castlePlatform | Mark the beginning of the scene and specify the setting |
| Bernardo: (appears on right) Who's there?<br>Francisco: (appears on left) Nay, answer me: stand, and unfold yourself.<br>Bernardo: Long live the king!<br>Francisco: Bernardo?<br>Bernardo: He.<br>Francisco: You come most carefully upon your hour.<br>Bernardo: (walks to Francisco quickly) 'Tis now struck twelve; get thee to bed, Francisco.<br>Francisco: For this relief much thanks: 'tis bitter cold, and I am sick at heart.<br>Bernardo: Have you had quiet guard?<br>Francisco: Not a mouse stirring.<br>Bernardo: Well, good night. If you do meet Horatio and Marcellus, the rivals of my watch, bid them make haste.<br>Francisco: I think I hear them. Stand, ho! Who's there? (takes sword) | Add all the dialog and action |

## 2.2 Compiling the Script

Once the eMuse script is complete, you run the following code at the command line to get a compiled Java class file:

```
$ emuse <script file>
```

This is actually a simple script that automates the entire compilation process, which consists of several steps.

## 2.3 Executing the Script

Finally, you can view the final output using:

```
$ java <compiled Java class file>
```

# 3 Language Reference Manual

## 3.1 Lexical Conventions

### 3.1.1 Character Set

The *character set* identifies all valid characters that may appear in an eMuse script. The following ASCII symbols comprise the complete character set:

- 26 lowercase Roman characters: `a-z`
- 26 uppercase Roman characters: `A-Z`
- 10 decimal numbers: `0-9`
- 32 graphic characters: `! @ # $ % ^ & * ( ) - _ = + ` ~ ' " : ; , . / | \ ? [ ] { } < >`
- 4 whitespace characters: `SP` (space), `HT` (horizontal tab), `LF` (line feed), `CR` (carriage return)

Below are a few rules that will be used throughout this manual.

*AlphaChar* → 'a'..'z' | 'A'..'Z'
*NumericChar* → '0'..'9'
*GraphicChar* → '!' | '@' | '#' | '$' | '%' | '^' | '&' | '*' | '(' | ')' | '-' | '_' | '=' | '+' | '`' | '~' | ''' | '"' | ':' | ';' | ',' | '.' | '/' | '|' | '\' | '?' | '[' | ']' | '{' | '}' | '<' | '>'

### 3.1.2 Tokens

A token is the basic element recognized by the compiler.

*Token* → *Identifier* | *Keyword* | *Punctuator*

### 3.1.3 Line Terminators

A line terminator signals the end of a line to the compiler.

*LineTerminator* → `CR` | `LF` | `CR LF`

### 3.1.4 Whitespace

eMuse is a whitespace-sensitive language. Whitespace is used to separate tokens (see §3.1.2), distinguish components in the definition section of a script (see §3.2.2), and identify dialog that continues for more than one line (see §3.2.3.2.2).

*Whitespace* → `SP` | `HT` | *LineTerminator*

### 3.1.5 Comments

Comments provide additional information to the programmer and are ignored by the compiler. eMuse comments begin with the pound sign (#) and continue until a line terminator is reached.

*Comment* → '#' (*CommentChar*)* *LineTerminator*

*CommentChar* → *AlphaChar* | *NumericChar* | *GraphicChar* | SP | HT

### 3.1.6 Identifiers

Identifiers must begin with a letter and may be followed by any combination of letters, numbers, and the underscore symbol. An identifier may not have the same spelling as a keyword.

*Identifier* → *AlphaChar* (*IdentifierChar*)*
*IdentifierChar* → *AlphaChar* | *NumericChar* | '_'

### 3.1.7 Keywords

The following list of keywords are reserved and may not be used as identifiers.

```
arm          head         scale
author       image        scene
body         left         short
center       leg          stage
characters   male         tall
chartypes    medium       thin
fat          props        title
female       proptypes    torso
groups       right        types
```

### 3.1.8 Punctuators

The following punctuators are used to organize text within a script.

```
, : ( )
```

Parentheses must be used in pairs.

## 3.2 Structure of an eMuse Script

Every eMuse script must contain definitions followed by an exposition. The definition section introduces all characters, props, and settings that will appear within the screenplay. The exposition section contains the body of the screenplay, including all dialogue and stage directions. An optional script identification section may precede these other two sections.

### 3.2.1 Identification

The identification block appears at the top of an eMuse script. The information included here will be displayed on a title screen when the program is executed.

#### 3.2.1.1 Title

The title of the script does not necessarily have any relation to the filename. To set the title, include the following code on its own line.

`Title:` *titleName*

#### 3.2.1.2 Author(s)

Any number of authors may be added to the script. For multiple authors, separate the names using the keyword `and`. The first and last names of an author may be more than one word, as in "von Neumann, John Louis".

`Author:` *lastName1*`,` *firstName1* `and` *lastName2*`,` *firstName2*

### 3.2.2 Definitions

All characters, props, and settings are created in the definition section. Individual characters and props are based on templates (character types and prop types, respectively) which must be defined first. Any template may be shared among multiple characters or props.

#### 3.2.2.1 Character Types

A character type provides a template for the appearance of a character. Any of the images for the specific body parts may be omitted, in which case that part of the body will be displayed as a stick figure. The `body` field contains a set of adjectives that are used create a customized body type. One adjective from each of the following lists may be included in the body field:

| Gender | Height | Build |
|---|---|---|
| male | short | thin |
| female | medium | medium |
| | tall | fat |

If no adjectives are specified for the body type, the default is "male, medium, medium".

```
CharacterTypes
    characterTypeName
        head: headFilename
        arm: armFilename
        leg: legFilename
        torso: torsoFilename
        body: attributes
```

#### 3.2.2.2 Characters

A character is created by specifying the character type to use for its body. Each new character definition must appear on a new line.

```
Characters
    characterName: characterTypeName
```

#### 3.2.2.3 Groups

Groups are a convenient way of referring to multiple characters, so that they may deliver lines or perform actions in unison. Any number of characters may be included within a group.

```
Groups
    groupName: characterName1, characterName2
```

#### 3.2.2.4 Prop Types

A prop type specifies an image to be displayed and a scale value, so that the image will be the appropriate size. The possible scale values are small, medium, and large, each of which is a keyword.

```
PropTypes
    propTypeName
        image: imageFileName
        scale: scaleValue
```

#### 3.2.2.5 Props

A prop is created by specifying a prop type.

```
Props
    propName: propTypeName
```

##### 3.2.2.5.1 Special Prop: `stage`

The stage is a special prop that is defined implicitly. It serves as the location where all action takes place, and it describes the area of the screen that is viewable when a script is executed. There are three regions of the stage: left, middle, and right. The stage provides a convenient way of specifying a character's or prop's location on screen (see §3.2.3.4).

#### 3.2.2.6 Settings

A setting specifies the image file that will be used in the background of a scene. There is no need to first create a "setting type", as there will only ever be one copy of a particular setting, unlike with characters and props.

```
Settings
    settingName
        image: imageFileName
```

### 3.2.3 Exposition

The exposition is the part of the script where dialogue and actions are specified. All characters, props, and settings used here need to have been defined in the previous section.

#### 3.2.3.1 Scene Heading

Scene headings mark the beginning of a new scene and specify the setting in which the scene will take place.

```
definitions
```

`SCENE 1:` *settingName*

*dialogue/action*

#### 3.2.3.2 Dialogue

Dialogue will appear as a speech bubble when the script is executed. Dialogue always has two components, the speaker and the speech, separated by a colon and whitespace.

*speakerName*: *speech*

##### 3.2.3.2.1 Speaker

The speaker may be either a character or a group. If the speaker is a group, all characters in the group will say the speech in unison. The speech bubble will appear directly above the speaker(s).

##### 3.2.3.2.2 Speech

If the speech exceeds one line, each subsequent line must be indented one tab from the left margin, indicating that it is a continuation of the previous line. Any characters (from the character set, see §3.1.1) may appear within speech except for parentheses and the pound sign, which are reserved for actions and comments, respectively.

#### 3.2.3.3 Actions

Actions are performed by embedding action calls in dialogue. The speech surrounding an action call is optional.

*speakerName*: *speech* (*verb directObj prepPhrase adverb*) *speech*

##### 3.2.3.3.1 Action Calls

There are components to an action call: verb, direct object, prepositional phrase, and adverb (see §3.2.3.3.3). The verb must always be written in third-person singular form (e.g. "walks", "jumps"). Action calls are enclosed in a pair of parentheses.

##### 3.2.3.3.2 Action Definitions

Actions are not part of the eMuse language itself, but are implemented in external libraries. In order to use a particular action within a script, the writer must have a copy of the library that contains the action definition. The eMuse standard action library included with the language provides Java implementations for several common verbs (see §3.3). Any verb may be implemented in a third-party library and used within an eMuse script.

##### 3.2.3.3.3 Arguments

Any of the last three components of an action call may be omitted. As an example, the proper format for expressing "John walks to Jane quickly" (which does not contain a direct object) is:

```
John: (walks to Jane quickly)
```

A direct object may be any character, group, or prop that has been defined for this script. Acceptable prepositional phrases and adverbs depend on the implementation provided for a given verb. Each verb supports a particular set of prepositions and adverbs, and the compiler will report an error if an unsupported preposition and/or adverb is used.

#### 3.2.3.4 Specifying Locations

Although characters and props are created in the definitions section of the script, they do not appear on-screen until they are explicitly given a position. Using "appears" or "enters" from the eMuse standard action library will bring a character or prop onto the screen. It is necessary to bring at least one character on screen at the beginning of a scene before any action can begin. As an example, to specify that the character John appears on the left side of the stage, the proper code would be:

```
John:  (appears on left)
```

Props are brought on-screen in the same way.

```
table:  (appears on center)
```

Both of the above examples show characters and props with an initial position relative to the stage.

If a prop is going to be connected to a particular character, such that whenever the character moves the prop goes with it, "takes" should be used instead. To indicate that John is wearing the hat and that the hat moves with him, the proper code would be:

```
John:  (takes hat)
```

Finally, to release the hat so that it no longer moves with John:

```
John:  (drops hat)
```

When a character or prop leaves the stage and should no longer be visible, "disappear" is used.

```
John:  (disappears)
```

By default, all characters and props still on stage at the end of a scene will "disappear".

## 3.3 eMuse Standard Action Library

The eMuse standard action library provides implementations for eleven common verbs:

```
appears       sings
disappears    speaks
drops         takes
jumps         turns
puts          walks
shouts
```

The verb "speaks" is implicitly used every time dialogue appears in a script. All other verbs must be called explicitly. Any verb not appearing on this list must be implemented by a third party before it can be used in a script.

# 4 Project Plan

We held frequent team meetings during the planning and specification stages of our project so that everyone could express their input and have a clear picture of our final goal. Once each team member had been assigned a particular task, development took place individually, and the code was shared using CVS. Each person was responsible for testing their own code before presenting it to the group to have others test it, as other people are often able to find mistakes you have overlooked. Our full testing plan is described in §6.

## 4.1 Team Responsibilities

Each team member had one primary responsibility, but frequently multiple teammates would collaborate in order to solve a particular problem. Below is each team member's primary task:

| | |
|---|---|
| Mark Ayzenshtat | code generator |
| Elena Filatova | lexer/parser |
| Kristina Holst | documentation |
| Vladislav Shchogolev | runtime library |

## 4.2 Project Timeline

The following is our list of completion dates for project milestones. See §4.5 for a more detailed log.

| | |
|---|---|
| Language details defined | Feb 8 |
| Whitepaper | Feb 16 |
| Grammar defined | Mar 21 |
| Language Reference Manual | Mar 23 |
| Compiler complete | Apr 16 |
| Renderer complete | May 4 |
| Final testing complete | May 10 |
| Final report | May 13 |

## 4.3 Software Development Environment

All compiler and renderer code was written in Java 2 SE 1.4. All lexer and parser code was written using ANTLR. We used CVS for version control and source code management. Our software IDE of choice was Eclipse, and we made heavy use of its built-in CVS client and the third-party ANTLR plug-in.

## 4.4 Style Guide

In any team project, it is important to develop coding standards that everyone follows to make cooperation and integration as easy as possible. We used the following guidelines but did not insist that everyone use the exact same conventions on minor points, as long as each person was consistent in the way they wrote their own code.

### 4.4.1 General Coding Style

Everyone writing Java code (Mark, Vlad, and Kristina) adhered to the programming principles outlined in Joshua Bloch's *Effective Java*. Bloch encourages practices such as

using singletons, favoring composition over inheritance, and prefering interfaces over abstract classes.

### 4.4.2 Commenting

Clearly commented code takes considerably less time to understand than uncommented code. We used Javadoc comments for all public Java methods and additional comments wherever appropriate.

### 4.4.3 Whitespace

Another key to making code more readable is the generous usage of whitespace. Logical blocks of code should always be separated by a blank line. Indentation should be used to make the code structure clear to anyone reading the code, as consolidated chunks of code can easily be misunderstood. At a quick glance, the following code may be interpreted incorrectly.

```
for (int i = 0; i < 100; array[i++] = 0);
i /= 2;
```

The purpose of this code becomes much more clear if written as:

```
for (int i = 0; i < 100; i++) {
     array[i] = 0;
}

i /= 2;
```

### 4.4.4 Naming Conventions

Classes, methods, and variables should have names that are descriptive but not excessively long. There is no reason to name a variable totalNumberOfPoints when numPoints will do. Member variables should have an "m" prefix to identify them, and constants should either have a "k" prefix or be written in all capital letters. In cases such as this, it doesn't not matter which convention you choose to use, as long as you are consistent throughout all of your code.

## 4.5 Project Log

This is a more detailed record of our progress throughout the semester.

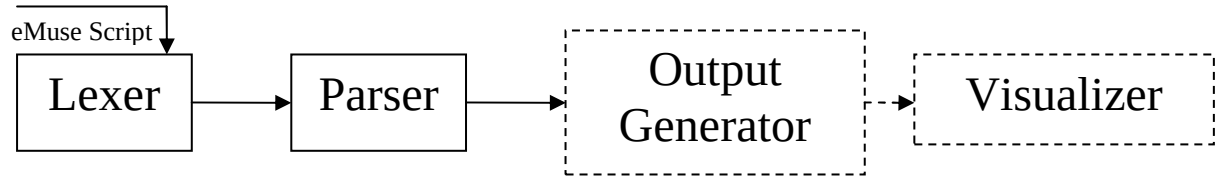
| | |
|---|---|
| Meeting to decide on language | Jan 25 |
| Meeting to decide on language | Jan 26 |
| Meeting to discuss lang details | Feb 1 |
| Meeting to assign responsibilities | Feb 4 |
| Language details defined | Feb 8 |
| Whitepaper – draft | Feb 13 |
| Whitepaper – final version | Feb 16 |
| Code overview developed | Mar 8 |
| Lexer written | Mar 14 |
| LRM – draft | Mar 19 |
| Grammar defined | Mar 21 |
| LRM – final version | Mar 23 |
| Lexer testing complete | Apr 3 |
| Semantic analyzer written | Apr 9 |
| Compiler back-end complete | Apr 16 |
| Basic renderer written | Apr 18 |
| Semantic analyzer testing complete | Apr 28 |
| Renderer complete | May 4 |
| Final testing complete | May 10 |
| Final report | May 13 |

# 5 Architectural Design

Below is a brief overview of the design of the eMuse language and platform. As mentioned in §4.1, Elena implemented the lexer and parser, Mark implemented the code generator (OutputGenerator in the diagram below), and Vladislav implemented the visualizer.

## 5.1 Block Diagram

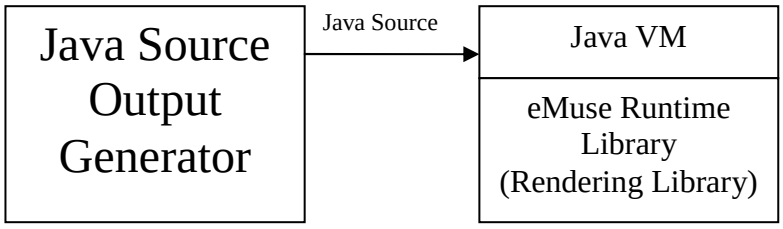
eMuse is not merely a language or compiler but a complete platform for prototyping screenplays. The pipeline through which a screenplay advances from text script to living animation involves several different components, each of which play a role in the overall architecture of eMuse. The four main components that compose the eMuse platform are the lexer, the parser, the output generator, and the visualizer. The block diagram that displays the relationships between these components is shown here:




The lexer and parser are fixed parts of the eMuse pipeline, while the output generator and visualizer are both pluggable. Given sufficient investment of labor, eMuse would be capable, at least theoretically, of generating and visualizing output in several interesting target formats. For example, a particular output generator could be authored that would produce Shockwave Flash files; its visualizer would be any modern web browser with the requisite plug-in. One could conceive of another output generator that produced text files that contained frames of ASCII art animation.

As a proof of concept, we have supplied default implementations of these components. Our output generator produces Java source that, when compiled and executed, animates the screenplay using our visualizer: a two-dimensional scene graph based rendering library.

This default implementation can be substituted into the second half of the above block diagram as follows:



## 5.2 Interfaces

The interface between the lexer and the parser was already well-defined, since we used ANTLR to generate both components.

The interface between the top half and bottom half of the compiler (i.e. the interface between the parser and output generator) is defined by the following `OutputGenerator` methods:

```
/**
 * Begins generating output.  This method must be called to initialize this
 * output generator before any <code>outputXXX()</code> methods are called.
 */
public void startOutput();

/**
 * Outputs the title of the screenplay.
 *
 * @param iTitle the title
 */
public void outputTitle(String iTitle);

/**
 * Outputs an author of the screenplay.  This method may be called more than
 * once if the screenplay has more than one author.
 *
 * @param iAuthor an author
 */
public void outputAuthor(String iAuthor);

/**
 * Begins to output a CharType.
 *
 * @param iLabel the label used to identify the CharType
 */
public void outputCharType(String iLabel);

/**
 * Outputs a CharType head parameter.
 *
 * @param iImageFile the image file
 */
public void outputCharTypeHead(String iImageFile);

/**
 * Outputs a CharType arm parameter.
 *
 * @param iImageFile the image file
 */
public void outputCharTypeArm(String iImageFile);

/**
 * Outputs a CharType leg parameter.
 *
 * @param iImageFile the image file
 */
public void outputCharTypeLeg(String iImageFile);
```

```
/**
 * Outputs a CharType torso parameter.
 *
 * @param iImageFile the image file
 */
public void outputCharTypeTorso(String iImageFile);

/**
 * Outputs the CharType body parameters.
 */
public void outputCharTypeBody(String iGender, String iHeight,
       String iBuild);

/**
 * Begins to output a PropType.
 *
 * @param iLabel the label used to identify the PropType
 */
public void outputPropType(String iLabel);

/**
 * Outputs a PropType image parameter.
 *
 * @param iImageFile the image file
 */
public void outputPropTypeImage(String iImageFile);

/**
 * Outputs a PropType scale parameter.
 *
 * @param iImageFile the scale
 */
public void outputPropTypeScale(String iScale);

/**
 * Outputs a setting.
 *
 * @param iLabel the label used to identify the setting
 */
public void outputSetting(String iLabel);

/**
 * Outputs the image field of a setting.
 *
 * @param iImageFile the image file
 */
public void outputSettingImage(String iImageFile);

/**
 * Outputs the sound field of a setting.
 *
 * @param iSoundFile the sound file
 */
public void outputSettingSound(String iSoundFile);

/**
 * Outputs a character definition.
 *
 * @param iName the character's name
 * @param iCharType the CharType
 */
public void outputCharacter(String iName, String iCharType);
```

```
/**
 * Outputs a prop definition.
 *
 * @param iName the character's name
 * @param iPropType the CharType
 */
public void outputProp(String iName, String iPropType);

/**
 * Outputs a group definition.
 *
 * @param iName the name of the group
 */
public void outputGroup(String iName);

/**
 * Outputs a group member.  This method may be called more than
 * once if the group has more than one member.
 * @param iName the group member
 */
public void outputGroupMember(String iName);

/**
 * Outputs a new scene.
 *
 * @param iSetting the setting of the scene
 */
public void outputScene(String iSetting);

/**
 * Outputs an action.
 *
 * @param iSubject the subject
 * @param iVerb the verb
 * @param iDirObj the direct object
 * @param iPrep the preposition
 * @param iIndirObj the indirect object
 * @param iAdverb the adverb
 */
public void outputAction(String iSubject, String iVerb, String iDirObj,
       String iPrep, String iIndirObj, String iAdverb);

/**
 * Completes output generation.  This method must be called to finalize
 * output generation after all <code>outputXXX()</code> methods have been
 * called.
 */
public void finishOutput() throws EmuseException;
```

The interface between the output generator and visualizer depends on the output generator implementation. In our default implementation, it consisted of animation library methods which the compiled screenplay calls to render itself. These methods are too numerous to list here.

# 6 Test Plan

We performed three major types of testing—module, incremental, and regression. We did not automate this testing process, due to the nature of our project. Whereas other languages may be tested by comparing actual textual output versus expected textual output, the only way to see if an eMuse script runs correctly is to actually view the animation and watch for aberrations.

## 6.1 Generated Source Code Sample

This is an example of the Java source code generated from an eMuse script. First, the script:

```
Title: Hamlet for Dummies
Author: William Shakespeare, Fake McFakerton, Kristina Grace Holst
Types
CharTypes
      castleGuard
            head: sedwards.gif
            arm: bush.gif
            leg: bush.gif
            torso: bush.gif
            body: tall, medium, male
      castleGuard2
            head: bush.gif
            arm: bush.gif
            leg: bush.gif
            torso: bush.gif
            body: short, fat, female
PropTypes
      guardSword
            image: money.jpg
            scale: medium
Settings
      castlePlatform
            image: meadow.jpg
            sound: chimes.wav

Characters
      Francisco: castleGuard
      Bernardo: castleGuard2
Props
      franSword: guardSword
      bernSword: guardSword
Groups
      guards: Francisco, Bernardo

SCENE 1: castlePlatform
      Francisco:(appears on right) Yo!
      Bernardo:(appears on left) Who's there?
      Francisco: Nay, answer me stand, and unfold yourself.
      Francisco: You come most carefully upon your hour.
      Bernardo:  (walks to Francisco quickly) You know, most of our imports
      come from outside the country.
      Francisco: For this relief much thanks; tis bitter cold, And I am sick
      at heart.
      Bernardo:  Have you had quiet guard?
      Francisco: Not a mouse stirring.
```

```
      Bernardo: Well, good night. If you do meet Horatio and Marcellus, The
      rivals of my watch, bid them make haste.
      Francisco:  I think I hear them. Stand, ho! Who s there?
```

And now the corresponding Java code:

```
/*
 * This source file was automatically generated by eMuse.
 * Do not modify.
 */

import java.util.*;
import emuse.*;
import emuse.runtime.actions.*;
import emuse.runtime.anim.*;
import emuse.runtime.model.*;
import emuse.runtime.model.Character;

/**
 * Generated screenplay class.
 */
public final class Test1ScreenplayOutput {
  private Screenplay mScreenplay;
  private CharType ct1;
  private CharType ct0;
  private PropType pt0;
  private Setting set0;
  private Character c0;
  private Character c1;
  private Prop p0;
  private Prop p1;
  private Group g0;

  public static void main(String[] args) {
    new Test1ScreenplayOutput();
  }

  /**
   * Creates the screenplay state and hands it off to a scene manager.
   */
  public Test1ScreenplayOutput() {
    try {
      mScreenplay = new Screenplay();

      setTitleAndAuthors();

      defineCharTypes();
      definePropTypes();
      defineSettings();
      defineCharacters();
      defineProps();
      defineGroups();
      defineScenes();

      AnimationFrame animator = AnimationFrame.forScreenplay(mScreenplay);
      animator.startAnimation();
```

```
  } catch (EmuseException ex) {
    ex.printStackTrace();
  }
}

private void setTitleAndAuthors() throws EmuseException {
  mScreenplay.setTitle("Hamlet for Dummies ");
  mScreenplay.setAuthors(new String[] {"William Shakespeare", "Fake
   McFakerton", "Kristina GraceHolst"});
}

private void defineCharTypes() throws EmuseException {
  ResourceManager rm = ResourceManager.getInstance();

  ct1 = new CharType();
  ct1.setHeadImage(rm.getImage("bush.gif"));
  ct1.setArmImage(rm.getImage("bush.gif"));
  ct1.setLegImage(rm.getImage("bush.gif"));
  ct1.setTorsoImage(rm.getImage("bush.gif"));
  ct1.setGender(Gender.valueOf("female"));
  ct1.setHeight(Height.valueOf("short"));
  ct1.setBuild(Build.valueOf("fat"));

  ct0 = new CharType();
  ct0.setHeadImage(rm.getImage("sedwards.gif"));
  ct0.setArmImage(rm.getImage("bush.gif"));
  ct0.setLegImage(rm.getImage("bush.gif"));
  ct0.setTorsoImage(rm.getImage("bush.gif"));
  ct0.setGender(Gender.valueOf("male"));
  ct0.setHeight(Height.valueOf("tall"));
  ct0.setBuild(Build.valueOf("medium"));

  mScreenplay.setCharacterTypes(new CharType[] {ct1, ct0});
}

private void definePropTypes() throws EmuseException {
  ResourceManager rm = ResourceManager.getInstance();

  pt0 = new PropType();
  pt0.setAppearance(rm.getImage("money.jpg"));
  pt0.setScale(Scale.valueOf("medium"));

  mScreenplay.setPropTypes(new PropType[] {pt0});
}

private void defineSettings() throws EmuseException {
  ResourceManager rm = ResourceManager.getInstance();

  set0 = new Setting();
  set0.setImage(rm.getImage("meadow.jpg"));
  set0.setSound(rm.getSound("chimes.wav"));

  mScreenplay.setSettings(new Setting[] {set0});
}

private void defineCharacters() throws EmuseException {
  c0 = new Character();
```

```
  c0.setName("Francisco");
  c0.setCharType(ct0);

  c1 = new Character();
  c1.setName("Bernardo");
  c1.setCharType(ct1);

  mScreenplay.setCharacters(new Character[] {c0, c1});
}

private void defineProps() throws EmuseException {
  p0 = new Prop();
  p0.setName("franSword");
  p0.setPropType(pt0);

  p1 = new Prop();
  p1.setName("bernSword");
  p1.setPropType(pt0);

  mScreenplay.setProps(new Prop[] {p0, p1});
}

private void defineGroups() throws EmuseException {
  g0 = new Group();
  g0.setName("guards");
  g0.setMembers(new StageObject[] {c0, c1});

  mScreenplay.setGroups(new Group[] {g0});
}

private void defineScenes() throws EmuseException {
  mScreenplay.addScene(createScene0());
}

private Scene createScene0() throws EmuseException {
  Scene s = new Scene();
  s.setSetting(set0);

  Action a0 = new AppearsAction();
  a0.setRunOrder(0);
  a0.initialize(c0, null, Preposition.valueOf("on"), Stage.RIGHT, null);
  s.addAction(a0);

  Action a1 = new SpeaksAction();
  a1.setRunOrder(1);
  a1.initialize(c0, "Yo! ", null, null, null);
  s.addAction(a1);

  Action a2 = new AppearsAction();
  a2.setRunOrder(2);
  a2.initialize(c1, null, Preposition.valueOf("on"), Stage.LEFT, null);
  s.addAction(a2);

  Action a3 = new SpeaksAction();
  a3.setRunOrder(3);
  a3.initialize(c1, "Who 's there? ", null, null, null);
  s.addAction(a3);
```

```
    Action a4 = new SpeaksAction();
    a4.setRunOrder(4);
    a4.initialize(c0, "Nay, answer me stand, and unfold yourself. ", null,
null, null);
    s.addAction(a4);

    Action a5 = new SpeaksAction();
    a5.setRunOrder(5);
    a5.initialize(c0, "You come most carefully upon your hour. ", null, null,
null);
    s.addAction(a5);

    Action a6 = new WalksAction();
    a6.setRunOrder(6);
    a6.initialize(c1, null, Preposition.valueOf("to"), c0,
Adverb.valueOf("quickly"));
    s.addAction(a6);

    Action a7 = new SpeaksAction();
    a7.setRunOrder(7);
    a7.initialize(c1, "You know, most of our imports come from outside the
country. ", null, null, null);
    s.addAction(a7);

    Action a8 = new SpeaksAction();
    a8.setRunOrder(8);
    a8.initialize(c0, "For this relief much thanks; tis bitter cold, And I am
sick at heart. ", null, null, null);
    s.addAction(a8);

    Action a9 = new SpeaksAction();
    a9.setRunOrder(9);
    a9.initialize(c1, "Have you had quiet guard? ", null, null, null);
    s.addAction(a9);

    Action a10 = new SpeaksAction();
    a10.setRunOrder(10);
    a10.initialize(c0, "Not a mouse stirring. ", null, null, null);
    s.addAction(a10);

    Action a11 = new SpeaksAction();
    a11.setRunOrder(11);
    a11.initialize(c1, "Well, good night. If you do meet Horatio and
Marcellus, The rivals of my watch, bid them make haste. ", null, null, null);
    s.addAction(a11);

    Action a12 = new SpeaksAction();
    a12.setRunOrder(12);
    a12.initialize(c0, "I think I hear them. Stand, ho! Who s there? ", null,
null, null);
    s.addAction(a12);

    s.sealActions();
    return s;
  }
}
```

## 6.2 Module Testing

Each team member was responsible for testing his/her code in isolation before letting it interact with other components. Testing was done as the programmer saw fit, typically using a combination of black-box and glass-box testing methods. As an example, when testing the lexer and parser, it was important to guarantee that all rules of our grammar were actually enforced (e.g. only Characters may be members of Groups, all Characters have unique names, etc.).

## 6.3 Incremental Testing

Whenever a new component had passed its module testing, it had to be integrated with the rest of the system. The addition of a new component required incremental testing to guarantee that all interactions between components had only the desired effects.

## 6.4 Regression Testing

Our regression testing involved verifying that previously-fixed bugs did not re-appear after modifying the code. We kept a running list of major problems to test for, and we performed regression testing after any significant change.

# 7 Lessons Learned

## 7.1 Individual Responses

### 7.1.1 Mark's Lesson

"Don't try to implement everything at once. It would have helped my own efforts tremendously if I had identified a minimal set of features early on, implemented those features first, and then added functionality to them incrementally."

### 7.1.2 Elena's Lesson

"In some cases (our language as an example), it is possible and even useful to avoid creating a separate semantic analyzer. It makes more sense to have it run parallel to (or even as part of) the syntactic analysis."

### 7.1.3 Kristina's Lesson

"The final report was made significantly easier by the whitepaper and LRM deadlines along the way. In a future project, I would try to get more of the final product done ahead of time, since there is no reason to leave it until the last minute (except final edits)."

### 7.1.4 Vladislav's Lesson

"We should have explored all of our animation options before deciding on Java 2D. If we had investigated OpenGL or Flash, we may have found that one of those would have been easier to use."

## 7.2 Advice for Future Teams

Our only advice for future teams is to start early and plan ahead. No project ever goes exactly as planned, so you must leave yourself enough time to handle unexpected problems. The beginning stages of this project are not as work intensive, so use this time to think about the fine details of your plan and work on a preliminary code outline, to make sure your project is feasible.

# Appendix: Code Listing

```
// ------------------------------------------------------------
// emuse.g
// Authors: Elena Filatova, Mark Ayzenshtat
// ------------------------------------------------------------

header {
package emuse.compiler.antlr;

import java.util.*;
import emuse.*;
import emuse.compiler.*;
}

class P extends Parser;
options {
      k = 2;
}
tokens {
  ColonName;
}
{
      private OutputGenerator og;
      private String mSourceFilePath = null;

      Map charType = new HashMap();
      Map propType = new HashMap();
      Map allVars = new HashMap();
      Map settings = new HashMap();
      Map groups   = new HashMap();

      { allVars.put("stage", null); }
      { allVars.put("right", null); }
      { allVars.put("left", null); }
      { allVars.put("center", null); }

      private void outputAction(String iStr) {
            String[] s = StringUtils.tokenizeActionString(iStr);
            og.outputAction(s[0], s[1], s[2], s[3], s[4], s[5]);
      }

      public void setSourceFilePath(String iPath) {
            mSourceFilePath = iPath;
      }
}
startRule
    :
    {
            og = new JavaOutputGenerator(mSourceFilePath);
      og.startOutput();
    }
      { String authors=null; String authors_add=null;}  // init-action
      { String title=null; String title_add=null;}  // init-action
     ("Title:"  title=words (title_add=words { title=title+title_add; })* {
      og.outputTitle(title);
     } )?
     ("Author:"  authors=authorName {
      og.outputAuthor(authors);
     } (c:COMMA authors=authorName {
      og.outputAuthor(authors);
     } )* )?
```

```
      (defSection)?
      (scene)+
       EOF!
{
       try {
              og.finishOutput();
       } catch (EmuseException ex) {
              ex.printStackTrace();
              throw new RecognitionException("Error: " + ex.getMessage());
       }
}
    ;

defSection :
       defTypes
       defVars
       ;
//typeCharBoth allows either just Characters types definitions or both cahracters and
Props types definition
//typeProp allows only Props types definitions
defTypes :
       "Types"
       (typeCharBoth | typeProp) (setting)?
       ;
typeCharBoth :
       "CharTypes" (instanceTypeChar)+  (typeProp)?
       ;
instanceTypeChar :
       t:NAME {
              og.outputCharType(t.getText());
       } {
    if (charType.containsKey(t.getText())) {
        System.err.println("Double declaration of character type "+t.getText());
        System.exit(0);
    } else {
        charType.put(t.getText(), new Object());
    }
  } elementsChar
       ;
elementsChar
       { String fileName=null; String paramBody=null;}
       :
       ("head:" fileName=nameFile {
              og.outputCharTypeHead(fileName);
       })?
       ("arm:" fileName=nameFile  {
              og.outputCharTypeArm(fileName);
       })?
       ("leg:" fileName=nameFile  {
              og.outputCharTypeLeg(fileName);
       })?
       ("torso:" fileName=nameFile {
              og.outputCharTypeTorso(fileName);
       })?
       "body:" paramBody=bodyParams {
              String[] s = StringUtils.tokenizeCharTypeBody(paramBody);
              og.outputCharTypeBody(s[2], s[0], s[1]);
       }
       ;
typeProp : "PropTypes" (instanceTypeProp)+
       ;
instanceTypeProp :
       t:NAME {
```

```
              og.outputPropType(t.getText());
       } {
    if (propType.containsKey(t.getText())) {
        System.err.println("Double declaration for prop type "+t.getText());
        System.exit(0);
    } else {
        propType.put(t.getText(), new Object());
    }
  } elementsProp
       ;

elementsProp
       { String fileName=null; String scaleParam = null;}
       :
       "image:" fileName=nameFile {
              og.outputPropTypeImage(fileName);
       }
       "scale:" scaleParam=scaleParams {
              og.outputPropTypeScale(scaleParam);
       }
       ;
setting
       { String fileName=null; }
       :
       "Settings" (n1:NAME {
              og.outputSetting(n1.getText());
       } {
    if (settings.containsKey(n1.getText())) {
        System.err.println("Double declaration for setting "+n1.getText());
        System.exit(0);
    } else {
        settings.put(n1.getText(), new Object());
    }
  }
              "image:" fileName=nameFile {
                     og.outputSettingImage(fileName);
              }
              ("sound:" fileName=nameFile {
                     og.outputSettingSound(fileName);
              })?)+
       ;
defVars : (varCharBoth | varProp) (varGroup)?;

varCharBoth
       : "Characters"  (instanceVarChar)+ (varProp)?
       ;
instanceVarChar
{String charVar = ""; int i = 0; String charVar_add = "";}
       : cv:ColonName {charVar = cv.getText(); i=charVar.length();
charVar=charVar.substring(0,i-1); charVar_add=charVar;}
       n:NAME  {
              og.outputCharacter(charVar, n.getText());
       } {
           if (charType.containsKey(n.getText())) {
               if (allVars.containsKey(charVar_add)) {
                 System.err.println("Double declaration of character "+charVar_add+"of
type "+n.getText());
                 System.exit(0);
               } else {
                 allVars.put(charVar_add, new Object());
               }
           } else {
```

```
               System.err.println("Character "+charVar_add+" cannot be defined:
character type "+n.getText()+" is not defined");
               System.exit(0);
            }
       }
       ;

varProp
       : "Props" (instanceVarProp)+
       ;
instanceVarProp
{String propVar = ""; int i = 0; String propVar_add = "";}
       : cv:ColonName {propVar = cv.getText(); i=propVar.length();
propVar=propVar.substring(0,i-1); propVar_add = propVar;}
       n:NAME {
              og.outputProp(propVar, n.getText());
       } {
              if (propType.containsKey(n.getText())) {
                  if (allVars.containsKey(propVar_add)) {
                     System.err.println("Double declaration of prop "+propVar_add+" of
type "+n.getText());
                     System.exit(0);
                  } else {
                     allVars.put(propVar_add, new Object());
                  }
              } else {
                  System.err.println("Prop "+propVar_add+" cannot be defined: prop type
"+n.getText()+" is not defined");
                  System.exit(0);
              }
    }
       ;
varGroup
       : "Groups"  (instanceVarGroup)+
       ;
instanceVarGroup
{String groupVar = ""; int i = 0; Map currGroup = new HashMap(); }
       : cv:ColonName {
              groupVar = cv.getText();
              i=groupVar.length();
              groupVar=groupVar.substring(0,i-1);

              og.outputGroup(groupVar);
       }
       n:NAME {
              og.outputGroupMember(n.getText());
       }  {
    if (groups.containsKey(groupVar)) {
          System.err.println("Double declaraion of group "+groupVar);
          System.exit(0);
    } else {
          groups.put(groupVar, new Object());
          if (allVars.containsKey(n.getText())){
            if (currGroup.containsKey(n.getText())){
              System.err.println("Double instance of character "+n.getText()+" in
group "+groupVar);
              System.exit(0);
            }else{
              currGroup.put(n.getText(), new Object());
            }
          } else {
            System.err.println("Character "+n.getText()+" is not defined and thus
cannot be inserted into the group "+groupVar);
```

```
            System.exit(0);
          }
    }
  }
       (COMMA n1:NAME {
              og.outputGroupMember(n1.getText());
    if (allVars.containsKey(n1.getText())){
      if (currGroup.containsKey(n1.getText())){
        System.err.println("Double instance of character "+n1.getText()+" in group
"+groupVar);
        System.exit(0);
      }else{
        currGroup.put(n1.getText(), new Object());
      }
    } else {
      System.err.println("Character "+n1.getText()+" is not defined and thus cannot be
inserted into the group "+groupVar);
      System.exit(0);
    }
       })*
       ;
scene : "SCENE" NUMBER COLON n:NAME {
       og.outputScene(n.getText());
}  (stageAction)*
       ;

stageAction
{String actVar=""; String speech=""; int i=0; String firstChar=""; String lastChar =
""; String charName = null; }
  : t:ColonName
    {
      charName=t.getText();
      i=charName.length();
      charName=charName.substring(0,i-1);

      if (!(allVars.containsKey(charName))) {
        System.err.println("Character "+charName+" is not defined");
        System.exit(0);
      }
    }
    (actVar=charAct {firstChar=actVar.substring(0,1);
                     i = actVar.length();
                     lastChar=actVar.substring(i-1);
                     if (firstChar.equals("("))
                            {
                            if (speech.length()>0)
                                   {
                                          speech =
charName+",speaks,\""+speech+"\",null,null,null";
                                                 outputAction(speech);
                                          speech = "";
                                   }
                            actVar=      actVar.substring(1,i-1);
                            actVar = charName+","+actVar;
                            outputAction(actVar);
                            }
                     else
                            speech=speech+actVar;
                     })+
    {if (speech.length()>0)
       {
              speech = charName+",speaks,\""+speech+"\",null,null,null";
```

```
              outputAction(speech);
        }
    }
  ;


charAct returns [String charAct]
{ charAct="";}  // init-action

  :
    charAct=action
  | charAct=words
  ;


words returns [String wordsString]
//{ wordsString = new String(); puncMark = new String();}   // init-action
{ wordsString=""; String puncMark=null;}

  : (t2:APOST { wordsString=wordsString+t2.getText(); })?
    t:NAME { wordsString=wordsString+t.getText(); }
    (puncMark = punc { wordsString=wordsString+puncMark; } )?
    { wordsString=wordsString+" "; }
  ;

authorName  returns [String author]
{ author = new String(); }   // init-action
       :
       n1:NAME { author=n1.getText()+" "; }
       (n2:NAME  { author=author+n2.getText(); }
       (d:DASH { author=author+d.getText(); } n3:NAME { author=author+n3.getText(); }
       |p:PERIOD { author=author+p.getText(); }
       )? { author=author+" "; })*
       ;

punc returns [String puncString]
{ puncString = new String(); }   // init-action
       :
       t1:PERIOD {puncString=t1.getText();}
       | t2:COMMA  {puncString=t2.getText();}
       | t3:QUESTION {puncString=t3.getText();}
       | t4:EXCLAM {puncString=t4.getText();}
       | t5:SEMICOLON {puncString=t5.getText();}
       ;

action returns [String action]
{action = ""; String addVar = null;}
       : LEFTPAR {action = "(";}
       (
              addVar= speechVerb {action = action+addVar;} addVar=speechAction {action
= action+","+addVar+",null,null";}
              |
              addVar = verb  {action = action+addVar;}
              (addVar = actionIND {action = action+",null,"+addVar;}
              | addVar = actionDO {action = action+","+addVar;}
              | /* nothing */
              )
       )
       (addVar=adverb {action = action+","+addVar;}
       |/* nothing */{action = action+",null";}
       )
       RIGHTPAR {action = action+")";}
       ;
```

```
speechAction returns [String directSpeech]
{directSpeech = null; String directSpeech_add = null;}
       : QUOTE directSpeech=words (directSpeech_add=words {directSpeech =
directSpeech+directSpeech_add;})* QUOTE {directSpeech = "\""+directSpeech+"\"";}
       ;


actionIND returns [String actionIND]
{actionIND=""; String p = null;}
       : p=prep n:NAME {
    actionIND=p+","+n.getText();
    if (!(allVars.containsKey(n.getText()))){
      System.err.println("Variable "+n.getText()+" is not defined");
      System.exit(0);
    }
  }
       ;

actionDO returns [String actionDO]
{actionDO=""; String p = null;}
       : n1:NAME {
    actionDO=n1.getText();

    if (!(allVars.containsKey(n1.getText()))) {
      System.err.println("Variable "+n1.getText()+" is not defined");
      System.exit(0);
    }
  } (p=prep n2:NAME {
    actionDO=actionDO+","+p+","+n2.getText();

    if (!(allVars.containsKey(n2.getText()))) {
      System.err.println("Variable "+n2.getText()+" is not defined");
      System.exit(0);
    }
  }
                                    | /* nothing */ {actionDO=actionDO+",null,null";})
       ;


verb returns [String verb]
{verb = null;}
       :
       t1:"speaks" {verb=t1.getText();}
       | t2:"takes" {verb=t2.getText();}
       | t3:"puts" {verb=t3.getText();}
       | t4:"walks" {verb=t4.getText();}
       | t5:"jumps" {verb=t5.getText();}
       | t6:"turns" {verb=t6.getText();}
       | t7:"appears" {verb=t7.getText();}
       | t8:"disappears" {verb=t8.getText();}
//     | t9:"sings" {verb=t9.getText();}
//     | t10:"shouts" {verb=t10.getText();}
       ;

speechVerb returns [String speechVerb]
{speechVerb = null;}
              :
        t9:"sings" {speechVerb=t9.getText();}
       | t10:"shouts" {speechVerb=t10.getText();}
       ;
```

```
prep returns [String prep]
{prep = null;}
       : "on" {prep="on";}| "in" {prep="in";}| "to" {prep="to";}| "center"
{prep="center";}
       ;

adverb returns [String adverb]
{adverb = null;}
       : "quickly" {adverb = "quickly";}| "loudly" {adverb = "loudly";}| "quitly"
{adverb = "quitly";}
       ;


nameFile returns [String fileString]
{fileString = "";}
       :
       (s1:SLASH { fileString=fileString+s1.getText(); })?
       (n1:NAME s2:SLASH  { fileString=fileString+n1.getText()+s2.getText(); })*
       n2:NAME { fileString=fileString+n2.getText();}
       (p:PERIOD n3:NAME { fileString=fileString+p.getText()+n3.getText(); })?
       ;



bodyParams returns [String bodyParams]
{bodyParams = null; String bodyParams_add = null;}
       :
       bodyParams=height COMMA
       bodyParams_add=build COMMA { bodyParams=bodyParams+","+bodyParams_add+","; }
       bodyParams_add=sex { bodyParams=bodyParams+bodyParams_add; }
       ;


//bodyPart returns [String bodyPart]
//{bodyPart = null}
//     : bodyPart = "head" | bodyPart = "arm" | bodyPart = "leg" | bodyPart = "torso"
//     ;

scaleParams returns [String scale]
{scale = null;}
       : "large" {scale = "large";}| "medium" {scale = "medium";}| "small" {scale =
"small";}
       ;

height returns [String height]
{height = null;}
       :
       "tall" {height="tall";}| "medium" {height="medium";}| "short" {height="short";}
       ;

build returns [String build]
{build = null;}
       :
       "fat" {build="fat";}| "medium" {build="medium";}| "thin" {build="thin";}
       ;

sex returns [String sex]
{sex = null;}
       :
       "male" {sex = "male";} | "female" {sex = "female";}
       ;
```

```
class L extends Lexer;

// one-or-more letters followed by a newline

NUMBER: ('0' .. '9')+ ;

NAME
  : ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9')*
  ( ':' { $setType(ColonName); }
  | /* nothing */
  )
  ;


WS : (' '|'\f'|'\t')+ {$setType(Token.SKIP);};

NL : ('\n'| '\r') {
       newline();
       $setType(Token.SKIP);
};

//IND : '\t' ;
//{$setType(Token.SKIP);};

COLON: ':';
PERIOD: '.';
COMMA: ',';
QUESTION: '?';
RIGHTPAR: ')';
LEFTPAR: '(';
QUOTE: '"';
EXCLAM: '!';
SEMICOLON: ';';
APOST: "'";
SLASH: '/';
DASH: '-';
```

```
package emuse.runtime.actions;

import java.util.Comparator;
import emuse.runtime.model.Action;
import emuse.runtime.model.Adverb;
import emuse.runtime.model.Preposition;
import emuse.runtime.model.StageObject;

/**
 * @author Vlad Shchogolev
 * @author Mark Ayzenshtat
 *
 * AbstractAction.java
 * Apr 8, 2003
 */
public abstract class AbstractAction implements Action {
	boolean firstTime = true;
	double elapsed = 0;

	public static final Comparator kRunOrderComparator =
		new RunOrderComparator();

	protected int mRunOrder;

	protected AbstractAction() {
		mRunOrder = 0;
	}

	/**
	 * @see emuse.runtime.model.Action#initialize(StageObject, Preposition,
StageObject, Adverb)
	 */
	public void initialize(
		StageObject iSubject,
		Object iDirectObject,
		Preposition iPrep,
		StageObject iIndirectObject,
		Adverb iAdverb) {
		// do nothing
	}

	protected void onFirstStep() {
	}

	protected void onStep() {
	}

	/**
	 * @see emuse.runtime.model.Action#step(int)
	 */
	public boolean step(double secs) {
		this.elapsed += secs;

		if (firstTime) {
			onFirstStep();
			firstTime = false;
		}

		onStep();
		return isDone();
	}

	/**
```

```
	 * @see emuse.runtime.model.Action#isDone()
	 */
	public boolean isDone() {
		return true;
	}

	/**
	 * Returns the "run order" number of this action.  The run order dictates
	 * when the action will be run by the scene manager.  A group of actions
	 * that should be run concurrently should be assigned the same run order.
	 *
	 * @return the "run order" number of this action
	 */
	public int getRunOrder() {
		return mRunOrder;
	}

	public void setRunOrder(int iOrder) {
		mRunOrder = iOrder;
	}

	private static class RunOrderComparator implements Comparator {
		public int compare(Object iO1, Object iO2) {
			Action a1 = (Action) iO1;
			Action a2 = (Action) iO2;

			return a1.getRunOrder() - a2.getRunOrder();
		}
	}
}
```

```
package emuse.compiler.javamodel;

/**
 * A typesafe enumeration for Java access modifiers.  Typesafe enum elements
 * can be compared using the <code>==</code> operator.
 *
 * @author Mark Ayzenshtat
 */
public class AccessModifier {
	/** The <code>public</code> access modifier. */
	public static final AccessModifier kPublic =
		new AccessModifier("public");

	/** The <code>private</code> access modifier. */
	public static final AccessModifier kPrivate =
		new AccessModifier("private");

	/** The <code>protected</code> access modifier. */
	public static final AccessModifier kProtected =
		new AccessModifier("protected");

	/** The default/implied/package/friend access modifier. */
	public static final AccessModifier kDefault =
		new AccessModifier("");

	private final String mKeyword;

	private AccessModifier(String iKeyword) {
		mKeyword = iKeyword;
	}

	/**
	 * Returns the <code>String</code> equivalent of this access modifier.
	 */
	public String toString() {
		return mKeyword;
	}
}
```

```
package emuse.runtime.model;

/**
 * Desribes an action that can be "played" in steps.
 *
 * @author Mark Ayzenshtat
 * @author Vlad Shchogolev
 * @author Kristina Holst
 */
public interface Action {
	/**
	 * Initializes this action for execution.  If <code>iIndirectObject</code> is
	 * non-null, then <code>iPrep</code> must also be non-null.  Apart from that
	 * requirement, any of the arguments may be made null as needed.
	 *
	 * @param iSubject the subject
	 * @param iDirectObject the direct object
	 * @param iPrep the preposition
	 * @param iIndirectObject the indirect object
	 * @param iAdverb the adverb
	 */
	public void initialize(StageObject iSubject, Object iDirectObject,
		Preposition iPrep, StageObject iIndirectObject, Adverb iAdverb);

	/**
	 * Tells the action to update the associated StageObject
	 * in response to movement in time.
	 *
	 * @param secs      how much time to step over in seconds
	 * @return whether this action has completed
	 */
	public boolean step(double secs);

	/**
	 * Returns the "run order" number of this action.  The run order dictates
	 * when the action will be run by the scene manager.  A group of actions
	 * that should be run concurrently should be assigned the same run order.
	 *
	 * @return the "run order" number of this action
	 */
	public int getRunOrder();

	public void setRunOrder(int iOrder);
}
```

```
package emuse.runtime.model;

import java.util.*;

/**
 * A typesafe enumeration for adverbs.  Typesafe enum elements
 * can be safely tested for equality using the <code>==</code> operator.
 *
 * @author Mark Ayzenshtat
 */
public class Adverb {
	private static Map kPrivateValues = new HashMap(7);
	/** A collection of all values in this enum. */
	public static final Collection kValues;

	/** Quickly. */
	public static final Adverb kQuickly = new Adverb("quickly");

	/** Slowly. */
	public static final Adverb kSlowly = new Adverb("slowly");

	/** Loudly. */
	public static final Adverb kLoudly = new Adverb("loudly");

	/** Quietly. */
	public static final Adverb kQuietly = new Adverb("quietly");

	static {
		kValues = Collections.unmodifiableCollection(kPrivateValues.values());

	}

	private final String mValue;

	private Adverb(String iValue) {
		mValue = iValue;
		kPrivateValues.put(iValue, this);
	}

	/**
	 * Returns the <code>String</code> equivalent of this adverb.
	 */
	public String toString() {
		return mValue;
	}

	/**
	 * Returns the <code>Adverb</code> constant equivalent of the given string.
	 */
	public static Adverb valueOf(String iA) {
		return (Adverb) kPrivateValues.get(iA);
	}
}
```

```
package emuse.runtime.anim;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.*;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.text.DecimalFormat;
import java.util.*;
import javax.swing.Timer;

import javax.swing.JFrame;
import javax.swing.JLabel;

import emuse.runtime.model.Screenplay;

/**
 * @author Vlad
 * @author Mark
 *
 * AnimationFrame.java
 * Mar 11, 2003
 */
public class AnimationFrame extends JFrame implements ActionListener {
	// the last kNumFramesToEval frames will be evaluated in computing the FPS
	private static final int kNumFramesToEval = 1000;
	private static final int kFPS = 50;
	public static final Dimension kSize = new Dimension(600,600);

	// for animation
	int frameNumber = -1;
	Timer timer;
	boolean frozen = false;

	// for frame rate computation
	DecimalFormat df = new DecimalFormat("0.0");
	String rateStr = "";
	private long mTimeOfLastStep = 0;
	private long mLastSecondTime;
	private int mLastSecondFrames;
	private double mLastSecondFPS = -1;
	private long mLastFiveSecondsTime;
	private int mLastFiveSecondsFrames;
	private double mLastFiveSecondsFPS = -1;

	// components
	JLabel label;
	DisplayPanel display;

	private SceneManager sceneManager;

	public static AnimationFrame forScreenplay(Screenplay play) {
		SceneManager sm = new SceneManager(play);
		return new AnimationFrame(sm, play.getTitle());
	}

	AnimationFrame(SceneManager sm, String windowTitle) {
```

```
        super(windowTitle);
        this.sceneManager = sm;

        int delay = (kFPS > 0) ? (1000 / kFPS) : 100;

        timer = new Timer(delay, this);
        timer.setInitialDelay(0);
        timer.setCoalesce(true);

        addWindowListener(new WindowAdapter() {
                public void windowIconified(WindowEvent e) {
                        stopAnimation();
                }
                public void windowDeiconified(WindowEvent e) {
                        startAnimation();
                }
                public void windowClosing(WindowEvent e) {
                        System.exit(0);
                }
        });

        label = new JLabel("Ready", JLabel.CENTER);
        label.setFont(new Font("Courier", Font.BOLD, 14));
        label.addMouseListener(new MouseAdapter() {
                public void mousePressed(MouseEvent e) {
                        if (frozen) {
                                frozen = false;

                                mLastSecondTime = System.currentTimeMillis();
                                mLastFiveSecondsTime = mLastSecondTime;
                                mLastSecondFrames = frameNumber;
                                mLastFiveSecondsFrames = frameNumber;

                                startAnimation();
                        } else {
                                frozen = true;
                                stopAnimation();
                        }
                }
        });

        display = new DisplayPanel(sm);

        getContentPane().add(label, BorderLayout.SOUTH);
        getContentPane().add(display, BorderLayout.CENTER);

        display.setPreferredSize(kSize);
        pack();
        setResizable(false);
        setVisible(true);
}

//Can be invoked by any thread (since timer is thread-safe).
public void startAnimation() {
        if (!frozen) {
                timer.start();
        }
}

//Can be invoked by any thread (since timer is thread-safe).
public void stopAnimation() {
        timer.stop();
}
```

```
	public void actionPerformed(ActionEvent e) {
		animateFrame();
	}

	public void animateFrame() {
		if (mTimeOfLastStep == 0) {
			mTimeOfLastStep = System.currentTimeMillis();

			mLastSecondTime = mTimeOfLastStep;
			mLastSecondFrames = 0;
			mLastFiveSecondsTime = mTimeOfLastStep;
			mLastFiveSecondsFrames = 0;

			return;
		}

		long time = System.currentTimeMillis();
		double delta = (double) (time - mTimeOfLastStep);

		if (time - mLastSecondTime >= 1000) {
			// update last-second FPS
			mLastSecondFPS =
				(frameNumber - mLastSecondFrames) * 1000d / (time -
mLastSecondTime);
			mLastSecondTime = time;
			mLastSecondFrames = frameNumber;
		}

		if (time - mLastFiveSecondsTime >= 5000) {
			// update last-5-seconds FPS
			mLastFiveSecondsFPS =
				(frameNumber - mLastFiveSecondsFrames)
					* 1000d
					/ (time - mLastFiveSecondsTime);
			mLastFiveSecondsTime = time;
			mLastFiveSecondsFrames = frameNumber;
		}

		//Advance the animation frame.
		frameNumber++;

		StringBuffer fpsText = new StringBuffer(50);
		fpsText.append("Frames: ").append(frameNumber).append(
			", FPS (last 1 sec): ");
		if (mLastSecondFPS >= 0) {
			fpsText.append(df.format(mLastSecondFPS));
		} else {
			fpsText.append("--");
		}

		fpsText.append(", FPS (last 5 secs): ");

		if (mLastFiveSecondsFPS >= 0) {
			fpsText.append(df.format(mLastFiveSecondsFPS));
		} else {
			fpsText.append("--");
		}

		label.setText(fpsText.toString());

		/* if complete, stop animation */
		if (sceneManager.step(delta))
```

```
                stopAnimation();

            mTimeOfLastStep = time;

            display.repaint();
        }

        public Dimension getDisplaySize() {
            return display.getSize();
        }
}
```

```
package emuse.runtime.anim;

import emuse.runtime.model.*;
//import emuse.runtime.actions.*;

/**
 * @author vlad
 *
 *
 */
public class Animator {

	/**
	 * Displays animation of the given screenplay
	 *
	 * @param iPlay            the model of the screenplay
	 */
	public void animate(Screenplay iPlay) {

	}

	public static void main(String args[]) {


	}
}
```

```
package emuse.runtime.actions;

import emuse.runtime.model.*;

public class AppearsAction extends AbstractAction implements Action
{
	StageObject so, target;
	boolean done = false;

	public AppearsAction() {

	}

	public AppearsAction(StageObject so, StageObject target) {
		initialize(so, null, null, target, null);
	}

	public void initialize(StageObject iSubject, Object iDirectObject,
			Preposition iPrep, StageObject iIndirectObject, Adverb iAdverb) {
		so = iSubject;
		target = iIndirectObject;
	}

	/* (non-Javadoc)
	 * @see emuse.runtime.actions.AbstractAction#isDone()
	 */
	public boolean isDone() {
		return done;
	}

	/* (non-Javadoc)
	 * @see emuse.runtime.actions.AbstractAction#onFirstStep()
	 */
	protected void onFirstStep() {
		if (target == null) {
			Stage.getInstance().addDescendent(so);
		} else {
			target.addDescendent(so);
		}

		done = true;
	}

}
```

```
package emuse.compiler.javamodel;

import java.util.*;

/**
 * A block-style (slash-star/star-slash) Java comment.
 *
 * @author Mark Ayzenshtat
 */
public class BlockComment extends Comment {
	public BlockComment(String iText) {
		super();
		buildLinesFromText(linesForText(iText));
	}

	public BlockComment(String[] iLines) {
		super();
		buildLinesFromText(Arrays.asList(iLines));
	}

	private void buildLinesFromText(List iLines) {
		int numLines = iLines.size();
		mLines = new String[numLines + 2];
		mLines[0] = "/*";
		for (int i = 0; i < numLines; i++) {
			mLines[i + 1] = " * " + iLines.get(i);
		}
		mLines[numLines + 1] = " */";
	}
}
```

```
/*
 * Created on Apr 10, 2003
 *
 */
package emuse.runtime.anim;

import java.awt.image.BufferedImage;

import emuse.runtime.model.StageObject;

/**
 * Represents either body part supported by an image, or
 * by a series of shapes
 *
 *
 * @author Vladislav
 *
 */
public class BodyPart {
	private final boolean isImage;
	protected StageObject so;

	public BodyPart() {
		this.isImage = false;
	}

	public BodyPart(StageObject object) {
		this.isImage = false;
		this.so = object;
	}

	public BodyPart(BufferedImage bi) {
		this.isImage = true;
		// todo: create ImageObject out of BufferedImage
	}

	public StageObject toStageObject() {
		return so;
	}
}
```

```
package emuse.runtime.model;

import java.util.*;

/**
 * A typesafe enumeration for builds.  Typesafe enum elements
 * can be safely tested for equality using the <code>==</code> operator.
 *
 * @author Mark Ayzenshtat
 */
public class Build {
	private static Map kPrivateValues = new HashMap(7);
	/** A collection of all values in this enum. */
	public static final Collection kValues;

	/** Thin. */
	public static final Build kThin = new Build("thin");

	/** Medium. */
	public static final Build kMedium = new Build("medium");

	/** Fat. */
	public static final Build kFat = new Build("fat");

	static {
		kValues = Collections.unmodifiableCollection(kPrivateValues.values());

	}

	private final String mValue;

	private Build(String iValue) {
		mValue = iValue;
		kPrivateValues.put(iValue, this);
	}

	/**
	 * Returns the <code>String</code> equivalent of this build.
	 */
	public String toString() {
		return mValue;
	}

	/**
	 * Returns the <code>Build</code> constant equivalent of the given string.
	 */
	public static Build valueOf(String iB) {
		return (Build) kPrivateValues.get(iB);
	}
}
```

```
package emuse.runtime.model;

import java.awt.*;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Shape;
import java.awt.geom.Ellipse2D;
import java.awt.geom.RoundRectangle2D;
import java.awt.image.BufferedImage;

import com.sun.rsasign.i;

import emuse.runtime.anim.ImageObject;
import emuse.runtime.anim.Knob;
import emuse.runtime.anim.PivotedShape;

/**
 * @author Mark Ayzenshtat
 * @author Vlad Shchogolev
 */
public class Character extends StageObject {
	private CharType mCharType;

	private double
		bodyHeight,
		bodyWidth,
		headDiameter,
		armL,
		armW,
		legL,
		legW;

	public final Knob
		neck = new Knob(),
		lShoulder = new Knob(1, 0),
		rShoulder = new Knob(-1, 0),
		lElbow = new Knob(),
		rElbow = new Knob(-1, 0),
		lLeg = new Knob(),
		rLeg = new Knob(),
		lKnee = new Knob(),
		rKnee = new Knob();

	StageObject leftArm, rightArm;
	StageObject leftLeg, rightLeg;
	StageObject head, body;

	public Character() {
		this(0,0,null);
		this.mOnStage = false;
	}

	public Character(double x, double y, String s) {}

	public void setCharType(CharType type) {
		mCharType = type;

		Gender gender = type.getGender();

		if (gender == Gender.kMale) {
			bodyWidth = 100;
		} else {
			bodyWidth = 80;
```

```
}

bodyHeight = 200;
headDiameter = 100;
armL = 100;
armW = 20;
legL = 100;
legW = 30;

Height height = type.getHeight();
if (height == Height.kShort) {
	bodyHeight *= 0.7;
	armL *= 0.7;
	legL *= 0.7;
} else if (height == Height.kTall) {
	bodyHeight *= 1.3;
	armL *= 1.3;
	legL *= 1.3;
}

Build build = type.getBuild();
if (build == Build.kThin) {
	bodyWidth *= 0.7;
	armW *= 0.7;
	legW *= 0.7;
} else if (build == Build.kFat) {
	bodyWidth *= 1.3;
	armW *= 1.3;
	legW *= 1.3;
}

// temporary
scale.setValue(0.5);

BufferedImage face = type.getHeadImage();

// set up body parts
if (face != null) {
	head = new ImageObject(face);
} else {
	head = createHead(headDiameter, neck);
}

body = createBody(bodyWidth, bodyHeight, gender);
leftArm = createLimb(armW, armL, lShoulder, lElbow, Color.PINK);
rightArm = createLimb(armW, armL, rShoulder, rElbow, Color.PINK);
leftLeg = createLimb(legW, legL, lLeg, lKnee, Color.BLACK);
rightLeg = createLimb(legW, legL, rLeg, rKnee, Color.BLACK);

addDescendent(body);
body.addDescendent(head);
body.addDescendent(leftArm);
body.addDescendent(rightArm);
body.addDescendent(leftLeg);
body.addDescendent(rightLeg);

leftArm.setPivot(-bodyWidth/2 - 10, -bodyHeight/2 + 10);
rightArm.setPivot(bodyWidth/2 + 10, -bodyHeight/2 + 10);

leftLeg.setPivot(-bodyWidth/2 + 10, bodyHeight/2 + 10);
rightLeg.setPivot(bodyWidth/2 - 10, bodyHeight/2 + 10);

head.setPivot(0.0, -bodyHeight/2 + 10);
```

```
		//lShoulder.setValue(1);
		//lElbow.setValue(1);
		//rShoulder.setValue(1);
		//rElbow.setValue(1);

		mWidth = bodyWidth + 2 * armW;
		mHeight = bodyHeight + 4 * legL;

		body.yLoc.setValue(-mHeight/2);
	}

	protected void drawObject(Graphics2D g2) {
		// TODO: draw name tag
	}

	private PivotedShape createHead(double diam, Knob knob) {
		// head is drawn on "top" of the head pivot point
		Shape shape = new Ellipse2D.Double(-diam/2, -diam, diam, diam);
		Color color = Color.PINK;
		PivotedShape ps = new PivotedShape(shape, color);
		ps.setKnob(knob);
		return ps;
	}

	private PivotedShape createBody(double w, double h, Gender iGender) {
		if (iGender == Gender.kMale) {
			// the body is centered at the body pivot
			Shape shape  = new RoundRectangle2D.Double(
				-w/2, -h/2, w, h, 10, 10);
			Color color = Color.BLUE;
			return new PivotedShape(shape, color);
		}

		// female

		int wOver2 = (int) w / 2;
		int hOver2 = (int) h / 2;

		Shape shape = new Polygon(
			new int[] {-wOver2, wOver2, wOver2, wOver2 + wOver2, -wOver2 -
wOver2, -wOver2},
			new int[] {-hOver2, -hOver2, 0, hOver2, hOver2, 0}, 6);
		Color color = Color.MAGENTA;
		return new PivotedShape(shape, color);
	}

	private PivotedShape createLimb(double w, double h, Knob knob, Knob bend, Color
c)
	{
		PivotedShape lowerPart = new PivotedShape(-w/2, 0d, w, h, c);
		lowerPart.setPivot(0, h-w/2);
		lowerPart.setKnob(bend);

		PivotedShape ps = new PivotedShape(
			new RoundRectangle2D.Double(-w/2, -w/2, w, h, w/2, w/2), c);
		ps.setKnob(knob);
		ps.addDescendent(lowerPart);

		return ps;
	}

	/**
```

```
	 * Returns the body.
	 * @return Body
	 */
	public StageObject getBody() {
		return body;
	}

	/**
	 * Returns the head.
	 * @return Head
	 */
	public StageObject getHead() {
		return head;
	}

	/**
	 * @return CharType
	 */
	public CharType getCharType() {
		return mCharType;
	}

}
```

```
package emuse.runtime.model;

import java.awt.image.BufferedImage;

/**
 * A character type.
 *
 * @author Mark Ayzenshtat
 */
public class CharType {
	private BufferedImage mHeadImage;
	private BufferedImage mArmImage;
	private BufferedImage mLegImage;
	private BufferedImage mTorsoImage;
	private Gender mGender;
	private Height mHeight;
	private Build mBuild;

	public boolean isStickFigure() {
		return (mHeadImage == null || mArmImage == null
			|| mLegImage == null || mTorsoImage == null);
	}

  /**
   * @return
   */
  public BufferedImage getArmImage() {
    return mArmImage;
  }

  /**
   * @return
   */
  public Build getBuild() {
    return mBuild;
  }

  /**
   * @return
   */
  public Gender getGender() {
    return mGender;
  }

  /**
   * @return
   */
  public BufferedImage getHeadImage() {
    return mHeadImage;
  }

  /**
   * @return
   */
  public Height getHeight() {
    return mHeight;
  }

  /**
   * @return
   */
  public BufferedImage getLegImage() {
    return mLegImage;
```

```
  }

  /**
   * @return
   */
  public BufferedImage getTorsoImage() {
    return mTorsoImage;
  }

  /**
   * @param iImage
   */
  public void setArmImage(BufferedImage iImage) {
    mArmImage = iImage;
  }

  /**
   * @param iBuild
   */
  public void setBuild(Build iBuild) {
    mBuild = iBuild;
  }

  /**
   * @param iGender
   */
  public void setGender(Gender iGender) {
    mGender = iGender;
  }

  /**
   * @param iImage
   */
  public void setHeadImage(BufferedImage iImage) {
    mHeadImage = iImage;
  }

  /**
   * @param iHeight
   */
  public void setHeight(Height iHeight) {
    mHeight = iHeight;
  }

  /**
   * @param iImage
   */
  public void setLegImage(BufferedImage iImage) {
    mLegImage = iImage;
  }

  /**
   * @param iImage
   */
  public void setTorsoImage(BufferedImage iImage) {
    mTorsoImage = iImage;
  }
}
```

```
package emuse.compiler.javamodel;

import java.util.*;

/**
 * A Java class.
 *
 * @author Mark Ayzenshtat
 */
public class Class implements Commentable, HasModifiers {
	private String mName;
	private List mClasses;
	private List mMethods;
	private List mFields;
	private Comment mComment;
	private AccessModifier mAccess;
	private Set mMiscModifiers;
	private String mExtended;
	private List mImplemented;

	public Class() {
		mClasses = new ArrayList();
		mMethods = new ArrayList();
		mFields = new ArrayList();
		mImplemented = new ArrayList();
		mMiscModifiers = new HashSet();
		mAccess = AccessModifier.kDefault;
	}

	public void setName(String iName) {
		mName = iName;
	}

	public void setComment(Comment iC) {
		mComment = iC;
	}

	public void setAccessModifier(AccessModifier iAccess) {
		mAccess = iAccess;
	}

	public void setExtendedClass(String iSuperclass) {
		mExtended = iSuperclass;
	}

	public void addClass(Class iC) {
		mClasses.add(iC);
	}

	public void addMethod(Method iM) {
		mMethods.add(iM);
	}

	public void addField(Field iF) {
		mFields.add(iF);
	}

	public void addImplementedInterface(String iInterface) {
		mImplemented.add(iInterface);
	}

	public void addModifier(MiscModifier iModifier) {
		mMiscModifiers.add(iModifier);
```

```
	}

	public String getName() {
		return mName;
	}

	public AccessModifier getAccessModifier() {
		return mAccess;
	}

	public Set getModifiers() {
		return Collections.unmodifiableSet(mMiscModifiers);
	}

	public List getClasses() {
		return Collections.unmodifiableList(mClasses);
	}

	public Comment getComment() {
		return mComment;
	}

	public String getExtendedClass() {
		return mExtended;
	}

	public List getFields() {
		return Collections.unmodifiableList(mFields);
	}

	public List getImplementedInterfaces() {
		return mImplemented;
	}

	public List getMethods() {
		return Collections.unmodifiableList(mMethods);
	}

}
```

```
package emuse.compiler.javamodel;

import java.util.*;

/**
 * Represents a Java comment.
 *
 * @author Mark Ayzenshtat
 */
public abstract class Comment {
	protected String[] mLines;

	protected Comment() {
	}

	public String[] getLines() {
		return mLines;
	}

	public String toString() {
		String[] lines = getLines();
		StringBuffer sb = new StringBuffer(80 * lines.length);

		for (int i = 0; i < lines.length; i++) {
			sb.append(lines[i]).append('\n');
		}

		return sb.toString();
	}

	protected List linesForText(String iText) {
		StringTokenizer st = new StringTokenizer(iText, "\n\r\f");
		List lines = new ArrayList();
		while (st.hasMoreTokens()) {
			lines.add(st.nextToken());
		}

		return lines;
	}
}
```

```
package emuse.compiler.javamodel;

/**
 * @author Mark Ayzenshtat
 */
public interface Commentable {
       public void setComment(Comment iC);
  public Comment getComment();
}
```

```
package emuse.compiler.antlr;

import java.io.*;

/**
 * @author Mark Ayzenshtat
 */
public class Compiler {
  public static void main(String[] args) {
    if (args.length < 1) {
      System.err.println("Usage: java Compiler <input_file>");
      return;
    }

    try {
       long start = System.currentTimeMillis();
      L lexer = new L(new DataInputStream(new FileInputStream(args[0])));
      P parser = new P(lexer);
      parser.setSourceFilePath(args[0]);
      parser.startRule();
      long end = System.currentTimeMillis();

      System.out.println("\"" + args[0] + "\" compiled in " +
       ((end - start)/(float)1000) + " seconds.");
    } catch (Exception e) {
      System.err.println("Error: " + e);
    }
  }
}
```

```
package emuse.runtime.anim;

import javax.swing.JPanel;
import java.awt.*;
import java.awt.image.*;

/**
 * @author vlad
 *
 * DisplayPanel.java
 * Mar 11, 2003
 */
public class DisplayPanel extends JPanel {
	private VolatileImage vImg;
	private SceneManager sceneManager;

	DisplayPanel(SceneManager sm) {
		super();

		this.sceneManager = sm;
		setBackground(Color.WHITE);
	}

	public void clear() {
		getGraphics().clearRect(0, 0, getWidth(), getHeight());
	}

	public void paintComponent(Graphics g) {
		super.paintComponent(g);

		if (vImg == null) {
			createBackBuffer();
		}

		draw(g);
	}

	private void draw(Graphics g) {
		do {
			// Test if image is lost and restore it.
			GraphicsConfiguration gc = this.getGraphicsConfiguration();
			int valCode = vImg.validate(gc);
			// No need to check for IMAGE_RESTORED since we are
			// going to re-render the image anyway.
			if (valCode == VolatileImage.IMAGE_INCOMPATIBLE) {
				createBackBuffer();
			}

			// Render to the Image
			renderFrame();
			// Render image to screen.

			g.drawImage(vImg, 0, 0, this);
			// Test if content is lost
		} while (vImg.contentsLost());
	}

	private void createBackBuffer() {
		GraphicsConfiguration gc = getGraphicsConfiguration();
		vImg = gc.createCompatibleVolatileImage(getWidth(), getHeight());
	}

	private void renderFrame() {
```

```
                Graphics g = vImg.getGraphics();

                g.clearRect(0, 0, getWidth(), getHeight());
                sceneManager.drawScene(g);
        }
}
```

```
package emuse;

/**
 * Represents a general or unspecified exception in the eMuse runtime or
 * compiler.
 *
 * @author Mark Ayzenshtat
 */
public class EmuseException extends Exception {
	public EmuseException() {
		super();
	}

	public EmuseException(String iMessage) {
		super(iMessage);
	}

	public EmuseException(String iMessage, Throwable iCause) {
		super(iMessage, iCause);
	}
}
```

```
package emuse.compiler.javamodel;

import java.util.*;

/**
 * A Java field.
 *
 * @author Mark Ayzenshtat
 */
public class Field implements Commentable, HasModifiers {
	private Comment mComment;
  private String mType;
  private String mName;
	private Set mMiscModifiers;
	private AccessModifier mAccess;

	public Field(String iType, String iName) {
		this();
		setType(iType);
		setName(iName);
	}

	public Field() {
		mAccess = AccessModifier.kDefault;
		mMiscModifiers = new HashSet();
	}

	public void setComment(Comment iC) {
		mComment = iC;
	}

  public Comment getComment() {
    return mComment;
  }

  public void setType(String iType) {
    mType = iType;
  }

  public String getType() {
    return mType;
  }

  public void setName(String iName) {
    mName = iName;
  }

  public String getName() {
    return mName;
  }

	public void setAccessModifier(AccessModifier iAccess) {
		mAccess = iAccess;
	}

	public AccessModifier getAccessModifier() {
		return mAccess;
	}

	public void addModifier(MiscModifier iModifier) {
		mMiscModifiers.add(iModifier);
	}
```

```
	public Set getModifiers() {
		return Collections.unmodifiableSet(mMiscModifiers);
	}
}
```

```
package emuse.runtime.model;

import java.util.*;

/**
 * A typesafe enumeration for genders.  Typesafe enum elements
 * can be safely tested for equality using the <code>==</code> operator.
 *
 * @author Mark Ayzenshtat
 */
public class Gender {
	private static Map kPrivateValues = new HashMap(7);
	/** A collection of all values in this enum. */
	public static final Collection kValues;

	/** Male. */
	public static final Gender kMale = new Gender("male");

	/** Female. */
	public static final Gender kFemale = new Gender("female");

	static {
		kValues = Collections.unmodifiableCollection(kPrivateValues.values());

	}

	private final String mValue;

	private Gender(String iValue) {
		mValue = iValue;
		kPrivateValues.put(iValue, this);
	}

	/**
	 * Returns the <code>String</code> equivalent of this gender.
	 */
	public String toString() {
		return mValue;
	}

	/**
	 * Returns the <code>Gender</code> constant equivalent of the given string.
	 */
	public static Gender valueOf(String iG) {
		return (Gender) kPrivateValues.get(iG);
	}
}
```

```
package emuse.runtime.model;


/**
 * @author Mark Ayzenshtat
 */
public class Group {
	private String mName;
	private StageObject[] mMembers;

	public String getName() {
		return mName;
	}

	public void setName(String name) {
		mName = name;
	}

	public StageObject[] getMembers() {
		return mMembers;
	}

	public void setMembers(StageObject[] members) {
		mMembers = members;
	}
}
```

```
package emuse.compiler.javamodel;

import java.util.Set;

/**
 * @author Mark Ayzenshtat
 */
public interface HasModifiers {
	public void setAccessModifier(AccessModifier iAccess);

	public AccessModifier getAccessModifier();

	public void addModifier(MiscModifier iModifier);

	public Set getModifiers();
}
```

```
package emuse.runtime.model;

import java.util.*;

/**
 * A typesafe enumeration for heights.  Typesafe enum elements
 * can be safely tested for equality using the <code>==</code> operator.
 *
 * @author Mark Ayzenshtat
 */
public class Height {
	private static Map kPrivateValues = new HashMap(7);
	/** A collection of all values in this enum. */
	public static final Collection kValues;

	/** Short. */
	public static final Height kShort = new Height("short");

	/** Medium. */
	public static final Height kMedium = new Height("medium");

	/** Tall. */
	public static final Height kTall = new Height("tall");

	static {
		kValues = Collections.unmodifiableCollection(kPrivateValues.values());

	}

	private final String mValue;

	private Height(String iValue) {
		mValue = iValue;
		kPrivateValues.put(iValue, this);
	}

	/**
	 * Returns the <code>String</code> equivalent of this height.
	 */
	public String toString() {
		return mValue;
	}

	/**
	 * Returns the <code>Height</code> constant equivalent of the given string.
	 */
	public static Height valueOf(String iH) {
		return (Height) kPrivateValues.get(iH);
	}
}
```

```
/*
 * Created on Apr 15, 2003
 */
package emuse.runtime.anim;

import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.IOException;

import emuse.EmuseException;
import emuse.ResourceManager;
import emuse.runtime.model.StageObject;

/**
 * @author Vladislav
 */
public class ImageObject extends StageObject {

	BufferedImage mImage;

	public ImageObject(String file) throws IOException {
		try {
			mImage = ResourceManager.getInstance().getImage(file);
			mWidth = mImage.getWidth();
			mHeight = mImage.getHeight();
		} catch (EmuseException e) {
			System.out.println(e);
			e.printStackTrace();
			System.exit(1);
		}
	}

	public ImageObject(BufferedImage bimg) {
		mImage = bimg;
		mWidth = mImage.getWidth();
		mHeight = mImage.getHeight();
	}

	/* (non-Javadoc)
	 * @see emuse.runtime.model.StageObject#drawObject(java.awt.Graphics2D)
	 */
	protected void drawObject(Graphics2D g) {
		g.drawImage(mImage, null, (int)-mWidth/2, (int)-mHeight);
	}
}
```

```
package emuse.compiler.javamodel;

import java.util.*;

/**
 * A Javadoc-style (slash-star-star/star-slash) Java comment.
 *
 * @author Mark Ayzenshtat
 */
public class JavadocComment extends Comment {
	public JavadocComment(String iText) {
		super();
		buildLinesFromText(linesForText(iText));
	}

	public JavadocComment(String[] iLines) {
		super();
		buildLinesFromText(Arrays.asList(iLines));
	}

	private void buildLinesFromText(List iLines) {
		int numLines = iLines.size();
		mLines = new String[numLines + 2];
		mLines[0] = "/**";
		for (int i = 0; i < numLines; i++) {
			mLines[i + 1] = " * " + iLines.get(i);
		}
		mLines[numLines + 1] = " */";
	}
}
```

```
package emuse.compiler;

import java.io.*;
import java.util.*;
import emuse.EmuseException;
import emuse.StringUtils;
import emuse.compiler.javamodel.*;
import emuse.compiler.javamodel.Class;

/**
 * <p>
 * An output generator that generates Java source code output for the eMuse
 * runtime library.
 * </p>
 *
 * @author Mark Ayzenshtat
 */
public class JavaOutputGenerator implements OutputGenerator {
	private String mOutputClassName;
	private JavaSource mSource;
	private Class mClass;
	private String mTitle;
	private List mAuthors;

	private int mNextRunOrder;
	private Map mCTNames;
	private Map mPTNames;
	private Map mSetNames;
	private Map mCNames;
	private Map mPNames;
	private Map mGNames;
	private List mScenes;
	private CT mCurCT;
	private PT mCurPT;
	private Set mCurSet;
	private G mCurG;
	private S mCurS;

	public JavaOutputGenerator(String iSourceFilePath) {
		if (iSourceFilePath == null || iSourceFilePath.equals("")) {
			mOutputClassName = "ScreenplayOutput";
		} else {
			File f = new File(iSourceFilePath);
			String fileName = f.getName();
			if (iSourceFilePath.equals("")) {
				mOutputClassName = "ScreenplayOutput";
			} else {
				mOutputClassName =
StringUtils.screenplayOutputClassName(fileName);
			}
		}

		mNextRunOrder = 0;
	}

	/**
	 * Begins generating output.  This method must be called to initialize this
	 * output generator before any <code>outputXXX()</code> methods are called.
	 */
	public void startOutput() {
		mCTNames = new HashMap();
		mPTNames = new HashMap();
		mSetNames = new HashMap();
```

```
            mCNames = new HashMap();
            mPNames = new HashMap();
            mGNames = new HashMap();
            mScenes = new ArrayList(20);

            mCurCT = null;
            mCurPT = null;
            mCurSet = null;
            mCurG = null;
            mCurS = null;

            mTitle = "";
            mAuthors = new ArrayList();

            mSource = new JavaSource();
            mSource.setComment(new BlockComment(new String[] {
                  "This source file was automatically generated by eMuse.",
                  "Do not modify."
            }));

            mSource.addImport("java.util.*");
            mSource.addImport("emuse.*");
            mSource.addImport("emuse.runtime.actions.*");
            mSource.addImport("emuse.runtime.anim.*");
            mSource.addImport("emuse.runtime.model.*");
            mSource.addImport("emuse.runtime.model.Character");

            mClass = new Class();
            mClass.setAccessModifier(AccessModifier.kPublic);
            mClass.addModifier(MiscModifier.kFinal);
            mClass.setComment(new JavadocComment("Generated screenplay class."));

            mClass.setName(mOutputClassName);
            mSource.addClass(mClass);

            Field f = new Field();
            f.setAccessModifier(AccessModifier.kPrivate);
            f.setType("Screenplay");
            f.setName("mScreenplay");
            mClass.addField(f);

            // make the main() method
            Method main = Method.newMainMethod();
            main.addBodyLine("new " + mOutputClassName + "();");
            mClass.addMethod(main);

            // make the constructor
            Method cons = new Method();
            cons.setAccessModifier(AccessModifier.kPublic);
            cons.setName(mOutputClassName);
            cons.setComment(new JavadocComment("Creates the screenplay state and
hands it off to a scene manager."));
            mClass.addMethod(cons);

            cons.addBodyLine("try {");
            cons.indent();
            cons.addBodyLine("mScreenplay = new Screenplay();");
            cons.addBodyLine("");
            cons.addBodyLine("setTitleAndAuthors();");
            cons.addBodyLine("");
            cons.addBodyLine("defineCharTypes();");
            cons.addBodyLine("definePropTypes();");
            cons.addBodyLine("defineSettings();");
```

```
                cons.addBodyLine("defineCharacters();");
                cons.addBodyLine("defineProps();");
                cons.addBodyLine("defineGroups();");
                cons.addBodyLine("defineScenes();");
                cons.addBodyLine("");
                cons.addBodyLine("AnimationFrame animator =
AnimationFrame.forScreenplay(mScreenplay);");
                cons.addBodyLine("animator.startAnimation();");
                cons.outdent();
                cons.addBodyLine("} catch (EmuseException ex) {");
                cons.indent();
                cons.addBodyLine("ex.printStackTrace();");
                cons.outdent();
                cons.addBodyLine("}");
        }

        /**
         * Outputs the title of the screenplay.
         *
         * @param iTitle the title
         */
        public void outputTitle(String iTitle) {
                mTitle = iTitle;
        }

        /**
         * Outputs an author of the screenplay.  This method may be called more than
         * once if the screenplay has more than one author.
         *
         * @param iAuthor an author
         */
        public void outputAuthor(String iAuthor) {
                mAuthors.add(iAuthor);
        }

        /**
         * Begins to output a CharType.
         *
         * @param iLabel the label used to identify the CharType
         */
        public void outputCharType(String iLabel) {
                CT ct = new CT();
                ct.mVar = "ct" + mCTNames.size();
                mCurCT = ct;

                mCTNames.put(iLabel, ct);
        }

        /**
         * Outputs a CharType head parameter.
         *
         * @param iImageFile the image file
         */
        public void outputCharTypeHead(String iImageFile) {
                mCurCT.mHead = iImageFile;
        }

        /**
         * Outputs a CharType arm parameter.
         *
         * @param iImageFile the image file
         */
        public void outputCharTypeArm(String iImageFile) {
```

```
        mCurCT.mArm = iImageFile;
}

/**
 * Outputs a CharType leg parameter.
 *
 * @param iImageFile the image file
 */
public void outputCharTypeLeg(String iImageFile) {
        mCurCT.mLeg = iImageFile;
}

/**
 * Outputs a CharType torso parameter.
 *
 * @param iImageFile the image file
 */
public void outputCharTypeTorso(String iImageFile) {
        mCurCT.mTorso = iImageFile;
}

/**
 * Outputs the CharType body parameters.
 */
public void outputCharTypeBody(String iGender, String iHeight,
              String iBuild) {
        mCurCT.mGender = iGender;
        mCurCT.mHeight = iHeight;
        mCurCT.mBuild = iBuild;
}

/**
 * Begins to output a PropType.
 *
 * @param iLabel the label used to identify the PropType
 */
public void outputPropType(String iLabel) {
        PT pt = new PT();
        pt.mVar = "pt" + mPTNames.size();
        mCurPT = pt;

        mPTNames.put(iLabel, pt);
}

/**
 * Outputs a PropType image parameter.
 *
 * @param iImageFile the image file
 */
public void outputPropTypeImage(String iImageFile) {
        mCurPT.mImage = iImageFile;
}

/**
 * Outputs a PropType scale parameter.
 *
 * @param iImageFile the scale
 */
public void outputPropTypeScale(String iScale) {
        mCurPT.mScale = iScale;
}

/**
```

```
 * Outputs a setting.
 *
 * @param iLabel the label used to identify the setting
 */
public void outputSetting(String iLabel) {
	Set s = new Set();
	s.mVar = "set" + mSetNames.size();
	mCurSet = s;

	mSetNames.put(iLabel, s);
}

/**
 * Outputs the image field of a setting.
 *
 * @param iImageFile the image file
 */
public void outputSettingImage(String iImageFile) {
	mCurSet.mImage = iImageFile;
}

/**
 * Outputs the sound field of a setting.
 *
 * @param iSoundFile the sound file
 */
public void outputSettingSound(String iSoundFile) {
	mCurSet.mSound = iSoundFile;
}

/**
 * Outputs a character definition.
 *
 * @param iName the character's name
 * @param iCharType the CharType
 */
public void outputCharacter(String iName, String iCharType) {
	C c = new C();
	c.mVar = "c" + mCNames.size();
	c.mCT = (CT) mCTNames.get(iCharType);

	mCNames.put(iName, c);
}

/**
 * Outputs a prop definition.
 *
 * @param iName the character's name
 * @param iPropType the CharType
 */
public void outputProp(String iName, String iPropType) {
	P p = new P();
	p.mVar = "p" + mPNames.size();
	p.mPT = (PT) mPTNames.get(iPropType);

	mPNames.put(iName, p);
}

/**
 * Outputs a group definition.
 *
 * @param iName the name of the group
 */
```

```
public void outputGroup(String iName) {
       G g = new G();
       g.mVar = "g" + mGNames.size();
       mCurG = g;

       mGNames.put(iName, g);
}

/**
 * Outputs a group member.  This method may be called more than
 * once if the group has more than one member.
 * @param iName the group member
 */
public void outputGroupMember(String iName) {
       mCurG.mMembers.add(iName);
}

/**
 * Outputs a new scene.
 *
 * @param iSetting the setting of the scene
 */
public void outputScene(String iSetting) {
       S s = new S();
       s.mSetting = iSetting;
       mCurS = s;

       mScenes.add(s);
}

/**
 * Outputs an action.
 *
 * @param iSubject the subject
 * @param iVerb the verb
 * @param iDirObj the direct object
 * @param iPrep the preposition
 * @param iIndirObj the indirect object
 * @param iAdverb the adverb
 */
public void outputAction(String iSubject, String iVerb, String iDirObj,
              String iPrep, String iIndirObj, String iAdverb) {
       Action a = new Action();

       a.mSubject = iSubject;
       a.mVerb = iVerb;
       a.mDirObj = iDirObj;
       a.mPrep = iPrep;
       a.mIndirObj = iIndirObj;
       a.mAdverb = iAdverb;
       a.mRunOrder = nextRunOrder();

       mCurS.mActions.add(a);
}

/**
 * Completes output generation.  This method must be called to finalize
 * output generation after all <code>outputXXX()</code> methods have been
 * called.
 */
public void finishOutput() throws EmuseException {
       makePrivateFields();
```

```
		makeSetTitleAndAuthorsMethod();
		makeDefineCharTypesMethod();
		makeDefinePropTypesMethod();
		makeDefineSettings();
		makeDefineCharacters();
		makeDefineProps();
		makeDefineGroups();
		makeDefineScenes();

		// write the source model
		try {
			JavaSourceWriter jsw = new StandardJavaSourceWriter(
				new java.io.FileWriter("src/" + mOutputClassName +
".java"));
			jsw.writeJavaSource(mSource);
			jsw.close();
		} catch (IOException ex) {
			throw new EmuseException("Error while writing java source.", ex);
		}
	}

	private int nextRunOrder() {
		return mNextRunOrder++;
	}

	private Method createPrivateVoidMethod(String iName) {
		Method m = new Method();
		m.setAccessModifier(AccessModifier.kPrivate);
		m.setReturnType("void");
		m.setName(iName);
		m.addException("EmuseException");
		mClass.addMethod(m);
		return m;
	}

	private void createPrivateField(String iType, String iName) {
		Field f = new Field();
		f.setAccessModifier(AccessModifier.kPrivate);
		f.setType(iType);
		f.setName(iName);
		mClass.addField(f);
	}

	private void makePrivateFields() {
		for (Iterator i = mCTNames.values().iterator(); i.hasNext(); ) {
			CT ct = (CT) i.next();
			createPrivateField("CharType", ct.mVar);
		}

		for (Iterator i = mPTNames.values().iterator(); i.hasNext(); ) {
			PT pt = (PT) i.next();
			createPrivateField("PropType", pt.mVar);
		}

		for (Iterator i = mSetNames.values().iterator(); i.hasNext(); ) {
			Set set = (Set) i.next();
			createPrivateField("Setting", set.mVar);
		}

		for (Iterator i = mCNames.values().iterator(); i.hasNext(); ) {
			C c = (C) i.next();
			createPrivateField("Character", c.mVar);
		}
```

```
		for (Iterator i = mPNames.values().iterator(); i.hasNext(); ) {
			P p = (P) i.next();
			createPrivateField("Prop", p.mVar);
		}

		for (Iterator i = mGNames.values().iterator(); i.hasNext(); ) {
			G g = (G) i.next();
			createPrivateField("Group", g.mVar);
		}
	}

	private void makeSetTitleAndAuthorsMethod() {
		Method m = createPrivateVoidMethod("setTitleAndAuthors");

		m.addBodyLine("mScreenplay.setTitle(\"" + mTitle + "\");");

		StringBuffer sb = new StringBuffer(100);
		sb.append("mScreenplay.setAuthors(new String[] {");
		if (!mAuthors.isEmpty()) {
			sb.append('\"').append(mAuthors.get(0)).append('\"');
			for (int i = 1, n = mAuthors.size(); i < n; i++) {
				sb.append(",
").append('\"').append(mAuthors.get(i)).append('\"');
			}
		}

		sb.append("});");

		m.addBodyLine(sb.toString());
	}

	private void makeDefineCharTypesMethod() {
		Method m = createPrivateVoidMethod("defineCharTypes");

		m.addBodyLine("ResourceManager rm = ResourceManager.getInstance();");
		m.addBodyLine("");

		for (Iterator i = mCTNames.values().iterator(); i.hasNext(); ) {
			CT ct = (CT) i.next();

			m.addBodyLine(ct.mVar + " = new CharType();");
			m.addBodyLine(ct.mVar +
				".setHeadImage(rm.getImage(\"" + ct.mHead + "\"));");
			m.addBodyLine(ct.mVar +
				".setArmImage(rm.getImage(\"" + ct.mArm + "\"));");
			m.addBodyLine(ct.mVar +
				".setLegImage(rm.getImage(\"" + ct.mLeg + "\"));");
			m.addBodyLine(ct.mVar +
				".setTorsoImage(rm.getImage(\"" + ct.mTorso + "\"));");
			m.addBodyLine(ct.mVar +
				".setGender(Gender.valueOf(\"" + ct.mGender + "\"));");
			m.addBodyLine(ct.mVar +
				".setHeight(Height.valueOf(\"" + ct.mHeight + "\"));");
			m.addBodyLine(ct.mVar +
				".setBuild(Build.valueOf(\"" + ct.mBuild + "\"));");
			m.addBodyLine("");
		}

		String s = commaSeparatedList(mCTNames.values().iterator());
		m.addBodyLine("mScreenplay.setCharacterTypes(new CharType[] {" + s +
"});");
	}
```

```
private void makeDefinePropTypesMethod() {
	Method m = createPrivateVoidMethod("definePropTypes");

	m.addBodyLine("ResourceManager rm = ResourceManager.getInstance();");
	m.addBodyLine("");

	for (Iterator i = mPTNames.values().iterator(); i.hasNext(); ) {
		PT pt = (PT) i.next();

		m.addBodyLine(pt.mVar + " = new PropType();");
		m.addBodyLine(pt.mVar +
			".setAppearance(rm.getImage(\"" + pt.mImage + "\"));");
		m.addBodyLine(pt.mVar +
			".setScale(Scale.valueOf(\"" + pt.mScale + "\"));");
		m.addBodyLine("");
	}

	String s = commaSeparatedList(mPTNames.values().iterator());
	m.addBodyLine("mScreenplay.setPropTypes(new PropType[] {" + s + "});");
}

private void makeDefineSettings() {
	Method m = createPrivateVoidMethod("defineSettings");

	m.addBodyLine("ResourceManager rm = ResourceManager.getInstance();");
	m.addBodyLine("");

	for (Iterator i = mSetNames.values().iterator(); i.hasNext(); ) {
		Set s = (Set) i.next();

		m.addBodyLine(s.mVar + " = new Setting();");
		m.addBodyLine(s.mVar +
			".setImage(rm.getImage(\"" + s.mImage + "\"));");
		m.addBodyLine(s.mVar +
			".setSound(rm.getSound(\"" + s.mSound + "\"));");
		m.addBodyLine("");
	}

	String s = commaSeparatedList(mSetNames.values().iterator());
	m.addBodyLine("mScreenplay.setSettings(new Setting[] {" + s + "});");
}

private void makeDefineCharacters() {
	Method m = createPrivateVoidMethod("defineCharacters");

	for (Iterator i = mCNames.keySet().iterator(); i.hasNext(); ) {
		String name = (String) i.next();
		C c = (C) mCNames.get(name);

		m.addBodyLine(c.mVar + " = new Character();");
		m.addBodyLine(c.mVar +
			".setName(\"" + name + "\");");
		m.addBodyLine(c.mVar +
			".setCharType(" + c.mCT.mVar + ");");
		m.addBodyLine("");
	}

	String s = commaSeparatedList(mCNames.values().iterator());
	m.addBodyLine("mScreenplay.setCharacters(new Character[] {" + s + "});");
}

private void makeDefineProps() {
```

```
            Method m = createPrivateVoidMethod("defineProps");

            for (Iterator i = mPNames.keySet().iterator(); i.hasNext(); ) {
                  String name = (String) i.next();
                  P p = (P) mPNames.get(name);

                  m.addBodyLine(p.mVar + " = new Prop();");
                  m.addBodyLine(p.mVar +
                        ".setName(\"" + name + "\");");
                  m.addBodyLine(p.mVar +
                        ".setPropType(" + p.mPT.mVar + ");");
                  m.addBodyLine("");
            }

            String s = commaSeparatedList(mPNames.values().iterator());
            m.addBodyLine("mScreenplay.setProps(new Prop[] {" + s + "});");
      }

      private void makeDefineGroups() {
            Method m = createPrivateVoidMethod("defineGroups");

            for (Iterator i = mGNames.keySet().iterator(); i.hasNext(); ) {
                  String name = (String) i.next();
                  G g = (G) mGNames.get(name);

                  m.addBodyLine(g.mVar + " = new Group();");
                  m.addBodyLine(g.mVar +
                        ".setName(\"" + name + "\");");

                  int size = g.mMembers.size();
                  List l = new ArrayList(size);
                  for (int j = 0; j < size; j++) {
                        l.add(((C) mCNames.get(g.mMembers.get(j))).mVar);
                  }
                  String s = commaSeparatedList(l.iterator());
                  m.addBodyLine(g.mVar + ".setMembers(new StageObject[] {" + s +
"});");
                  m.addBodyLine("");
            }

            String s = commaSeparatedList(mGNames.values().iterator());
            m.addBodyLine("mScreenplay.setGroups(new Group[] {" + s + "});");
      }

      private void makeDefineScenes() throws EmuseException {
            Method m = createPrivateVoidMethod("defineScenes");

            for (int i = 0, n = mScenes.size(); i < n; i++) {
                  m.addBodyLine("mScreenplay.addScene(createScene" + i + "());");
                  makeCreateScene(i);
            }
      }

      private void makeCreateScene(int iSceneIndex) throws EmuseException {
            S s = (S) mScenes.get(iSceneIndex);

            Method m = createPrivateVoidMethod("createScene" + iSceneIndex);
            m.setReturnType("Scene");

            m.addBodyLine("Scene s = new Scene();");
            m.addBodyLine("s.setSetting(" +
                  ((Set) mSetNames.get(s.mSetting)).mVar + ");");
            m.addBodyLine("");
```

```
	for (int i = 0, n = s.mActions.size(); i < n; i++) {
		Action a = (Action) s.mActions.get(i);
		String aVar = "a" + i;
		String subj = resolveObjectVarName(a.mSubject);
		String dirObj;
		String indirObj;

		// get direct object
		if (a.mDirObj != null && a.mDirObj.startsWith("\"") &&
				a.mDirObj.endsWith("\"")) {
			dirObj = a.mDirObj;
		} else {
			dirObj = resolveObjectVarName(a.mDirObj);
		}

		// get indirect object
		indirObj = resolveObjectVarName(a.mIndirObj);

		// generate lines
		m.addBodyLine("Action " + aVar + " = new " +
			StringUtils.toPascalCase(a.mVerb) + "Action();");
		m.addBodyLine(aVar + ".setRunOrder(" + a.mRunOrder + ");");
		StringBuffer sb = new StringBuffer(150);
		sb.append(aVar).append(".initialize(").append(subj).append(", ")
			.append(dirObj).append(", ");
		if (a.mPrep == null || a.mPrep.equals("null")) {
			sb.append("null");
		} else {

sb.append("Preposition.valueOf(\"").append(a.mPrep).append("\")");
		}
		sb.append(", ").append(indirObj).append(", ");
		if (a.mAdverb == null || a.mAdverb.equals("null")) {
			sb.append("null");
		} else {

sb.append("Adverb.valueOf(\"").append(a.mAdverb).append("\")");
		}
		sb.append(");");

		m.addBodyLine(sb.toString());
		m.addBodyLine("s.addAction(" + aVar + ");");
		m.addBodyLine("");
	}

	m.addBodyLine("s.sealActions();");
	m.addBodyLine("return s;");
}

private String resolveObjectVarName(String iNameInScript)
		throws EmuseException {
	if (iNameInScript == null || iNameInScript.equals("null")) {
		return null;
	}

	// check to see if iNameInScript is a built-in object
	if (iNameInScript.equals("left")) {
		return "Stage.LEFT";
	} else if (iNameInScript.equals("right")) {
		return "Stage.RIGHT";
	} else if (iNameInScript.equals("center")) {
		return "Stage.CENTER";
```

```
	} else if (iNameInScript.equals("stage")) {
		return "Stage.CENTER";
	}

	C c1 = (C) mCNames.get(iNameInScript);
	if (c1 == null) {
		P p = (P) mPNames.get(iNameInScript);
		if (p == null) {
			throw new EmuseException("Cannot resolve object name: " +
				iNameInScript);
		} else {
			return p.mVar;
		}
	} else {
		return c1.mVar;
	}
}

private String commaSeparatedList(Iterator iItr) {
	StringBuffer sb = new StringBuffer(100);

	if (iItr.hasNext()) {
		sb.append(iItr.next());

		while (iItr.hasNext()) {
			sb.append(", ").append(iItr.next());
		}
	}

	return sb.toString();
}

private static class CT {
	String mVar;
	String mHead;
	String mArm;
	String mLeg;
	String mTorso;
	String mGender;
	String mHeight;
	String mBuild;

	public String toString() {
		return mVar;
	}
}

private static class PT {
	String mVar;
	String mImage;
	String mScale;

	public String toString() {
		return mVar;
	}
}

private static class Set {
	String mVar;
	String mImage;
	String mSound;

	public String toString() {
```

```
                return mVar;
        }
    }

    private static class C {
        String mVar;
        CT mCT;

        public String toString() {
            return mVar;
        }
    }

    private static class P {
        String mVar;
        PT mPT;

        public String toString() {
            return mVar;
        }
    }

    private static class G {
        String mVar;
        List mMembers;

        public G() {
            mMembers = new ArrayList();
        }

        public String toString() {
            return mVar;
        }
    }

    private static class S {
        String mSetting;
        List mActions;

        public S() {
            mActions = new ArrayList(100);
        }
    }

    private static class Action {
        String mSubject;
        String mVerb;
        String mDirObj;
        String mPrep;
        String mIndirObj;
        String mAdverb;
        int mRunOrder;
    }
}
```

```
package emuse.compiler.javamodel;

import java.io.*;
import java.util.*;

/**
 * Represents the structure of a Java language source file in its entirety.
 *
 * @author Mark Ayzenshtat
 */
public class JavaSource implements Commentable {
	private String mPackage;
	private List mImports;
	private List mClasses;
	private Comment mComment;

	public JavaSource() {
		mImports = new ArrayList();
		mClasses = new ArrayList();
	}

	public String getPackage() {
		return mPackage;
	}

	public void setPackage(String iPackage) {
		mPackage = iPackage;
	}

	public void addImport(String iImport) {
		mImports.add(iImport);
	}

	public List getImports() {
		return Collections.unmodifiableList(mImports);
	}

	public void addClass(Class iClass) {
		mClasses.add(iClass);
	}

	public List getClasses() {
		return Collections.unmodifiableList(mClasses);
	}

	public void setComment(Comment iC) {
		mComment = iC;
	}

	public Comment getComment() {
		return mComment;
	}

	public String toString() {
		try {
			StringWriter sw = new StringWriter();
			JavaSourceWriter sourceWriter = new StandardJavaSourceWriter(sw);
			sourceWriter.writeJavaSource(this);
			sourceWriter.close();
			String value = sw.toString();
			sw.close();
			return value;
		} catch (Exception ex) {
```

```
                        throw new RuntimeException(
                                "Could not output Java source as String: ", ex);
                }
        }
}
```

```
package emuse.compiler.javamodel;

import java.io.*;

import emuse.EmuseException;

/**
 * Writes Java source code encapsulated in a <code>JavaSource</code> object
 * to a target <code>Writer</code>.  Formatting is left to concrete
 * implementations.
 *
 * @author Mark Ayzenshtat
 */
public abstract class JavaSourceWriter extends FilterWriter {
	/**
	 * Creates a <code>JavaSourceWriter</code> that outputs to the target Writer.
	 *
	 * @param iOut the target <code>Writer</code>
	 * @see java.io.FilterWriter#FilterWriter(Writer)
	 */
	protected JavaSourceWriter(Writer iOut) {
		super(iOut);
	}

	/**
	 * Writes a Java source code object to the underlying Writer.
	 *
	 * @param iSource the source code object to output
	 * @throws IOException any exception thrown by the underlying Writer
	 */
	public abstract void writeJavaSource(JavaSource iSource)
		throws IOException;
}
```

```
package emuse.runtime.actions;

import emuse.runtime.model.*;

public class JumpsAction extends AbstractAction implements Action {
	public JumpsAction() {

	}

	public void initialize(StageObject iSubject, StageObject iDirectObject,
			Preposition iPrep, StageObject iIndirectObject, Adverb iAdverb) {
		// do nothing
	}
}
```

```
package emuse.runtime.anim;

/**
 * @author vlad
 *
 * Knob.java
 * Apr 9, 2003
 */
public class Knob {
	protected double mValue = 0;
	protected double mScale, mOffset;

	public Knob() {
		this(1, 0);
	}

	public Knob(double val) {
		this(1, 0);
		this.mValue = val;
	}

	public Knob(double scale, double offset) {
		this.mScale = scale;
		this.mOffset = offset;
	}

	/**
	 * @return
	 */
	public double getValue() {
		return mValue;
	}

	/**
	 * @param knob
	 */
	public void setValue(double knob) {
		//if (knob > 1 || knob < -1)
			//throw new IllegalArgumentException("Knob value out of bounds");

		this.mValue = knob * mScale + mOffset;
	}

	/**
	 * @return
	 */
	public double getOffset() {
		return mOffset;
	}

	/**
	 * @return
	 */
	public double getScale() {
		return mScale;
	}

	/**
	 * @param d
	 */
	public void setOffset(double d) {
		mOffset = d;
	}
```

```
	/**
	 * @param d
	 */
	public void setScale(double d) {
		mScale = d;
	}

	public void vary(double start, double end) {
		setOffset(start);
		setScale(end - start);
	}

	/**
	 *
	 */
	public void reset() {
		setOffset(0);
		setScale(1);
		setValue(0);
	}
}
```

```
package emuse.runtime.actions;

import emuse.runtime.anim.Knob;

/**
 * @author vlad
 *
 * KnobAction.java
 * Apr 9, 2003
 */
public class KnobAction extends AbstractAction {
	double mElapsed = 0;
	Knob mKnob;

	/**
	 * Constructor KnobAction.
	 * @param knob
	 */
	public KnobAction(Knob knob) {
		this.mKnob = knob;
	}


	public boolean step(double secs) {
		mElapsed += secs;
		mKnob.setValue(Math.sin(mElapsed));
		return false;
	}

}
```

```
package emuse.compiler.javamodel;

import java.util.*;

/**
 * A line (slash-slash) comment.
 *
 * @author Mark Ayzenshtat
 */
public class LineComment extends Comment {
	public LineComment(String iText) {
		super();
		buildLinesFromText(linesForText(iText));
	}

	public LineComment(String[] iLines) {
		super();
		buildLinesFromText(Arrays.asList(iLines));
	}

	private void buildLinesFromText(List iLines) {
		int numLines = iLines.size();
		mLines = new String[numLines];

		for (int i = 0; i < numLines; i++) {
			mLines[i] = "// " + iLines.get(i);
		}
	}
}
```

```
package emuse.compiler.javamodel;

import java.util.*;

/**
 * Represents a Java method.
 *
 * @author Mark Ayzenshtat
 */
public class Method implements Commentable, HasModifiers {
	private String mName;
	private Comment mComment;
	private String mReturnType;
	private List mArguments;
	private List mExceptions;
	private Set mMiscModifiers;
	private AccessModifier mAccess;
	private int mCurrentIndentLevel;
	private List mBodyLines;

	public Method() {
		mArguments = new ArrayList();
		mExceptions = new ArrayList();
		mMiscModifiers = new HashSet();
		mBodyLines = new ArrayList();
		mAccess = AccessModifier.kDefault;
		mCurrentIndentLevel = 0;
	}

	public String getReturnType() {
		return mReturnType;
	}

	public void setReturnType(String iType) {
		mReturnType = iType;
	}

	public void setComment(Comment iC) {
		mComment = iC;
	}

	public Comment getComment() {
		return mComment;
	}

	public void setName(String iName) {
		mName = iName;
	}

	public String getName() {
		return mName;
	}

	public void addArgument(Field iF) {
		mArguments.add(iF);
	}

	public List getArguments() {
		return Collections.unmodifiableList(mArguments);
	}

	public void addException(String iException) {
		mExceptions.add(iException);
```

```
}

public List getExceptions() {
	return Collections.unmodifiableList(mExceptions);
}

public void setAccessModifier(AccessModifier iAccess) {
	mAccess = iAccess;
}

public AccessModifier getAccessModifier() {
	return mAccess;
}

public void addModifier(MiscModifier iModifier) {
	mMiscModifiers.add(iModifier);
}

public Set getModifiers() {
	return Collections.unmodifiableSet(mMiscModifiers);
}

public void addBodyLine(String iLine) {
	mBodyLines.add(new BodyLine(iLine, mCurrentIndentLevel));
}

public List getBody() {
	return Collections.unmodifiableList(mBodyLines);
}

public void indent() {
	mCurrentIndentLevel++;
}

public void outdent() {
	if (mCurrentIndentLevel == 0) {
		return;
	}

	mCurrentIndentLevel--;
}

public static Method newMainMethod() {
	Method m = new Method();

	m.setAccessModifier(AccessModifier.kPublic);
	m.addModifier(MiscModifier.kStatic);
	m.setReturnType("void");
	m.setName("main");
	m.addArgument(new Field("String[]", "args"));

	return m;
}

public static class BodyLine {
	private String mText;
	private int mIndentLevel;

	public BodyLine(String iText, int iIndentLevel) {
		mText = iText;
		mIndentLevel = iIndentLevel;
	}
```

```
		public String getText() {
			return mText;
		}

		public int getIndentLevel() {
			return mIndentLevel;
		}
	}
}
```

```
package emuse.compiler.javamodel;

/**
 * A typesafe enumeration for miscellaneous class, method, and field modifiers.
 * Typesafe enum elements can be compared using the <code>==</code> operator.
 *
 * @author Mark Ayzenshtat
 */
public class MiscModifier {
	/** The <code>abstract</code> modifier. */
	public static final MiscModifier kAbstract = new MiscModifier("abstract");

	/** The <code>static</code> modifier. */
	public static final MiscModifier kStatic = new MiscModifier("static");

	/** The <code>final</code> modifier. */
	public static final MiscModifier kFinal = new MiscModifier("final");

	private final String mKeyword;

	private MiscModifier(String iKeyword) {
		mKeyword = iKeyword;
	}

	/**
	 * Returns the <code>String</code> equivalent of this modifier.
	 */
	public String toString() {
		return mKeyword;
	}
}
```

```
package emuse.compiler;

import java.util.List;
import emuse.EmuseException;

/**
 * <p>
 * An output generator is an abstraction for a class that generates output for
 * an eMuse screenplay in a specific target format.  The default implementation
 * of <code>OutputGenerator</code> is <code>JavaOutputGenerator</code>.
 * </p>
 *
 * <p>
 * The <code>startOutput()</code> method must be called before any output is
 * generated, and the <code>finishOutput()</code> method must be called
 * afterward.
 * </p>
 *
 * @author Mark Ayzenshtat
 */
public interface OutputGenerator {
	/**
	 * Begins generating output.  This method must be called to initialize this
	 * output generator before any <code>outputXXX()</code> methods are called.
	 */
	public void startOutput();

	/**
	 * Outputs the title of the screenplay.
	 *
	 * @param iTitle the title
	 */
	public void outputTitle(String iTitle);

	/**
	 * Outputs an author of the screenplay.  This method may be called more than
	 * once if the screenplay has more than one author.
	 *
	 * @param iAuthor an author
	 */
	public void outputAuthor(String iAuthor);

	/**
	 * Begins to output a CharType.
	 *
	 * @param iLabel the label used to identify the CharType
	 */
	public void outputCharType(String iLabel);

	/**
	 * Outputs a CharType head parameter.
	 *
	 * @param iImageFile the image file
	 */
	public void outputCharTypeHead(String iImageFile);

	/**
	 * Outputs a CharType arm parameter.
	 *
	 * @param iImageFile the image file
	 */
	public void outputCharTypeArm(String iImageFile);
```

```
/**
 * Outputs a CharType leg parameter.
 *
 * @param iImageFile the image file
 */
public void outputCharTypeLeg(String iImageFile);

/**
 * Outputs a CharType torso parameter.
 *
 * @param iImageFile the image file
 */
public void outputCharTypeTorso(String iImageFile);

/**
 * Outputs the CharType body parameters.
 */
public void outputCharTypeBody(String iGender, String iHeight,
       String iBuild);

/**
 * Begins to output a PropType.
 *
 * @param iLabel the label used to identify the PropType
 */
public void outputPropType(String iLabel);

/**
 * Outputs a PropType image parameter.
 *
 * @param iImageFile the image file
 */
public void outputPropTypeImage(String iImageFile);

/**
 * Outputs a PropType scale parameter.
 *
 * @param iImageFile the scale
 */
public void outputPropTypeScale(String iScale);

/**
 * Outputs a setting.
 *
 * @param iLabel the label used to identify the setting
 */
public void outputSetting(String iLabel);

/**
 * Outputs the image field of a setting.
 *
 * @param iImageFile the image file
 */
public void outputSettingImage(String iImageFile);

/**
 * Outputs the sound field of a setting.
 *
 * @param iSoundFile the sound file
 */
public void outputSettingSound(String iSoundFile);

/**
```

```
         * Outputs a character definition.
         *
         * @param iName the character's name
         * @param iCharType the CharType
         */
        public void outputCharacter(String iName, String iCharType);

        /**
         * Outputs a prop definition.
         *
         * @param iName the character's name
         * @param iPropType the CharType
         */
        public void outputProp(String iName, String iPropType);

        /**
         * Outputs a group definition.
         *
         * @param iName the name of the group
         */
        public void outputGroup(String iName);

        /**
         * Outputs a group member.  This method may be called more than
         * once if the group has more than one member.
         * @param iName the group member
         */
        public void outputGroupMember(String iName);

        /**
         * Outputs a new scene.
         *
         * @param iSetting the setting of the scene
         */
        public void outputScene(String iSetting);

        /**
         * Outputs an action.
         *
         * @param iSubject the subject
         * @param iVerb the verb
         * @param iDirObj the direct object
         * @param iPrep the preposition
         * @param iIndirObj the indirect object
         * @param iAdverb the adverb
         */
        public void outputAction(String iSubject, String iVerb, String iDirObj,
               String iPrep, String iIndirObj, String iAdverb);

        /**
         * Completes output generation.  This method must be called to finalize
         * output generation after all <code>outputXXX()</code> methods have been
         * called.
         */
        public void finishOutput() throws EmuseException;
}
```

```
package emuse.runtime.anim;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.Shape;
import java.awt.geom.*;

import emuse.runtime.model.StageObject;

/**
 * A rectangular object, with a specified color.
 *
 * @author vlad
 *
 * PivotedShape.java
 * Mar 11, 2003
 */
public class PivotedShape extends StageObject {
	protected Shape shape;
	protected Color color;

	public PivotedShape(Shape s, Color c) {
		this.shape = s;
		this.color = c;
	}

	/**
	 * Constructor for PivotedShape.
	 */
	public PivotedShape(double x, double y, double w, double h, Color c) {
		this.shape = new Rectangle2D.Double(x, y, w, h);
		this.color = c;
	}

	protected void drawObject(Graphics2D g) {
		g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
			RenderingHints.VALUE_ANTIALIAS_ON);
		g.setColor(color);
		g.fill(shape);
	}

	/**
	 * @return
	 */
	public Knob getKnob() {
		return theta;
	}

	/**
	 * @param k
	 */
	public void setKnob(Knob k) {
		this.theta = k;
	}

}
```

```
package emuse.runtime.model;

import java.util.*;

/**
 * A typesafe enumeration for prepositions.  Typesafe enum elements
 * can be safely tested for equality using the <code>==</code> operator.
 *
 * @author Mark Ayzenshtat
 */
public class Preposition {
	private static Map kPrivateValues = new HashMap(100);
	/** A collection of all values in this enum. */
	public static final Collection kValues;

	/** Above. */
	public static final Preposition kAbove = new Preposition("above");

	/** Across. */
	public static final Preposition kAcross = new Preposition("across");

	/** After. */
	public static final Preposition kAfter = new Preposition("after");

	/** Following. */
	public static final Preposition kFollowing = makeEqualTo(kAfter, "following");

	/** Against. */
	public static final Preposition kAgainst = new Preposition("against");

	/** Around. */
	public static final Preposition kAround = new Preposition("around");

	/** At. */
	public static final Preposition kAt = new Preposition("at");

	/** Before. */
	public static final Preposition kBefore = new Preposition("before");

	/** Behind. */
	public static final Preposition kBehind = new Preposition("behind");

	/** Below. */
	public static final Preposition kBelow = new Preposition("below");

	/** Beneath. */
	public static final Preposition kBeneath = makeEqualTo(kBelow, "beneath");

	/** Under. */
	public static final Preposition kUnder = makeEqualTo(kBelow, "under");

	/** Underneath. */
	public static final Preposition kUnderneath = makeEqualTo(kBelow,
"underneath");

	/** Beside. */
	public static final Preposition kBeside = new Preposition("beside");

	/** Besides. */
	public static final Preposition kBesides = makeEqualTo(kBeside, "besides");

	/** Between. */
	public static final Preposition kBetween = new Preposition("between");
```

```
/** By. */
public static final Preposition kBy = new Preposition("by");

/** Near. */
public static final Preposition kNear = makeEqualTo(kBy, "near");

/** Down. */
public static final Preposition kDown = new Preposition("down");

/** Except. */
public static final Preposition kExcept = new Preposition("except");

/** Excepting. */
public static final Preposition kExcepting = makeEqualTo(kExcept, "excepting");

/** Excluding. */
public static final Preposition kExcluding = makeEqualTo(kExcept, "excluding");

/** For. */
public static final Preposition kFor = new Preposition("for");

/** From. */
public static final Preposition kFrom = new Preposition("from");

/** In. */
public static final Preposition kIn = new Preposition("in");

/** Inside. */
public static final Preposition kInside = makeEqualTo(kIn, "inside");

/** Into. */
public static final Preposition kInto = makeEqualTo(kIn, "into");

/** Of. */
public static final Preposition kOf = new Preposition("of");

/** Off. */
public static final Preposition kOff = new Preposition("off");

/** On. */
public static final Preposition kOn = new Preposition("on");

/** Onto. */
public static final Preposition kOnto = makeEqualTo(kOn, "onto");

/** Outside. */
public static final Preposition kOutside = new Preposition("outside");

/** Over. */
public static final Preposition kOver = new Preposition("over");

/** Through. */
public static final Preposition kThrough = new Preposition("through");

/** To. */
public static final Preposition kTo = new Preposition("to");

/** Toward. */
public static final Preposition kToward = new Preposition("toward");

/** Towards. */
public static final Preposition kTowards = makeEqualTo(kToward, "towards");
```

```
	/** Up. */
	public static final Preposition kUp = new Preposition("up");

	/** With. */
	public static final Preposition kWith = new Preposition("with");

	static {
		kValues = Collections.unmodifiableCollection(kPrivateValues.values());

	}

	private final String mValue;

	private Preposition(String iValue) {
		mValue = iValue;
		kPrivateValues.put(iValue, this);
	}

	private static Preposition makeEqualTo(Preposition iOther, String iValue) {
		kPrivateValues.put(iValue, iOther);
		return iOther;
	}

	/**
	 * Returns the <code>String</code> equivalent of this preposition.
	 */
	public String toString() {
		return mValue;
	}

	/**
	 * Returns the <code>Preposition</code> constant equivalent of the
	 * given string.
	 */
	public static Preposition valueOf(String iP) {
		return (Preposition) kPrivateValues.get(iP);
	}
}
```

```
package emuse.runtime.model;

import java.awt.Graphics2D;

/**
 * @author Mark Ayzenshtat
 */
public class Prop extends StageObject {
	private PropType mPropType;

	public PropType getPropType() {
		return mPropType;
	}

	public void setPropType(PropType propType) {
		mPropType = propType;
	}

	protected void drawObject(Graphics2D g) {}

	public double getStageX() {
		return 0;	// stub
	}
}
```

```
package emuse.runtime.model;

import java.awt.image.BufferedImage;

/**
 * @author Mark Ayzenshtat
 */
public class PropType {
	private BufferedImage mAppearance;
	private Scale mScale;

	public BufferedImage getAppearance() {
		return mAppearance;
	}

	public Scale getScale() {
		return mScale;
	}

	public void setAppearance(BufferedImage appearance) {
		mAppearance = appearance;
	}

	public void setScale(Scale scale) {
		mScale = scale;
	}
}
```

```
package emuse;

import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.InputStream;
import java.util.LinkedHashMap;
import java.util.Map;

import javax.imageio.ImageIO;
import javax.sound.sampled.*;

/**
 * Retrieves images and sounds, caches the most recently accessed files.
 *
 * @author Kristina Holst
 */
public class ResourceManager {

	private static final ResourceManager kInstance = new ResourceManager();
	private static final int kMaxCapacity = 100;

	/* Maps path to BufferedImage for images and path to Clip for sounds */
	private Map mResources;

	private ResourceManager() {
		mResources = new LinkedHashMap((int) (kMaxCapacity / .75), .75f, true) {
			protected boolean removeEldestEntry(Map.Entry iEldest) {
				return size() > kMaxCapacity;
			}
		};
	}

	public static ResourceManager getInstance() {
		return kInstance;
	}

	/**
	 * Returns a BufferedImage for the given file (.gif, .jpg, or .png)
	 *
	 * @param iPath
	 * @return
	 * @throws EmuseException
	 */
	public BufferedImage getImage(String iPath) throws EmuseException {
		BufferedImage image = (BufferedImage) mResources.get(iPath);

		if (image == null) {
			image = loadImage(iPath);
		}

		return image;
	}

	/**
	 * Returns a Clip for the given file (.wav or .aiff)
	 *
	 * NOTE:  It is necessary to call System.exit() at the end of your program
	 *                          if you have used this method to obtain a Clip.  This
is a documented
	 *                          bug with Java Sound, and if you don't do this, your
program won't
	 *                          terminate.
	 *
```

```
	 * @param iPath
	 * @return
	 * @throws EmuseException
	 */
	public Clip getSound(String iPath) throws EmuseException {
		Clip sound = (Clip) mResources.get(iPath);

		if (sound == null) {
			sound = loadSound(iPath);
		}

		return sound;
	}

	public boolean canLoadResource(String iPath) {
		return (ClassLoader.getSystemResourceAsStream(iPath) != null);
	}

	/**
	 * Loads and returns the requested image.
	 *
	 * @param iPath path of the image file
	 * @return the BufferedImage
	 */
	private BufferedImage loadImage(String iPath) throws EmuseException {
		BufferedImage image = null;

		InputStream stream = ClassLoader.getSystemResourceAsStream(iPath);

		if (stream != null) {
			try {
				image = ImageIO.read(stream);
				stream.close();
			} catch (IOException ex) {
				throw new EmuseException("Cannot load image: " + iPath,
ex);
			}

			// add to cache
			mResources.put(iPath, image);
		} else {
			throw new EmuseException("Cannot find file: " + iPath);
		}

		return image;
	}

	/**
	 * Loads and returns the requested sound.
	 *
	 * @param iPath
	 * @return
	 * @throws EmuseException
	 */
	private Clip loadSound(String iPath) throws EmuseException {
		Clip sound = null;

		InputStream stream = ClassLoader.getSystemResourceAsStream(iPath);

		if (stream != null) {
			try {
				AudioInputStream audioStream =
AudioSystem.getAudioInputStream(stream);
```

```
                        AudioFormat format = audioStream.getFormat();
                        DataLine.Info info = new DataLine.Info(Clip.class, format);
                        sound = (Clip) AudioSystem.getLine(info);
                        sound.open(audioStream);
                  } catch (Exception ex) {
                        throw new EmuseException("Cannot load sound: " + iPath,
ex);
                  }

                  // add to cache
                  mResources.put(iPath, sound);
            } else {
                  throw new EmuseException("Cannot find file: " + iPath);
            }

            return sound;
      }
}
```

```
package emuse.runtime.model;

import java.util.*;

/**
 * A typesafe enumeration for image scales.  Typesafe enum elements
 * can be safely tested for equality using the <code>==</code> operator.
 *
 * @author Mark Ayzenshtat
 */
public class Scale {
	private static Map kPrivateValues = new HashMap(7);
	/** A collection of all values in this enum. */
	public static final Collection kValues;

	/** The "small" image scale. */
	public static final Scale kSmall = new Scale("small");

	/** The "medium" image scale. */
	public static final Scale kMedium = new Scale("medium");

	/** The "large" image scale. */
	public static final Scale kLarge = new Scale("large");

	static {
		kValues = Collections.unmodifiableCollection(kPrivateValues.values());

	}

	private final String mValue;

	private Scale(String iValue) {
		mValue = iValue;
		kPrivateValues.put(iValue, this);
	}

	/**
	 * Returns the <code>String</code> equivalent of this image scale.
	 */
	public String toString() {
		return mValue;
	}

	/**
	 * Returns the <code>Scale</code> constant equivalent of the given string.
	 */
	public static Scale valueOf(String iScale) {
		return (Scale) kPrivateValues.get(iScale);
	}
}
```

```
package emuse.runtime.model;

import java.util.*;
import emuse.runtime.actions.*;

/**
 * A scene consists of a list of actions that occur at a particular setting.
 *
 * @author Mark Ayzenshtat
 */
public class Scene {
	private Setting mSetting;
	private List mActions;

	public Scene() {
		mActions = new ArrayList();
	}

	/**
	 * @return List
	 */
	public List getActions() {
		return Collections.unmodifiableList(mActions);
	}

	/**
	 * @return Setting
	 */
	public Setting getSetting() {
		return mSetting;
	}

	public void addAction(Action iA) {
		mActions.add(iA);
	}

	public void sealActions() {
		Collections.sort(mActions, AbstractAction.kRunOrderComparator);
	}

	/**
	 * Sets the setting.
	 * @param setting The setting to set
	 */
	public void setSetting(Setting iSetting) {
		mSetting = iSetting;
	}
}
```

```
package emuse.runtime.anim;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

import emuse.runtime.model.Action;
import emuse.runtime.model.Scene;
import emuse.runtime.model.Screenplay;
import emuse.runtime.model.Stage;
import emuse.runtime.model.StageObject;


/**
 * This class is used to represent the state of the play
 * as it is being performed.
 *
 * @author Vlad
 * @author Mark
 *
 * SceneManager.java
 * Mar 11, 2003
 */
public final class SceneManager {
	Stage rootObject;   // root node of scene tree
	LinkedList actions;        // list of actions currently performed
	Screenplay play;
	Scene currentScene;
	AffineTransform scale;

	int sceneIndex = 0;
	int actionIndex = 0;
	int runIndex = -1;

	int numScenes;

	long timeOfLastStep = 0;
	double speedScaleFactor = 1.0;

	public SceneManager(Screenplay sp) {
		rootObject = Stage.getInstance();
		actions = new LinkedList();
		play = sp;
		numScenes = play.getScenes().size();
		scale = new AffineTransform();
	}

	void addAction(Action a) {
		actions.add(a);
	}

	/**
	 * Method step.
	 *
	 * @return true if playback of script is complete
	 */
	public boolean step(double iUnscaledTimeDelta) {
		Action a;
		double stepLength;
```

```
		boolean done = false, haveActions = false;

		stepLength = iUnscaledTimeDelta * getSpeedScaleFactor() / 1000;

		Iterator iter = actions.iterator();
		while (iter.hasNext()) {
			a = (Action) iter.next();
			if (a.step(stepLength)) {
				iter.remove();	/* remove this action if it is
completed */
			}
			haveActions = true;
		}

		if (!haveActions) {
			done = updateActionList();
		}

		return !haveActions && done;
	}

	/**
	 * Grabs the next run order set of actions
	 */
	private final boolean updateActionList() {
		if (sceneIndex >= numScenes) {
			return true;
		}

		if (currentScene == null) {
			currentScene = (Scene) play.getScenes().get(sceneIndex);
			rootObject.setSetting(currentScene.getSetting());
		}

		List actions = currentScene.getActions();
		int numActions = actions.size();

		if (actionIndex >= numActions) {
			actionIndex = 0;
			runIndex = -1;
			sceneIndex++;
			currentScene = null;
		}
		else {
			runIndex++;
			Action action = (Action)actions.get(actionIndex);

			while (action.getRunOrder() == runIndex)
			{
				addAction(action);
				actionIndex++;
				if (actionIndex >= numActions)
					break;
				action = (Action)actions.get(actionIndex);
			}
		}

		return false;
	}


	public void resumeFromPause() {
		// Vlad, why is this method necessary?
```

```
		/*
		this.timeOfLastStep = System.currentTimeMillis();
		//step();
		 */
	}

	public void drawScene(Graphics g) {
		rootObject.draw(g);
	}

	/**
	 * Method addSprite.
	 * @param d
	 */
	void addSprite(StageObject so) {
		rootObject.addDescendent(so);
	}

	/**
	 * Returns the speedScaleFactor.
	 * @return double
	 */
	public double getSpeedScaleFactor() {
		return speedScaleFactor;
	}

	/**
	 * Sets the speedScaleFactor.
	 * @param speedScaleFactor The speedScaleFactor to set
	 */
	void setSpeedScaleFactor(double speedScaleFactor) {
		if (speedScaleFactor <= 0.01) {
			speedScaleFactor = 0.01;
		}

		this.speedScaleFactor = speedScaleFactor;
	}

}
```

```
package emuse.runtime.model;

import java.util.*;
import emuse.runtime.model.Character;

/**
 * @author Vlad Shchogolev
 * @author Mark Ayzenshtat
 */
public class Screenplay {
	private String mTitle;
	private String[] mAuthors;
	private CharType[] mCharacterTypes;
	private PropType[] mPropTypes;
	private Setting[] mSettings;
	private Character[] mCharacters;
	private Prop[] mProps;
	private Group[] mGroups;
	private List mScenes;

	public Screenplay() {
		mScenes = new ArrayList();
	}

	public String[] getAuthors() {
		return mAuthors;
	}

	public CharType[] getCharacterTypes() {
		return mCharacterTypes;
	}

	public String getTitle() {
		return mTitle;
	}

	public void setAuthors(String[] iAuthors) {
		mAuthors = iAuthors;
	}

	public void setCharacterTypes(CharType[] iCharacterTypes) {
		mCharacterTypes = iCharacterTypes;
	}

	public void setTitle(String title) {
		mTitle = title;
	}

	public PropType[] getPropTypes() {
		return mPropTypes;
	}

	public void setPropTypes(PropType[] iPropTypes) {
		mPropTypes = iPropTypes;
	}

	public Setting[] getSettings() {
		return mSettings;
	}

	public void setSettings(Setting[] settings) {
		mSettings = settings;
	}
```

```
	/**
	 * @return Character[]
	 */
	public Character[] getCharacters() {
		return mCharacters;
	}

	/**
	 * @return Group[]
	 */
	public Group[] getGroups() {
		return mGroups;
	}

	/**
	 * @return Prop[]
	 */
	public Prop[] getProps() {
		return mProps;
	}

	/**
	 * @return Scene[]
	 */
	public List getScenes() {
		return Collections.unmodifiableList(mScenes);
	}

	/**
	 * Sets the characters.
	 * @param characters The characters to set
	 */
	public void setCharacters(Character[] characters) {
		mCharacters = characters;
	}

	/**
	 * Sets the groups.
	 * @param groups The groups to set
	 */
	public void setGroups(Group[] groups) {
		mGroups = groups;
	}

	/**
	 * Sets the props.
	 * @param props The props to set
	 */
	public void setProps(Prop[] props) {
		mProps = props;
	}

	/**
	 * Sets the scenes.
	 * @param scenes The scenes to set
	 */
	public void addScene(Scene iScene) {
		mScenes.add(iScene);
	}
}
```

```
package emuse.runtime.model;

import java.awt.image.BufferedImage;
import javax.sound.sampled.Clip;

/**
 * @author Mark Ayzenshtat
 */
public class Setting {
	private BufferedImage mImage;
	private Clip mSound;

	public BufferedImage getImage() {
		return mImage;
	}

	public Clip getSound() {
		return mSound;
	}

	public void setImage(BufferedImage image) {
		mImage = image;
	}

	public void setSound(Clip sound) {
		mSound = sound;
	}
}
```

```
/*
 * Created on Apr 22, 2003
 */
package emuse.runtime.anim;

import java.awt.Graphics2D;
import java.util.List;

import emuse.runtime.model.Stage;
import emuse.runtime.model.StageObject;

/**
 * This is a pseudo-object. It represents a location within a "real"
 * object. Whenever an object is made a descendent of a SnapPoint,
 * the object is made a descendent of the "real" object instead.
 *
 * @author Vladislav
 */
public abstract class SnapPoint extends StageObject {
	StageObject parentObject;

	public SnapPoint() {
		parentObject = Stage.getInstance();
	}

	public SnapPoint(StageObject so) {
		parentObject = so;
	}

	/* (non-Javadoc)
	 * @see emuse.runtime.model.StageObject#drawObject(java.awt.Graphics2D)
	 */
	protected void drawObject(Graphics2D g) {}


	/* (non-Javadoc)
	 * @see
emuse.runtime.model.StageObject#addDescendent(emuse.runtime.model.StageObject)
	 */
	public void addDescendent(StageObject so) {
		positionChild(so);
		parentObject.addDescendent(so);
	}

	/* (non-Javadoc)
	 * @see emuse.runtime.model.StageObject#getDescendents()
	 */
	public List getDescendents() {
		return parentObject.getDescendents();
	}

	/* (non-Javadoc)
	 * @see
emuse.runtime.model.StageObject#removeDescendent(emuse.runtime.model.StageObject)
	 */
	public void removeDescendent(StageObject so) {
		parentObject.removeDescendent(so);
	}

	public abstract void positionChild(StageObject so);

	public static class Left extends SnapPoint {
		public Left(StageObject so) { super(so); }
```

```
		public void positionChild(StageObject so) {
			so.setPivot(so.getWidth() / 2, parentObject.getHeight());
		}
	}

	public static class Right extends SnapPoint {
		public Right(StageObject so) { super(so); }
		public void positionChild(StageObject so) {
			so.setPivot(parentObject.getWidth() - so.getWidth() / 2,
parentObject.getHeight());
		}
	}

	public static class Center extends SnapPoint {
		public Center(StageObject so) { super(so); }
		public void positionChild(StageObject so) {
			so.setPivot(parentObject.getWidth()/2, parentObject.getHeight());
		}
	}
}
```

```
package emuse.runtime.actions;

import emuse.runtime.anim.SpeechBalloon;
import emuse.runtime.model.*;
import emuse.runtime.model.Character;

/**
 *
 * @author Vlad Shchogolev
 * @author Mark Ayzenshtat
 */

public class SpeaksAction extends AbstractAction implements Action {

	public static int kScrollLimit = 60;

	/**
	 * @param c
	 * @param string
	 */
	public SpeaksAction(Character c, String string) {
		initialize(c, string, null, null, null);
	}

	boolean firstTime = true;
	Character ch;
	String speechText;
	double duration = 0;
	SpeechBalloon sb;

	public SpeaksAction() {}

	public void initialize(StageObject iSubject, Object iDirectObject,
			Preposition iPrep, StageObject iIndirectObject, Adverb iAdverb) {
		this.ch = (Character) iSubject;
		this.speechText = (String) iDirectObject;
	}

	protected void onFirstStep() {
		sb = new SpeechBalloon(speechText, 400, 300);
		ch.getHead().addDescendent(sb);
		sb.yLoc.setValue(-ch.getHead().getHeight());
	}

	protected void onStep() {
		sb.timeElapsed(elapsed);

		if (isDone()) {
			ch.getHead().removeDescendent(sb);
		}
	}

	public boolean isDone() {
		int len = speechText.length();

		if (len < kScrollLimit) {
			return (elapsed > 2);
		} else {
			return sb.isDone();
		}
	}
}
```

```
/*
 * Created on Apr 17, 2003
 *
 * To change the template for this generated file go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
package emuse.runtime.anim;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.Shape;
import java.awt.font.FontRenderContext;
import java.awt.font.LineBreakMeasurer;
import java.awt.font.TextAttribute;
import java.awt.font.TextLayout;
import java.awt.geom.Rectangle2D;
import java.text.AttributedCharacterIterator;
import java.text.AttributedString;
import java.util.Hashtable;

import emuse.runtime.actions.SpeaksAction;
import emuse.runtime.model.StageObject;

/**
 * @author Vladislav
 * @author Mark
 */
public class SpeechBalloon extends StageObject {

	private static final Hashtable map = new Hashtable();
	static {
		Font font = new Font("Arial", Font.BOLD, 60);
		map.put(TextAttribute.FONT, font);
	}

	LineBreakMeasurer lineMeasurer;
	int paragraphStart;
	int paragraphEnd;

	Shape balloon;
	boolean firstTime = true;
	float width;
	float height;
	private float textLowerBound = Float.MAX_VALUE, scrollTextY;
	private int length;

	/**
	 * @param speechText
	 */
	public SpeechBalloon(String speechText, double width, double height) {
		this.width = (float) width;
		this.height = (float) height;
		initText(speechText);
		length = speechText.length();
	}

	/* (non-Javadoc)
	 * @see emuse.runtime.model.StageObject#drawObject(java.awt.Graphics2D)
	 */
	protected void drawObject(Graphics2D g) {

		if (firstTime) {
```

```
                balloon = new Rectangle2D.Double(
                        -25 - width/2, -50 - height, width+50, height+50);

                firstTime = false;
        }

        g.setColor(Color.orange);
        g.fill(balloon);
        g.setColor(Color.black);
        g.draw(balloon);

        g.setClip(balloon);
        drawText(g);
        g.setClip(null);
}

public void timeElapsed(double elapsed) {
        scrollTextY = (int) (elapsed * -40);
}

private void drawText(Graphics2D g) {
        float drawPosX;

        float drawPosY;
        if (length > SpeaksAction.kScrollLimit)
                drawPosY = scrollTextY;
        else
                drawPosY = -height;

        lineMeasurer.setPosition(paragraphStart);

        // Get lines from lineMeasurer until the entire
        // paragraph has been displayed.
        while (lineMeasurer.getPosition() < paragraphEnd) {

                // Retrieve next layout.
                TextLayout layout = lineMeasurer.nextLayout(width);

                // Move y-coordinate by the ascent of the layout.
                drawPosY += layout.getAscent();
                drawPosX = -width/2;

                // Draw the TextLayout at (drawPosX, drawPosY).
                layout.draw(g, drawPosX, drawPosY);

                // Move y-coordinate in preparation for next layout.
                drawPosY += layout.getDescent() + layout.getLeading();
        }

        textLowerBound = drawPosY;
}

private void initText(String string) {
        AttributedString text = new AttributedString(string, map);
        AttributedCharacterIterator paragraph = text.getIterator();
        paragraphStart = paragraph.getBeginIndex();
        paragraphEnd = paragraph.getEndIndex();

        // Create a new LineBreakMeasurer from the paragraph.
        lineMeasurer =
                new LineBreakMeasurer(
                        paragraph,
                        new FontRenderContext(null, false, false));
```

```
	}

	/* (non-Javadoc)
	 * @see emuse.runtime.model.StageObject#getStageX()
	 */
	public double getStageX() {
		return 0;
	}

	/**
	 * @return whether the text has been displayed
	 */
	public boolean isDone() {
		if (length > SpeaksAction.kScrollLimit)
			return textLowerBound < -height;
		else
			return true;
	}

}
```

```
package emuse.runtime.model;

import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;

import emuse.runtime.anim.AnimationFrame;
import emuse.runtime.anim.SnapPoint;

/**
 * "All the world's a stage"
 *           - the Bard
 *
 * @author vlad
 *
 * Stage.java
 * Mar 19, 2003
 */
public class Stage extends StageObject {

	private static final Stage kSingleton = new Stage();

	public static final StageObject LEFT = new SnapPoint.Left(kSingleton);
	public static final StageObject RIGHT = new SnapPoint.Right(kSingleton);
	public static final StageObject CENTER = new SnapPoint.Center(kSingleton);

	protected Setting setting;
	protected BufferedImage background;

	private Stage() {
		Dimension size = AnimationFrame.kSize;
		mWidth = size.getWidth();
		mHeight = size.getHeight();
	}

	protected void drawObject(Graphics2D g) {
		if (background != null)
			g.drawImage(background, null, 0, 0);
	}

	/**
	 * Returns the stage setting.
	 * @return Setting
	 */
	public Setting getSetting() {
		return setting;
	}

	/**
	 * Sets the stage setting.
	 * @param setting The setting to set
	 */
	public void setSetting(Setting setting) {
		this.setting = setting;

		BufferedImage bg = setting.getImage();
		background = new BufferedImage((int)mWidth, (int)mHeight, bg.getType());

		AffineTransform trans = new AffineTransform();

		double sx = mWidth / (double)bg.getWidth();
		double sy = mHeight / (double)bg.getHeight();
```

```
                trans.scale(sx, sy);

                Graphics2D g = background.createGraphics();
                g.drawImage(bg, trans, null);
        }

        /**
         * @return
         */
        public static Stage getInstance() {
                return kSingleton;
        }
}
```

```
package emuse.runtime.model;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

import emuse.runtime.anim.Knob;

/**
 * A stage object represents anything that moves around on the stage and
 * can be drawn.  It is defined by a 2D transformation matrix and scene graph,
 * which determine the position, orientation, and size of the object and all
 * of its children.
 *
 * @author Mark Ayzenshtat
 * @author Vlad Shchogolev
 */
public abstract class StageObject {
	/** The name of this stage object. */
	protected String mName;

	/** This stage object's 2D transformation matrix. */
	protected AffineTransform mTransform;

	/** Is this object on stage right now? */
	protected boolean mOnStage;

	/** The children of this stage object in the scene graph. */
	private LinkedList mDescendents;

	public Knob
		xLoc = new Knob(),
		yLoc = new Knob(),
		theta = new Knob(),
		scale = new Knob(1.0);

	protected double mWidth, mHeight;

	/**
	 * Constructs a stage object with an identity transformation matrix
	 * and no descendents.
	 */
	public StageObject() {
		mTransform = new AffineTransform();
		mDescendents = new LinkedList();
		mName = "";
	}

	/**
	 * Returns this stage object's name.
	 */
	public String getName() {
		return mName;
	}

	/**
	 * Assigns this stage object's name.
	 *
	 * @param iName the name
```

```
 */
public void setName(String iName) {
	mName = iName;
}

/**
 * Draws this object.  Subclasses should override this method instead of
 * <code>draw(Graphics)</code>.
 *
 * @param g the graphics object
 */
protected abstract void drawObject(Graphics2D g);

/**
 * Tells this object to update its <code>AffineTransform</code> to
 * reflect changes in its state.
 *
 * @param g the graphics object
 */
public void updateObject() {
	mTransform.setToTranslation(xLoc.getValue(), yLoc.getValue());
	mTransform.scale(scale.getValue(), scale.getValue());
	mTransform.rotate(theta.getValue());
}

/**
 * Draws this object and all of its children.  Subclasses should NOT override
 * this method.  Instead, override <code>drawObject(Graphics)</code>.
 *
 * @param g the graphics object
 */
public final void draw(Graphics g) {
	StageObject so;

	Graphics2D g2 = (Graphics2D) g;
	AffineTransform saveXform = g2.getTransform();

	// draw self
	updateObject();
	g2.transform(mTransform);
	drawObject(g2);

	// draw kids
	synchronized (mDescendents) {
		Iterator iter = mDescendents.iterator();
		while (iter.hasNext()) {
			so = (StageObject) iter.next();
			so.draw(g);
		}
	}

	g2.setTransform(saveXform);
}

/**
 * Adds a descendent/child to this stage object's node in the scene graph.
 * @param so the child to add
 */
public void addDescendent(StageObject so) {
	synchronized (mDescendents) {
		mDescendents.add(so);
	}
}
```

```
	/**
	 * @param so
	 */
	public void removeDescendent(StageObject so) {
		synchronized (mDescendents) {
			mDescendents.remove(so);
		}
	}

	/**
	 * Returns the Descendents.
	 */
	public List getDescendents() {
		return Collections.unmodifiableList(mDescendents);
	}

	/**
	 * Returns whether this stage object is currently on the stage.
	 */
	public boolean isOnStage() {
		return mOnStage;
	}

	/**
	 * Assigns whether this stage object is currently on the stage.
	 */
	public void setOnStage(boolean b) {
		mOnStage = b;
	}

	/**
	 * Returns this stage object's transformation matrix.
	 */
	private AffineTransform getTransform() {
		return mTransform;
	}

	/**
	 * @param d
	 * @param e
	 */
	public void setPivot(double x, double y) {
		xLoc.setValue(x);
		yLoc.setValue(y);
	}
	/**
	 * @return
	 */
	public double getHeight() {
		return mHeight;
	}

	/**
	 * @return
	 */
	public double getWidth() {
		return mWidth;
	}

}
```

```
package emuse.compiler.javamodel;

import java.io.*;
import java.util.Iterator;
import emuse.EmuseException;

/**
 * The standard implementation of <code>JavaSourceWriter</code>.
 *
 * @author Mark Ayzenshtat
 */
public class StandardJavaSourceWriter extends JavaSourceWriter {
	private static final int kDefaultNumIndentSpaces = 2;
	private static final String kEmptyString = "";

	private PrintWriter mPW;
	private int mNumIndentSpaces;
	private int mCurrentIndentSize;
	private String mIndentation;

	/**
	 * Creates a <code>StandardJavaSourceWriter</code> that outputs to the target
	 * Writer.
	 *
	 * @param iOut the target <code>Writer</code>
	 * @see java.io.FilterWriter#FilterWriter(Writer)
	 */
	public StandardJavaSourceWriter(Writer iOut) {
		this(iOut, kDefaultNumIndentSpaces);
	}

	/**
	 * Creates a <code>StandardJavaSourceWriter</code> that outputs to the target
	 * Writer.
	 *
	 * @param iOut the target <code>Writer</code>
	 * @param iNumIndentSpaces the number of spaces to use per indentation
	 * @see java.io.FilterWriter#FilterWriter(Writer)
	 */
	public StandardJavaSourceWriter(Writer iOut, int iNumIndentSpaces) {
		super(new PrintWriter(iOut));

		mPW = (PrintWriter) out;
		mNumIndentSpaces = iNumIndentSpaces;
		mCurrentIndentSize = 0;
		rebuildIndentation();
	}

	/**
	 * Writes a Java source code object to the underlying Writer.
	 *
	 * @param iSource the source code object to output
	 * @throws IOException any exception thrown by the underlying Writer
	 */
	public void writeJavaSource(JavaSource iSource) throws IOException {
		doWriteJavaSource(iSource);
	}

	private void doWriteJavaSource(JavaSource iSource) {
		// write the source file comment
		doWriteComment(iSource);
		mPW.println();
```

```
	// write the package statement
	String p = iSource.getPackage();
	if (p != null) {
		printLine("package " + p + ";");
		mPW.println();
	}

	// write the import statements
	for (Iterator i = iSource.getImports().iterator(); i.hasNext(); ) {
		printLine("import " + i.next() + ";");
	}
	mPW.println();

	// write the classes
	for (Iterator i = iSource.getClasses().iterator(); i.hasNext(); ) {
		doWriteClass((Class) i.next());
		mPW.println();
	}
}

private void doWriteClass(Class iC) {
	// write the class comment
	doWriteComment(iC);

	// indent as needed
	mPW.print(mIndentation);

	// write modifiers
	doWriteModifiers(iC);

	mPW.print("class " + iC.getName());

	String extendedClass = iC.getExtendedClass();
	if (extendedClass != null) {
		mPW.print(" extends " + extendedClass);
	}

	Iterator interfaces = iC.getImplementedInterfaces().iterator();

	if (interfaces.hasNext()) {
		mPW.print(" implements " + interfaces.next());
		while (interfaces.hasNext()) {
			mPW.print(", " + interfaces.next());
		}
	}

	mPW.println(" {");
	indent();

	// write all fields
	for (Iterator i = iC.getFields().iterator(); i.hasNext(); ) {
		print(kEmptyString);
		doWriteField((Field) i.next());
		mPW.println(';');
	}

	// write all methods
	if (!iC.getMethods().isEmpty()) {
		mPW.println();
	}
	for (Iterator i = iC.getMethods().iterator(); i.hasNext(); ) {

		doWriteMethod((Method) i.next());
```

```
                mPW.println();
        }

        // write all inner classes
        if (!iC.getClasses().isEmpty()) {
                mPW.println();
        }
        for (Iterator i = iC.getClasses().iterator(); i.hasNext(); ) {

                doWriteClass((Class) i.next());
        }
        outdent();
        printLine("}");
}

private void doWriteField(Field iF) {
        //     write the field comment
        doWriteComment(iF);

        // write modifiers
        doWriteModifiers(iF);

        mPW.print(iF.getType());
        mPW.print(' ');
        mPW.print(iF.getName());
}

private void doWriteMethod(Method iM) {
        // write the method comment
        doWriteComment(iM);

        // indent as needed
        mPW.print(mIndentation);

        // write modifiers
        doWriteModifiers(iM);

        // write the return type
        String returnType = iM.getReturnType();
        if (returnType != null && !returnType.trim().equals(kEmptyString)) {
                mPW.print(returnType);
                mPW.print(' ');
        }

        mPW.print(iM.getName());
        mPW.print('(');

        // write method arguments
        Iterator args = iM.getArguments().iterator();
        if (args.hasNext()) {
                doWriteField((Field) args.next());

                while (args.hasNext()) {
                        mPW.print(", ");
                        doWriteField((Field) args.next());
                }
        }

        mPW.print(')');

        // write throws clause
        Iterator exceptions = iM.getExceptions().iterator();
        if (exceptions.hasNext()) {
```

```
                        mPW.print(" throws " + exceptions.next());

                        while (exceptions.hasNext()) {
                                mPW.print(", " + exceptions.next());
                        }
                }

                mPW.println(" {");
                indent();

                // write method body
                for (Iterator i = iM.getBody().iterator(); i.hasNext(); ) {
                        Method.BodyLine line = (Method.BodyLine) i.next();

                        // indent as needed
                        mPW.print(mIndentation);
                        mPW.print(buildIndentation(line.getIndentLevel() *
mNumIndentSpaces));

                        // write the source line
                        mPW.println(line.getText());
                }

                outdent();
                printLine("}");
        }

        private void doWriteComment(Commentable iC) {
                Comment c = iC.getComment();

                if (c == null) {
                        return;
                }

                String[] commentLines = c.getLines();
                for (int i = 0; i < commentLines.length; i++) {
                        printLine(commentLines[i]);
                }
        }

        private void doWriteModifiers(HasModifiers iH) {

                // write the access modifier
                doWriteAccessModifier(iH.getAccessModifier());

                // write all of the other modifiers
                for (Iterator i = iH.getModifiers().iterator(); i.hasNext(); ) {
                        mPW.print(i.next());
                        mPW.print(" ");
                }
        }

        private void doWriteAccessModifier(AccessModifier iAccess) {
                if (iAccess == null || iAccess == AccessModifier.kDefault) {
                        return;
                }

                mPW.print(iAccess.toString());
                mPW.print(' ');
        }

        private void indent() {
                mCurrentIndentSize += mNumIndentSpaces;
```

```
		rebuildIndentation();
	}

	private void outdent() {
		mCurrentIndentSize -= mNumIndentSpaces;
		if (mCurrentIndentSize < 0) {
			mCurrentIndentSize = 0;
		}
		rebuildIndentation();
	}

	private void rebuildIndentation() {
		if (mCurrentIndentSize == 0) {
			mIndentation = kEmptyString;
			return;
		}

		mIndentation = buildIndentation(mCurrentIndentSize);
	}

	private static String buildIndentation(int iSize) {
		StringBuffer sb = new StringBuffer(iSize);
		for (int i = 0; i < iSize; i++) {
			sb.append(' ');
		}

		return sb.toString();
	}

	private void printLine(String iLine) {
		mPW.print(mIndentation);
		mPW.println(iLine);
	}

	private void print(String iLine) {
		mPW.print(mIndentation);
		mPW.print(iLine);
	}
}
```

```
package emuse;

import java.util.*;

/**
 * Various utilities related to string manipulation.
 *
 * @author Mark Ayzenshtat
 */
public class StringUtils {
	private StringUtils() {
		// enforce non-instantiability
	}

	/**
	 * Converts the given word to "Pascal case", in which the first letter of
	 * the word is capitalized and the rest are made lower-case.
	 *
	 * @param iWord the word to convert
	 * @return the converted word
	 */
	public static String toPascalCase(String iWord) {
		int len = iWord.length();
		if (len == 0) {
			return iWord;
		}

		StringBuffer sb = new StringBuffer(len);

		sb.append(Character.toUpperCase(iWord.charAt(0)));

		if (len > 1) {
			sb.append(iWord.substring(1).toLowerCase());
		}

		return sb.toString();
	}

	/**
	 * Tokenizes an action string, using commas as delimiters and taking double-
	 * quoted strings into account.
	 */
	public static String[] tokenizeActionString(String iStr) {
		String[] tokens = new String[6];
		StringTokenizer st = new StringTokenizer(iStr, ",");

		try {
			int firstQuoteIndex = iStr.indexOf('\"');
			if (firstQuoteIndex == -1) {
				// iStr does not contain double-quote characters -- handle
normally

				for (int i = 0; i < 6; i++) {
					tokens[i] = st.nextToken().trim();
				}
			} else {
				// iStr contains double-quoted characters

				int lastQuoteIndex = iStr.lastIndexOf('\"');


				if (firstQuoteIndex == lastQuoteIndex) {
					// should never happen if grammar is
```

```
                                    // written correctly, but just in case
                                    throw new RuntimeException("Only one double-quote
char.");
                              }

                              for (int i = 0; i < 2; i++) {
                                    tokens[i] = st.nextToken().trim();
                              }

                              tokens[2] = iStr.substring(firstQuoteIndex, lastQuoteIndex
+ 1).trim();

                              st = new StringTokenizer(iStr.substring(lastQuoteIndex +
2), ",");

                              for (int i = 3; i < 6; i++) {
                                    tokens[i] = st.nextToken().trim();
                              }
                        }
                  } catch (NoSuchElementException ex) {
                        // should never happen if grammar is written correctly, but just
in case
                        throw new RuntimeException("Error: Not enough tokens in action
string.");
                  }

                  return tokens;
            }

            public static String[] tokenizeCharTypeBody(String iStr) {
                  String[] tokens = new String[3];
                  StringTokenizer st = new StringTokenizer(iStr, ",");

                  try {
                        for (int i = 0; i < 3; i++) {
                              tokens[i] = st.nextToken();
                        }
                  } catch (NoSuchElementException ex) {
                        // should never happen if grammar is written correctly, but just
in case
                        throw new
                              RuntimeException("Error: Not enough tokens in CharType body
string.");
                  }

                  return tokens;
            }

            public static String screenplayOutputClassName(String iFileName) {
                  int dotIndex = iFileName.indexOf('.');
                  if (dotIndex == -1) {
                        return toPascalCase(iFileName) + "ScreenplayOutput";
                  }

                  String safeBase = iFileName.substring(0, dotIndex);
                  return safeBase + "ScreenplayOutput";
            }
}/*
 * TakesAction.java created on Apr 21, 2003
 */
```

```
package emuse.runtime.actions;

import emuse.runtime.model.Adverb;
import emuse.runtime.model.Preposition;
import emuse.runtime.model.StageObject;

/**
 * @author vs299
 */
public class TakesAction extends AbstractAction {

	/* (non-Javadoc)
	 * @see emuse.runtime.model.Action#initialize(emuse.runtime.model.StageObject,
java.lang.Object, emuse.runtime.model.Preposition, emuse.runtime.model.StageObject,
emuse.runtime.model.Adverb)
	 */
	public void initialize(
		StageObject iSubject,
		Object iDirectObject,
		Preposition iPrep,
		StageObject iIndirectObject,
		Adverb iAdverb) {
		// TODO Auto-generated method stub
		super.initialize(
			iSubject,
			iDirectObject,
			iPrep,
			iIndirectObject,
			iAdverb);
	}

	/* (non-Javadoc)
	 * @see emuse.runtime.model.Action#isDone()
	 */
	public boolean isDone() {
		// TODO Auto-generated method stub
		return super.isDone();
	}

	/* (non-Javadoc)
	 * @see emuse.runtime.actions.AbstractAction#onFirstStep()
	 */
	protected void onFirstStep() {
		// TODO Auto-generated method stub
		super.onFirstStep();
	}

	/* (non-Javadoc)
	 * @see emuse.runtime.actions.AbstractAction#onStep()
	 */
	protected void onStep() {
		// TODO Auto-generated method stub
		super.onStep();
	}

}
/*
 * TestDriver.java created on Apr 21, 2003
 */
```

```
package emuse.runtime.anim;
import emuse.ResourceManager;
import emuse.runtime.actions.AppearsAction;
import emuse.runtime.actions.SpeaksAction;
import emuse.runtime.actions.WalksAction;
import emuse.runtime.model.Action;
import emuse.runtime.model.Character;
import emuse.runtime.model.Scene;
import emuse.runtime.model.Screenplay;
import emuse.runtime.model.Setting;
import emuse.runtime.model.Stage;
import emuse.runtime.model.StageObject;



/**
 * This class should eventually be integrated into the
 * generated screenplay code.
 *
 * @author vs299
 */
public class TestDriver {

	public static void test(Screenplay play) throws Exception {

		AnimationFrame animator = AnimationFrame.forScreenplay(play);

		//It's OK to start the animation here because
		//startAnimation can be invoked by any thread.
		//animator.startAnimation();
	}

	public static void test() throws Exception {

		//Initialize the scene
		StageObject cash = new ImageObject("money.jpg");
		Character bush = new Character(0, 0, "bush.gif");
		Character edwards = new Character(200, 200, "sedwards.gif");

		Setting set = new Setting();
		set.setImage(ResourceManager.getInstance().getImage("Sunset.jpg"));

		Scene s = new Scene();
		s.setSetting(set);

		cash.xLoc.setValue(400);
		cash.yLoc.setValue(300);
		cash.scale.setValue(2);

		Action a0 = new AppearsAction(bush, Stage.LEFT);
		Action a01 = new AppearsAction(edwards, Stage.RIGHT);
		Action a02 = new AppearsAction(cash, Stage.CENTER);
		Action a1 = new WalksAction(bush, cash);
		Action a2 = new SpeaksAction(bush, "It is imperative that we used the
correct access modifiers. Show me the money! Friends! Romans! Countrymen! Hello
nurse!");
		Action a3 = new SpeaksAction(edwards,
			"Well, good night. If you do meet Horatio and Marcellus, The
rivals of my watch, bid them make haste.");

		a0.setRunOrder(0);
		a01.setRunOrder(0);
		a02.setRunOrder(0);
```

```
            a1.setRunOrder(1);
            a2.setRunOrder(2);
            a3.setRunOrder(3);

            s.addAction(a0);
            s.addAction(a01);
            s.addAction(a02);
            s.addAction(a1);
            s.addAction(a2);
            s.addAction(a3);
            Screenplay sp = new Screenplay();
            sp.addScene(s);

            SceneManager sceneManager = new SceneManager(sp);

            sceneManager.addSprite(cash);
            sceneManager.setSpeedScaleFactor(1);

            AnimationFrame animator = new AnimationFrame(sceneManager, "Default");

            //It's OK to start the animation here because
            //startAnimation can be invoked by any thread.
            animator.startAnimation();
      }

      public static void main(String args[]) throws Exception {
            test();
      }
}
```

```
package emuse.compiler;

import java.util.*;
import emuse.*;
import emuse.runtime.actions.*;
import emuse.runtime.model.*;
import emuse.runtime.model.Character;

/**
 * A generated screenplay class should eventually look like this.  This class
 * is intentially package-private and should not be exposed as part of the
 * public API.
 *
 * @author Mark Ayzenshtat
 */
class TestScreenplay {
	private Screenplay mScreenplay;
	private CharType ct1;
	private PropType pt1;
	private Setting set1;
	private Character c1;
	private Character c2;
	private Prop p1;
	private Prop p2;
	private Group g1;

	public TestScreenplay() {
		try {
			setTitleAndAuthors();

			defineCharTypes();
			definePropTypes();
			defineSettings();
			defineCharacters();
			defineProps();
			defineGroups();

			defineScenes();
		} catch (EmuseException ex) {
			ex.printStackTrace();
		}
	}

	private void setTitleAndAuthors() throws EmuseException {
		mScreenplay.setTitle("Hamlet for Dummies");
		mScreenplay.setAuthors(new String[] {"Shakespeare, William"});
	}

	private void defineCharTypes() throws EmuseException {
		ResourceManager rm = ResourceManager.getInstance();

		ct1 = new CharType();
		ct1.setHeadImage(rm.getImage("guardhead.jpg"));
		ct1.setArmImage(rm.getImage("guardarm.jpg"));
		ct1.setLegImage(rm.getImage("guardleg.jpg"));
		ct1.setTorsoImage(rm.getImage("guardtorso.jpg"));
		ct1.setGender(Gender.valueOf("male"));
		ct1.setHeight(Height.valueOf("tall"));
		ct1.setBuild(Build.valueOf("medium"));

		mScreenplay.setCharacterTypes(new CharType[] { ct1 });
	}
```

```
	private void definePropTypes() throws EmuseException {
		ResourceManager rm = ResourceManager.getInstance();

		pt1 = new PropType();
		pt1.setAppearance(rm.getImage("sword.jpg"));
		pt1.setScale(Scale.valueOf("medium"));

		mScreenplay.setPropTypes(new PropType[] { pt1 });
	}

	private void defineSettings() throws EmuseException {
		ResourceManager rm = ResourceManager.getInstance();

		set1 = new Setting();
		set1.setImage(rm.getImage("platform.jpg"));
		set1.setSound(rm.getSound("spookynight.mp3"));

		mScreenplay.setSettings(new Setting[] { set1 });
	}

	private void defineCharacters() throws EmuseException {
		c1 = new Character();
		c1.setName("Francisco");
		c1.setCharType(ct1);

		c2 = new Character();
		c2.setName("Bernardo");
		c2.setCharType(ct1);

		mScreenplay.setCharacters(new Character[] { c1, c2 });
	}

	private void defineProps() throws EmuseException {
		p1 = new Prop();
		p1.setName("franSword");
		p1.setPropType(pt1);

		p2 = new Prop();
		p2.setName("bernSword");
		p2.setPropType(pt1);

		mScreenplay.setProps(new Prop[] { p1, p2 });
	}

	private void defineGroups() throws EmuseException {
		g1 = new Group();
		g1.setName("guards");
		g1.setMembers(new StageObject[] { c1, c2 });

		mScreenplay.setGroups(new Group[] { g1 });
	}

	private void defineScenes() throws EmuseException {
		mScreenplay.addScene(createScene1());
	}

	private Scene createScene1() throws EmuseException {
		Scene s1 = new Scene();
		s1.setSetting(set1);

		Action a0 = new JumpsAction();
		a0.initialize(c2, p1, Preposition.valueOf("with"), p2,
Adverb.valueOf("quickly"));
```

```
                s1.addAction(a0);

                return s1;
        }
}
```

```
/*
 * Created on Apr 10, 2003
 */
package emuse.runtime.actions;

import emuse.runtime.model.Adverb;
import emuse.runtime.model.Character;
import emuse.runtime.model.Preposition;
import emuse.runtime.model.StageObject;

/**
 * @author Vladislav
 * @author "Markislav"
 */
public class WalksAction extends AbstractAction {
	// in pixels per second
	public static final double kSlowSpeed = 50.0;
	public static final double kMediumSpeed = 80.0;
	public static final double kFastSpeed = 150.0;

	public static final double kLegAngle = Math.toRadians(20);
	public static final double kLegOffset = Math.toRadians(0);

	public static final double kKneeAngle = Math.toRadians(30);
	public static final double kKneeOffset = Math.toRadians(30);

	// instance variables
	double footAnimationSpeed;
	double speed, dx, dy, dt;
	Character ch = null;
	StageObject target;

	public WalksAction() {}

	public WalksAction(Character c, StageObject target) {
		initialize(c, null, Preposition.kTo, target, null);
	}

	/**
	 * @param iSubject                    the Character that walks
	 * @param iDirectObject          this argument is ignored
	 * @param iPrep                       should be "to", "toward", etc.
	 * @param iIndirectObject        walking to who or what?
	 * @param iAdverb the adverb     slowly or quickly or null
	 */
	public void initialize(
		StageObject iSubject,
		Object iDirectObject,
		Preposition iPrep,
		StageObject iIndirectObject,
		Adverb iAdverb) {

		// set subject
		ch = (Character) iSubject;
		target = iIndirectObject;

		// determine speed
		if (iAdverb == Adverb.kQuickly) {
			this.speed = kFastSpeed;
		} else if (iAdverb == Adverb.kSlowly) {
			this.speed = kSlowSpeed;
		} else {
			this.speed = kMediumSpeed;
```

```
	}

	// the following line is based on empirical observation
	// of test code, not on any scientific formulation
	this.footAnimationSpeed = this.speed / 10;
}

protected void onFirstStep() {
	// determine direction
	double dir = 0;
	double myX = ch.xLoc.getValue();
	double targetX = target.xLoc.getValue();

	if ((targetX - myX) > 0) dir = 1;
	else dir = -1;

	targetX -= target.getWidth() * dir;

	this.dx = targetX - myX;
	this.dt = Math.abs(dx / speed);

	// setup knobs
	ch.lKnee.setOffset(kKneeOffset * dir);
	ch.lKnee.setScale(kKneeAngle * dir);
	ch.lLeg.setOffset(kLegOffset * dir);
	ch.lLeg.setScale(kLegAngle * dir);
	ch.rKnee.setOffset(kKneeOffset * dir);
	ch.rKnee.setScale(kKneeAngle * dir);
	ch.rLeg.setOffset(kLegOffset * dir);
	ch.rLeg.setScale(kLegAngle * dir);

	ch.xLoc.vary(myX, targetX);

	ch.rShoulder.setValue(1);
	ch.rElbow.setValue(1);
	//rShoulder.setValue(1);
	//rElbow.setValue(1);
}

public void onStep() {
	double w = elapsed * footAnimationSpeed;

	if (ch == null) {
		throw new IllegalStateException("WalkAction not initialized");
	}

	if (isDone()) {
		ch.xLoc.setValue(1);
		ch.lKnee.reset();
		ch.lLeg.reset();
		ch.rKnee.reset();
		ch.rLeg.reset();
		return;
	}

	ch.lKnee.setValue(Math.sin(w));
	ch.lLeg.setValue(Math.cos(w));

	ch.rKnee.setValue(Math.sin(w + Math.toRadians(180)));
	ch.rLeg.setValue(Math.cos(w + Math.toRadians(180)));

	ch.xLoc.setValue(elapsed/dt);
	//ch.updateObject();
```

```
	}

	public boolean isDone() {
		return elapsed > dt;
	}
}
```