GRIMM: the choose your own story language
Mike Lenner, Bill Liu, Mariya Nomanbhoy and Becky Plummer
CS4115: Programming Languages and Translators
12/20/04
1. Introduction

The programming language GRIMM is designed to allow the user to easily create a “choose your own adventure” story for children. Although simple, the language is flexible and powerful enough to create complex, text-based adventure games. GRIMM is named in honor of the brothers Grimm, a pair of famous authors of children’s stories.

Using GRIMM, the programmer is able to quickly and naturally create a story with characters, items, and choices for the player using the programmer’s own images. GRIMM allows the programmer to create a personalized story with an ending chosen by the child. The experience is interactive, making it much more engaging than reading a bedtime story. The child will have choices, and therefore will direct the progression of the story until they reach their own ending. The language is designed so that less technical users, such as parents and teachers, are able to quickly create programs and detailed stories even if they lack extensive programming experience.

1.1 Motivations

GRIMM came out of our interest in creating a children’s storytelling language, text-based game creation language, and FSM modeling language. We wanted to create a language that interested us and also allowed the programmers to create something unique, provide entertainment or educate the end user of the application. Combining these interests, it is easy to see how we arrived at a “choose your own adventure” storybook language.

After researching the programming languages created by groups from previous years, we decided we wanted to create a language that allows the programmer to do something that is not already implemented in an existing language. Our language shares some features with the eMuse language. eMuse was designed to allow the programmer to create a screenplay and visualize it before getting actors involved. However, GRIMM allows the user to directly interact with the story in an open-ended, non-deterministic environment.

Lastly, we wanted a language that would allow the user of a created application to be entertained or educated in some way. Initially, we considered implementing a tool that would help students in school by using our language to visualize a problem. However, the creation of a story would be entertaining for a child and inspire them to create their own stories. Therefore, this meets our two-fold objective: to entertain the child and expose them to programming while advancing their education by facilitating them to write their own stories.

1.2 Goals

Inherent in the programming of all text based, role playing games are a number of common concepts. These include creating characters, items, and a space in which these characters and items exist. Using a non-domain specific language, such tasks are non-trivial. Our goal was to create a language with these constructs built in, allowing the programmer to concentrate on the creative aspects of building their adventure.
By taking advantage of the common software aspects of these items and locations, GRIMM requires the developer to merely specify a few details for each new entity they wish to introduce in their story. So, instead of wasting time writing code to implement a front door, the developer can create such an object quickly, then move on to determine what happens when the character passes through.

GRIMM takes the developers focus away from the programming details, and puts it back to where it should lie, in the story. By doing so, the programmer can create a much richer and more intricate plot, making the adventure much more appealing. Additionally, GRIMM gives those unfamiliar with the aspects of programming an easy interface on which to create the stories they may dream up.

1.3 Features

GRIMM: A simple, intuitive, flexible, object oriented, interactive, visual, fun language.

**Simple**

In order for teachers and parents to use the language it must be simple for them to learn and use. These types of people are busy enough without having to undertake learning a programming language, especially if they are not already involved in application development. Therefore we wanted to create a language that would be easy for non-technical people to understand and use. We are offering these people a language that does the image processing and placement, scene creation and movement, and choice implementation and execution without requiring the writer to know about the intricacies of programming. The keywords are straightforward and easy to recall and use. The language is also small and basic. The programmer can make the stories as simple or complex as they feel comfortable with by choosing how many scenes to have, how many objects to use, and how many options the user will have.

**Intuitive**

For non-technical people to use GRIMM it must loosely resemble English to make story-board creation intuitive. Languages such as C++ and Java are hard for non-technical people to understand because the syntax is unfamiliar. With GRIMM, users will understand the keywords and control structures because they will be described using English-based conventions. New story-board creators should be able to easily program-by-example by examining other stories. This will enable non-technical parents and teachers to quickly learn and start writing stories in GRIMM.

**Flexible**

There are many aspects to a story including: scenes, items, characters, descriptions, and open-ended choices. We allow the writer to completely create their own world within our framework. The writer can assign attributes to scenes, such as interesting characters and items. Additionally they can associate images with scenes that will be shown at runtime. This
allows the writer to create stories involving their own surroundings as well as make-believe items using basic predefined data types such as character, scene, and item. This flexibility allows there to be an endless number of possible stories with GRIMM.

**Object Oriented**

GRIMM has many objects, each with their own specific attributes. There are various basic data types that are used to describe a story. One example is a “scene”. Each scene must be created and then assigned values to its attributes such as name and description.

We thought initially that non-technical people could be confused by the idea of Object Oriented, but we feel that grouping all aspects of a data type will instead be helpful to them due to its clear organization. For example, all of the attributes of the scene “hallway” are directly attached to the scene instance:

```plaintext
scene hallway
hallway name "Hallway"
hallway description "You're in the hallway now."
hallway picture "hallway.jpg"
hallway exit bathroom is hidden
hallway exit kitchen
hallway contains item key
```

Object Orientation will help the writer organize objects and their attributes and therefore facilitate the creation of a story.

**Interactive**

Once a GRIMM program is created and presented to a child the possibilities are endless. The child has several choices in every scene to perform actions such as going to another scene, manipulating items, and interacting with characters. For example, if there is a glass on the table, the character is allowed to pick it up. The scene choices are entered through the exits of a current scene that the programmer defines for the user. In some cases an exit may be hidden and the only way the user can find it is through talking to a character. In other cases the user may need an item to get through an exit. This allows the programmer to define many options for the user to interact with in order to move through the story. The open-ended world is a robust feature that makes GRIMM so enjoyable for children. The child’s interest will be captured while they are working within the story. Aside from the child being able to make choices, the child gets to decide how the story progresses and ends based on the choices that they make.

**Visual**

A special feature of GRIMM is the ability for the programmer to input any images of their creation into the story. Each scene in the story has the ability to display a visual element. This allows the story to become more personal for the child who is exploring through the world. As the user moves throughout various scenes, images will be displayed as a
background. We feel that the visual capabilities of GRIMM will help the writer and user to enjoy the experience.

**Fun**

GRIMM allows the writer to create any story with an unlimited number of possibilities. We want people to have fun with the flexibility, interactivity, and visual aspects in order to engage a child in an interesting story. Both aspects of the program should be enjoyable: development and usage. We want the writer to be engaged in the development by making the language intuitive and simple. Its flexibility allows the writer to be creative. The child will become engaged because of the engaging nature of the stories and perhaps partake in the development of new ones.

1.4 Summary

The GRIMM language will open up programming to the programming novice, taking away the intimidating syntax and replacing it with intuitive, domain specific keywords. We have also provided a flexible tool, such that games and stories can be created as simple or as complex as the developer desires. GRIMM also brings a new education aspect to the classroom or the home, one in which skills translate directly into an adventure both teacher and student can enjoy. And finally, GRIMM stops the wasted time spent by previous developers in coding the same basic concepts for every game or story they create.

However, in spite of all these advantages, the most important aspect of the GRIMM language is that of allowing the programmer to focus on what is important. GRIMM redistributes the programmer’s effort to where it belongs - on the story, not on the code.
2. Language Tutorial

2.1 Writing a GRIMM program
There are two main sections of a GRIMM program: the declaration & assignment section and the statement section. You must have at least one statement in the file for it to be a valid GRIMM program. We give you three examples increasing in difficulty.

2.1.1 Hello World
This simple program shows how to use comments and a single statement in the GRIMM language.

Create a file named “hello.gmm” and type in the following example:

```plaintext
:note: My first GRIMM program
say "Hello World"
```

Then follow the compilation instructions below.

Here is the Game Frame output you should see:

![Game Frame output](image)

The program simply outputs “hello world” on the terminal and then ends.
2.1.2 A Simple Room Example
This example shows the simplest use of a room. **NOTE**: each room must have a name

Create a file named “simple.gmm” and type in the following example:

```
:note: a simple GRIMM file with one room
:note: declaration section
scene room
room name "My Room"

:note: action loop section
goto room
read user input
while not user says "quit"
    say "There is nothing in my room. Type 'quit' to end"
    read user input
endwhile
```

Then follow the compilation instructions below.

The Game Frame shows:
Then you must type something at the prompt to get a response. I typed “what is in my room?”

There is nothing in my room. Type ‘quit’ to end.
2.1.3 Successful Program Images

You are standing on a dirt path. Ahead of you in the distance you see the Castle of the North Kingdom. You think you hear something that sounds like trumpets blaring way off in the distance. To your left is a small dagger.

There is an exit to North Castle Gate

pick up dagger

You picked up the dagger
2.2 Compiling the program

Your GRIMM files must be saved in the grimm folder with the compiler. Open a terminal and move into your grimm folder.

Then at the terminal prompt type:

```bash
$ make <example_name>
```

The name of your example is the filename with no extension. For example if your file is called:

```bash
simple.gmm
```

Then type:

```bash
$ make simple
```

at the prompt.
If the program compiles correctly you will see:

```java
java antlr.Tool grimm.g
ANTLR Parser Generator   Version 2.7.4   1989-2004 jGuru.com
java antlr.Tool GRIMMTranslator.g
ANTLR Parser Generator   Version 2.7.4   1989-2004 jGuru.com
javac GRIMM.java
javac Character.java Scene.java GRIMM.java SymbolTable.java GRIMM Lexer.java
Thing.java User.java GRIMM Parser.java GRIMM TokenTypes.java gameFrame.java
GRIMM Walker.java GRIMM Translator.java GRIMM TranslatorTokenTypes.java
outputFrame.java Item.java CustomAST.java
Note: GRIMM Parser.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
java GRIMM hello.gmm
javac hello.java
java hello
```

If you have typed in errors you will see error messages:

```java
java antlr.Tool grimm.g
ANTLR Parser Generator   Version 2.7.4   1989-2004 jGuru.com
java antlr.Tool GRIMMTranslator.g
ANTLR Parser Generator   Version 2.7.4   1989-2004 jGuru.com
javac GRIMM.java
javac Character.java Scene.java GRIMM.java SymbolTable.java GRIMM Lexer.java
Thing.java User.java GRIMM Parser.java GRIMM TokenTypes.java gameFrame.java
GRIMM Walker.java GRIMM Translator.java GRIMM TranslatorTokenTypes.java
outputFrame.java Item.java CustomAST.java
Note: GRIMM Parser.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
java GRIMM hello.gmm
ERROR MESSAGES HERE
```
3. Language Manual

3.1 Lexical Conventions

3.1.1 Tokens
There are three classes of tokens: identifiers, keywords, and string constants. Newlines signify the end of a statement or end of part of a statement. Comments and whitespace are ignored as they only separate tokens.

3.1.2 Comments
These items are ignored, but are used by the programmer to leave notes about what a piece of code does. Therefore, comments are appropriately indicated by the character sequence: “:note:” and end when a new line character is reached, therefore comments may only be one line long.

:note: this is a comment

3.1.3 Identifiers
An identifier can be any alphanumeric combination beginning with an alphabet character. Upper and lowercase letters are considered to be different. An identifier may not have the same spelling as a keyword. All identifiers must be unique.

3.1.4 Keywords
The following words are keywords in the GRIMM language and may not be used as identifiers.

<table>
<thead>
<tr>
<th>and</th>
<th>gameover</th>
<th>item</th>
<th>read</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>goto</td>
<td>name</td>
<td>say</td>
</tr>
<tr>
<td>contains</td>
<td>has</td>
<td>not</td>
<td>says</td>
</tr>
<tr>
<td>description</td>
<td>hidden</td>
<td>or</td>
<td>scene</td>
</tr>
<tr>
<td>drops</td>
<td>holds</td>
<td>otherwise</td>
<td>then</td>
</tr>
<tr>
<td>endif</td>
<td>if</td>
<td>otherwiseif</td>
<td>user</td>
</tr>
<tr>
<td>endwhile</td>
<td>input</td>
<td>pickup</td>
<td>while</td>
</tr>
</tbody>
</table>

User keyword
The user keyword is a special variable that represents the application user. It contains the state of the user throughout the life of the program. For example, the last input given by the user, the item the user holds, and the scene the user is currently in. It has other keywords that are associated with it in order to retrieve or change these states.

3.1.5 Newline
This token signifies the end of a line and is used to terminate program statements. It is defined as a carriage return at the end of a line (hitting the enter key on a keyboard). Even though newlines are ignored in actual program execution they are very important to the structure of the program. A statement may appear only on a line by itself, with the exception of while loops and if statements where newlines identify the different segments of the construct.
3.1.6 Whitespace
Blanks, tabs, newlines, comments, etc are considered white space. Newlines serve the purposes described above, but all others are ignored as they only separate tokens.

3.1.7 String Constants
A series of characters surrounded by double quotes.

3.2 Declarations

3.2.1 Scenes
Scenes represent states the user can occupy usually named after locations such as “hallway” or “school”. A scene is declared using the keyword scene followed by the variable name of the scene. These are all declared at the very beginning of the file, and indicate all different locations available to the story. For example, if a story contained the possibility of movement from the bathroom, bedroom, and backyard the scenes would be declared as follows:

    scene bathroom
    scene bedroom
    scene hallway

3.2.2 Characters
The programmer may choose to put characters in all or some scenes. Characters represent people that the user may interact with to gain information or items in the story. Characters can move between scenes if the programmer wishes. Characters are declared using the keyword character and then the variable name.

    character steven

3.2.3 Items
Items represent objects that can be held or traded by the user. Items do not make sense if they cannot be picked up by a user, for example “house” is not a good choice for an item. At some points the user may need an item to be able to use an exit. The programmer may choose to put items in all or some of their scenes. These items are declared using the keyword item and then the variable name. They are either associated with a scene or a character and may be hidden.

    hallway contains item key
    steven holds item book is hidden

Items contained by a scene cannot have the same name as items held by characters. If a character is in a scene and a character holds the item, this item is not considered to be in the scene. The user must interact with the character to obtain the item.

3.3 Statements
All statements must be followed by a newline character including the last statement in the file.
3.3.1 Assignments
Following the declarations section each scene, character, and item is assigned certain attributes.

Scenes
Each scene is defined by specifying its name, a description, its associated image, its exits, and what items it contains. The name and description will be displayed as text in the runtime environment. Beneath the name, the image will be displayed and then the description. These items give the user information or clues to the layout of a scene. The Programmer must define the name of the scene or they will get a compile error.

The programmer must define exits; they will be displayed at the bottom of the console. Additionally, any items contained in the room will be displayed. As a user may need an item to use an exit, for example they may need a key to open a door, and therefore some exits can be hidden. These will not immediately be displayed at the bottom of the console.

An example scene definition is shown below.

```plaintext
hallway name "Hallway"
hallway description "You're in the hallway now."
hallway picture "hallway.jpg"
hallway exit bathroom is hidden
hallway exit bedroom
hallway contains item knife
```

Characters
The interaction between a character and user is determined by the programmer. However, each character has the capacity to hold items. To define these items the programmer must say that a character holds the item. For example:

```plaintext
steven holds item dragonBook
steven holds item gradedproject is hidden
```

3.3.2 Boolean Expressions
There are 3 conditional statements using the keywords says, inside, and holds.

user says “quote”
The user says expression is used to evaluate whether the last input by the user includes the quoted text.

```plaintext
user says “hello world”
```

For example, if the user says the following expressions it will match.

“hello world Puppy”
“Zippy says hello world lalala”
The following example will not work.
“hello Zippy world”

**user inside scenename**
This expression is used to determine if a user is in a specific scene.

```
user inside kitchen
```

**user has item** or **charactername has item**
This determines if a user or a character holds an item. This can be used to activate hidden exits.

```
steven has key
user has key
```

### 3.3.3 Actions

These statements describe a change in the state of the user or interactions with the user.

**Goto**
A jump statement allows the user to move from one scene to another. This is indicated by using the goto keyword followed by a scene name. If the programmer specifies a scene that is not an exit of the current scene, this will cause a runtime error and the program will exit.

```
goto hallway
```

**Say**
A say statement allows the programmer to speak to convey information to the user by way of the terminal. These messages can be merely instructions or quotes from a character. The action is indicated by the keyword say followed by a quote.

```
say “hello”
```

**Pick up**
The user or characters may pick up items in the scene. This allows users to get items or swap items with a character. This is indicated by the pickup keyword followed by the name of the item to be picked up. The programmer must specify the user or the character name before the pickup keyword. If the programmer specifies an item that is not in the current scene, this causes a runtime error and the program will exit.

```
user pickup key
steven pickup book
```

**Drop**
The programmer may allow for a user or a character to drop an item, therefore no longer possessing it. The user or a character name precedes the drops command to specify whether the user or a character is executing the drop action. If the
programmer specifies an item that the user or character does not currently possess, this causes a runtime error and the program will exit.

   user drops book
   steven drops book

**User input**
The programmer may read user input from the console at any time. As described above the programmer may check for specific input using the user says “quote” statement. These actions are executed by the statement:

   read user input

**Gameover**
The game over action is used to define when the story is over. The programmer may put this at the end of the program or somewhere in a scene. It is indicated by using the keyword gameover.

   gameover

### 3.3.4 Selection
If statements allow the programmer to check the state of the user and then do specified actions. The first line must always be if and must be followed by a boolean expression terminated by the keyword then and a newline character. On successive lines there can be a series of statements. The end of an if-statement must be indicated on a new line using the keyword endif.

   if user says "pickup key" then
     pickup key
   endif

The programmer may compound if-statements with other option by using otherwise. There are two types of compound if-statements: otherwiseif and otherwise.

The otherwiseif are constructed by an otherwiseif, followed by a boolean expression, terminated by a then and a newline character followed by a series of statements. There can be as many otherwiseif’s as needed.

   if user says "pickup key" then
     user pickup key
   otherwiseif user has key and user says "goto bathroom then
     goto bathroom
   endif

Finally there may be a default choice specified by otherwise, a newline character and a series of statements.

   if user says "pickup key" then
     user pickup key
   otherwise
All if-statements, compound or not, must be terminated by the endif keyword.

```
if user says "pickup key" then
    user pickup key
otherwise
    if user has key and user says "goto bathroom" then
        goto bathroom
    otherwise
        gameover
    endif
ENDIF
```

3.3.5 Iteration
While statements allow the programmer to have a set of statements occur as long as a certain state is occurring. While loops evaluate the validity of a boolean expression and then execute a sequence of statements. If the boolean expression is true the statements in the body of the loop will be executed. Otherwise the loop will be terminated. The while statements are indicated by the keyword while followed by a boolean expression followed by a newline character. On succeeding lines there can be any number of statements. The end of a while loop is indicated by the keyword endwhile.

```
while user says "hello"
    read user input
endwhile
```

A while loop with an if-statement in the body:

```
while user inside kitchen
    read user input
    if user says "pickup key" then
        user pickup key
    otherwise
        if user has key and user says "goto bathroom" then
            goto bathroom
        otherwise
            gameover
        endif
    endif
endwhile
```

NOTE: Tabs and blank lines are not necessary, but can help the readability of the program.

3.4 Scope and Namespace
There is one global scope in GRIMM. All variables are declared in the declarations section and then are used in the action loop below. GRIMM has only one namespace and therefore all identifiers must be unique.
3.5 Program Structure
A GRIMM script has two main parts: declarations & assignments and the action loop. A program must contain a statement section (although it does not necessarily have to be a loop) but does not have to contain the declarations or assignments section. The declarations & assignments section is optional. An empty file is not a valid GRIMM script, the simplest GRIMM script may be one statement.

3.5.1 Declarations & Assignments
The programmer must define all scenes, characters, and items before they may be used. Therefore, we force the programmer to do all declarations and assignments at the top of the program before the action loop.

```plaintext
:note: all my scenes
scene hallway
scene bathroom
scene kitchen

:note: attributes of the hallway
hallway name "Hallway"
hallway description "You're in the hallway now."
hallway picture "hallway.jpg"
hallway exit bathroom is hidden
hallway exit kitchen
hallway contains item key

:note: my character
character mariya
mariya holds item dragonbook
```

3.5.2 Action Loop
The action loop is where all actions specified by the programmer are executed. All scenes, character, and items used here must have been declared in the above section. This section can be any sequence of valid statements; we only suggest the action loop structure because of the intuition in the story: if the story is not over, do these actions. For example:

```plaintext
while user inside hallway
    read user input
    if user says "pickup key" then
        user pickup key
    otherwiseif user has key and user says "goto bathroom" then
        goto bathroom
    otherwiseif user says "talk mariya" then
        mariya says "hello, do you want a key?"
    otherwiseif user says "how old are you mariya?" then
        mariya says "24"
    otherwise
        say "I do not understand what you said"
    endif
endwhile
```
4. Project Plan

In order to complete this project in an organized manner we assembled as a group early on in the semester and then set up weekly meetings. After the white paper we each had our own tasks, but we met to bring the group up to speed and to voice any concerns. In addition to our group meetings, we also met at the TA office hours so that we could ask him questions we had. This system worked out well for us until the final weeks of the project when we needed to meet more often to tie things together. Below you will find descriptions of the milestones of the project and organizational details.

4.1 Team Responsibilities

At first we were all unsure about the details of the project and therefore were timid to split up tasks. In the beginning we worked a lot as a group, while doing individual research. We would come up with questions about a piece of the compiler and discuss them. After implementing the first stages of the lexer and parser we started to become more comfortable and eventually each person volunteered to implement part of the project and in some cases two people decided to work together. You may see below the partitioning is a little different than that suggested in class.

<table>
<thead>
<tr>
<th>Mike Lenner</th>
<th>AST Walker/Runtime Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill Liu</td>
<td>AST Walker/Runtime Environment/Testing</td>
</tr>
<tr>
<td>Mariya Nomanbhoy</td>
<td>Translator/Runtime Environment</td>
</tr>
<tr>
<td>Becky Plummer</td>
<td>Lexer/Parser/Documentation</td>
</tr>
</tbody>
</table>

Even though Professor Edwards suggested that each team pick a leader, we thought that we could work as a democracy, however we eventually chose Mariya as our leader. We did end up having a few arguments, but they were settled through much discussion. We made sure that everyone was satisfied or at least in agreement before proceeding. Our group was easy to work in, because no one neglected their responsibilities.

4.2 Project Timeline

The following chart shows the major milestones in the course of our project. You may find a more detailed progression in the “Project Logs” section 4.5. There was no road map to this project other than the due dates of the papers. We found that completing the lexer and parser before the LRM was very helpful.

<table>
<thead>
<tr>
<th>Language Details Defined</th>
<th>Sept 23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whitepaper</td>
<td>Sept 28</td>
</tr>
<tr>
<td>Lexer &amp; Parser Completed</td>
<td>Oct 19</td>
</tr>
<tr>
<td>Language Reference Manual</td>
<td>Oct. 21</td>
</tr>
<tr>
<td>AST Generation Completed</td>
<td>Nov. 11</td>
</tr>
<tr>
<td>Translator Completed</td>
<td>Dec. 9</td>
</tr>
<tr>
<td>AST Walker Completed</td>
<td>Dec 14</td>
</tr>
<tr>
<td>Runtime Library Completed</td>
<td>Dec. 14</td>
</tr>
<tr>
<td>Final Testing Completed</td>
<td>Dec. 19</td>
</tr>
<tr>
<td>Final Report &amp; Presentation</td>
<td>Dec. 20</td>
</tr>
</tbody>
</table>
4.3 Software Development Environments
The compiler and runtime environment code was written in Java, the testing suites were written using python, and the entire project was version controlled with CVS. The lexer and parser portions were created using ANTLR. The AST Walker and translator walked the AST using ANTLR, but specific operations were implemented in nested Java instructions.

4.4 Style Guide
Below you may find the formatting guide for our project. Professor Edwards mentioned the value of using CVS in the project overview lecture. We think that a mention should be made about the use of a style guide. Style guides are very important to the readability and cohesiveness of a large implementation.

4.4.1 Spacing Conventions
All indents are 8 spaces, not 4.

All lines must be less than 80 characters wide.

Curly braces are aligned K&R style. For example:

Function
public int FunctionName(void) {
    //
}

If-statement
if (foo) {
} else {
    //
}

While loop
while (1) {
    //
}

For loop
for (;;) {
    //
}

Try-catch block
try {
    //
} catch {
    //
}

NOTE: Use whitespace appropriately to improve readability.

4.4.2 Commenting Conventions

Important Comments
At the top of each file please add a comment block naming the author(s).
Each class should have a block comment before it describing its purpose.

Each method should have a block comment before it describing its purpose and return values.

Comment Style
A block comment should be formatted:
/*
 * Here is a block comment.
 */

Single line comments should be very short and use /* */. Align them in the same column as other single line comments:
if(true) {
    x = 1;    /* comment aligned */
} else {
    x = 2;    /* comment2 aligned also */
}

Use // only for commenting out code.

4.4.3 Declaration Conventions
Declare all variables at the beginning of blocks. The only exception is the for loop index it may be declared in the for loop structure.
for(int i = 0; ... 

4.4.4 Naming Conventions
Name all identifiers appropriately to the usage or contents.
public scene getUserScene(){
    ...
}

Classes & Functions
Class names are mixed case with the first letter capitalized.
class FooBar {
    int Zoo;
}

Class method names are mixed case with the first letter uncapitalized.
class FooBar {
    public int theStyleGuide(int char) {
        int x = 1;
        ...
    }
}
Variables

Class member variables are mixed case with underscores, the first letter capitalized.

```java
class FooBar {
    int Member_Name;
    ...
}
```

Method variables are lower case with underscores between words.

```java
public int FooBar() {
    int my_variable_name;
    ...
}
```

Constants are upper case with underscores between words.

```java
public int FooBar() {
    const type_t THE_TYPE = "character";
    ...
}
```

Global variables start with a lowercase “g”, words are capitalized with underscores between them.

```java
int gMy_Global_Variable;
```

4.5 Project Logs

4.5.1 Makefile

Working file: Makefile
head: 1.10
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 10; selected revisions: 10
description:

```
revision 1.10
date: 2004/12/20 01:11:15; author: mhn2102; state: Exp; lines: +5 -3
Updated makefile to remove old java file before compiling.
```

```
revision 1.9
date: 2004/12/15 01:09:07; author: wl2139; state: Exp; lines: +3 -0
Added module for test (running all the test scripts)
```

```
revision 1.8
date: 2004/12/15 00:42:31; author: wl2139; state: Exp; lines: +4 -1
Added testclean module to clean up test scripts
```

```
revision 1.7
date: 2004/12/14 22:59:32; author: wl2139; state: Exp; lines: +5 -4
Added rule for CustomAST.java and added a rule for Make clean
```

```
revision 1.6
```
Okay! Makefile doing good now. Now we can just type make example where the folder has a file example.g file in it and then it will build everything and un the program if it compiles. woo woo

----------------------------
revision 1.5
date: 2004/12/14 19:16:54;  author: mhn2102;  state: Exp;  lines: +3 -4
Updated makefile again

----------------------------
revision 1.4
date: 2004/12/14 18:28:12;  author: mhn2102;  state: Exp;  lines: +1 -1
Removed more unnecessay java files... why isnt it producing tokentype files for anyone else?

----------------------------
revision 1.3
date: 2004/12/14 18:24:25;  author: mhn2102;  state: Exp;  lines: +1 -1
Changed Makefile to take out the extra java files being compiled. Dont really understand why I have these files and no one else does...

----------------------------
revision 1.2
date: 2004/12/14 06:10:35;  author: mhn2102;  state: Exp;  lines: +5 -3
I fixed the makefile up and also adding better debugging to GRIMM.java. Everything still compiles and runs as it was before.

----------------------------
revision 1.1
date: 2004/12/09 21:03:26;  author: mhn2102;  state: Exp;
The translator is pretty much completely working. I need to fix the user inside boolean and other than that everything has been minimally tested and is ready for the real test phase. I added a Makefile so now we can just type make and have everything go.

4.5.2 GRIMM.java
Working file: GRIMM.java
head: 1.8
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 8;  selected revisions: 8
description:

----------------------------
revision 1.8
date: 2004/12/18 21:35:06;  author: mml2108;  state: Exp;  lines: +1 -1
changed name of global walker vaibale

----------------------------
revision 1.7
date: 2004/12/15 00:48:48;  author: wl2139;  state: Exp;  lines: +1 -1
Checking command line argument length correctly now

----------------------------
revision 1.6
date: 2004/12/14 18:50:50;  author: mhn2102;  state: Exp;  lines: +0 -3
Updated GRIMM.java so that it doesnt print out some of the debug statements

----------------------------
revision 1.5
date: 2004/12/14 06:10:34;  author: mhn2102;  state: Exp;  lines: +13 -4
I fixed the makefile up and also adding better debugging to GRIMM.java. Everything still compiles and runs as it was before.

--------------------
revision 1.4
date: 2004/12/09 22:57:12; author: mhn2102; state: Exp; lines: +8 -2
Fixed character node problem. Changed show scene so it now calls user as a parameter. Things are looking good...
--------------------
revision 1.3
date: 2004/11/23 23:14:59; author: mml2108; state: Exp; lines: +30 -0
Added error checking and calls to tree walker
--------------------
revision 1.2
date: 2004/10/15 16:55:48; author: rp2176; state: Exp; lines: +20 -8
added program name as a command line arg. Also does checking for missing program name
--------------------
revision 1.1
date: 2004/10/14 23:52:13; author: rp2176; state: Exp;
inital Interpreter

4.5.3 grimm.g
Working file: grimm.g
head: 1.29
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 29; selected revisions: 29
description:
--------------------
revision 1.29
date: 2004/12/20 03:15:29; author: mml2108; state: Exp; lines: +17 -14
enforced < 80 character line limit per style guidelines
--------------------
revision 1.28
date: 2004/12/20 02:57:18; author: rp2176; state: Exp; lines: +1 -1
no more mac \r
--------------------
revision 1.27
date: 2004/12/20 00:21:14; author: rp2176; state: Exp; lines: +15 -18
newlines fixed FOREVER
--------------------
revision 1.26
date: 2004/12/19 22:48:47; author: mml2108; state: Exp; lines: +7 -0
check for adding same exit to a scene twice
--------------------
revision 1.25
date: 2004/12/18 22:48:51; author: mml2108; state: Exp; lines: +9 -7
more walker error message correction
--------------------
revision 1.24
date: 2004/12/18 22:27:28; author: wl2139; state: Exp; lines: +1 -1
Added underscore character to the language
--------------------
revision 1.23
date: 2004/12/18 21:53:08; author: mml2108; state: Exp; lines: +20 -20
fixed bugs and standardized wordings in walker error statements
----------------------------
revision 1.22
date: 2004/12/18 21:34:33; author: mml2108; state: Exp; lines: +157 -107
improved error checking in walker
----------------------------
revision 1.21
date: 2004/12/15 23:47:39; author: wl2139; state: Exp; lines: +2 -1
Fixed typo for error msg
----------------------------
revision 1.20
date: 2004/12/15 23:41:16; author: mml2108; state: Exp; lines: +1 -1
corrected typo
----------------------------
revision 1.19
date: 2004/12/15 23:38:36; author: mml2108; state: Exp; lines: +13 -4
added more error reporting to walker
----------------------------
revision 1.18
date: 2004/12/15 22:15:15; author: mml2108; state: Exp; lines: +10 -4
check for type conflicts on item assignments
----------------------------
revision 1.17
added error messages for used identifier
----------------------------
revision 1.16
date: 2004/12/15 21:09:18; author: mml2108; state: Exp; lines: +1 -1
fixed exit assign error message (for real this time)
----------------------------
revision 1.15
date: 2004/12/15 21:08:17; author: mml2108; state: Exp; lines: +2 -2
fixed exit assign error message
----------------------------
revision 1.14
Improved error output in Walker
----------------------------
revision 1.13
date: 2004/12/14 21:42:34; author: wl2139; state: Exp; lines: +4 -0
Added authors comment
----------------------------
revision 1.12
date: 2004/12/14 21:34:19; author: mml2108; state: Exp; lines: +65 -53
Changed to match new SymbolTable class and fixed newline issue
----------------------------
revision 1.11
date: 2004/12/14 18:40:29; author: mhn2102; state: Exp; lines: +1 -1
Changed grimm.g so that the newline counts the correct number of newlines
----------------------------
revision 1.10
date: 2004/12/09 21:03:26; author: mhn2102; state: Exp; lines: +0 -5
The translator is pretty much completely working. I need to fix the user
inside boolean and other than that everything has been minimally tested and
is ready for the real test phase. I added a Makefile so now we can just type make and have everything go.

----------------------------
revision 1.9
date: 2004/11/23 23:12:36; author: mml2108; state: Exp; lines: +253 -20
Added GRIMMWalker class
----------------------------
revision 1.8
grammar file with beginning of the Tree Walker
----------------------------
revision 1.7
date: 2004/11/03 20:41:52; author: rp2176; state: Exp; lines: +52 -26
partially fixed tree. Still some funny things with decls and if statements.
----------------------------
revision 1.6
date: 2004/10/26 19:33:01; author: rp2176; state: Exp; lines: +36 -28
beginning work on implementing building ASTs
CVs: ----------------------------------------------------------------------
----------------------------
revision 1.5
date: 2004/10/19 22:45:24; author: rp2176; state: Exp; lines: +138 -126
updated grammar with characters, pickup and drop for users and characters.
Characters can begin the story holding items and the scene can begin containing an item
----------------------------
revision 1.4
date: 2004/10/19 19:42:43; author: rp2176; state: Exp; lines: +5 -10
updated grammar file, removed the instance of the character and added more interesting boolean expression definition
----------------------------
revision 1.3
date: 2004/10/15 21:27:05; author: rp2176; state: Exp; lines: +88 -33
grammar including while and if constructs
----------------------------
revision 1.2
date: 2004/10/15 16:57:36; author: mml2108; state: Exp; lines: +1 -1
support multiple actions in exit assignment
----------------------------
revision 1.1
date: 2004/10/14 23:51:52; author: rp2176; state: Exp;
initial grammar file

4.5.4 SymbolTable.java
Working file: SymbolTable.java
head: 1.7
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 7; selected revisions: 7
description:
----------------------------
revision 1.7
date: 2004/12/19 22:48:47; author: mml2108; state: Exp; lines: +48 -15
check for adding same exit to a scene twice

revision 1.6
date: 2004/12/19 22:34:19; author: wl2139; state: Exp; lines: +2 -0
style update

revision 1.5
date: 2004/12/18 20:01:51; author: wl2139; state: Exp; lines: +111 -82
Stylized according to style sheet

revision 1.4
date: 2004/12/15 22:15:45; author: mml2108; state: Exp; lines: +10 -1
add overloaded function isDeclared to except a type argument

revision 1.3
date: 2004/12/14 22:51:17; author: mml2108; state: Exp; lines: +l -1
improved error reporting

revision 1.2
date: 2004/12/14 21:25:07; author: mml2108; state: Exp; lines: +61 -83
Redesigned the symbol table

revision 1.1
date: 2004/11/23 23:06:50; author: mml2108; state: Exp;
Initial version

4.5.5 CustomAST.java
Working file: CustomAST.java
head: 1.3
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 3; selected revisions: 3
description:

revision 1.3
date: 2004/12/19 22:32:03; author: wl2139; state: Exp; lines: +2 -0
style update

revision 1.2
date: 2004/12/19 22:25:08; author: wl2139; state: Exp; lines: +44 -25
Updated according to style guide

revision 1.1
date: 2004/12/14 22:45:22; author: mml2108; state: Exp;
New class such that walker has access to tree node's line numbers

4.5.6 GRIMMTranslator.g
Working file: GRIMMTranslator.g
head: 1.11
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv

total revisions: 11; selected revisions: 11

description:
----------------------------
revision 1.11
date: 2004/12/20 00:45:54; author: mml2108; state: Exp; lines: +2 -2
changed exact string matching to substring matching for user says statements
----------------------------
revision 1.10
date: 2004/12/20 00:35:12; author: mhn2102; state: Exp; lines: +1 -1
Updated translator to fix pickup statement
----------------------------
revision 1.9
date: 2004/12/19 23:53:17; author: mml2108; state: Exp; lines: +4 -2
improved gameover functionality and corrected string matching
----------------------------
revision 1.8
date: 2004/12/19 23:20:32; author: mhn2102; state: Exp; lines: +1 -1
Fixed say problem in translator.
----------------------------
revision 1.7
date: 2004/12/19 21:22:03; author: mhn2102; state: Exp; lines: +414 -490
I updated the trnslator to mangle variable names oand to fit styl guide.
----------------------------
revision 1.6
date: 2004/12/15 22:43:56; author: mml2108; state: Exp; lines: +1 -1
used compareToIgnoreCase instead of equals
----------------------------
revision 1.5
date: 2004/12/15 22:33:45; author: mml2108; state: Exp; lines: +1 -1
used equals function for testing string equality
----------------------------
revision 1.4
date: 2004/12/15 21:47:11; author: mhn2102; state: Exp; lines: +12 -3
Added error checking functions to user and character so that they can only
pickup and drop items that are in the scene or that they have.
----------------------------
revision 1.3
date: 2004/12/09 22:57:12; author: mhn2102; state: Exp; lines: +18 -13
Fixed character node problem. Changed show scene so it now calls user as a
parameter. Things are looking good...
----------------------------
revision 1.2
date: 2004/12/09 21:37:45; author: mhn2102; state: Exp; lines: +3 -2
Updated translator so pickup and inside now work. Next step is to update the
GRIMM file and fix the unexpected node warning
----------------------------
revision 1.1
date: 2004/12/09 21:03:26; author: mhn2102; state: Exp;
The translator is pretty much completely working. I need to fix the user
inside boolean and other than that everything has been minimally tested and
is ready for the real test phase. I added a Makefile so now we canjust type
make and have everything go.
4.5.7 gameFrame.java
Working file: gameFrame.java  
head: 1.12  
branch:  
locks: strict  
access list:  
symbolic names:  
keyword substitution: kv  
total revisions: 12; selected revisions: 12  
description:  
-----------------------------  
revision 1.12  
date: 2004/12/20 00:55:18; author: mml2108; state: Exp; lines: +2 -2  
changed JSplit pane init (seriously)  
-----------------------------  
revision 1.11  
date: 2004/12/20 00:46:59; author: mml2108; state: Exp; lines: +1 -1  
changed JSplit pane init  
-----------------------------  
revision 1.10  
date: 2004/12/19 23:19:23; author: mml2108; state: Exp; lines: +22 -13  
don't show scene if given a null scene  
-----------------------------  
revision 1.9  
date: 2004/12/18 22:00:58; author: mml2108; state: Exp; lines: +486 -378  
updated per style sheet guidelines  
-----------------------------  
revision 1.8  
date: 2004/12/17 00:00:40; author: mml2108; state: Exp; lines: +132 -88  
cleaned up layout, fixed two bugs  
-----------------------------  
revision 1.7  
date: 2004/12/15 22:34:06; author: mml2108; state: Exp; lines: +8 -1  
fixed bug when no image is given  
-----------------------------  
revision 1.6  
Added error checking functions to user and character so that they can only  
pickup and drop items that are in the scene or that they have.  
-----------------------------  
revision 1.5  
date: 2004/12/15 04:42:17; author: mml2108; state: Exp; lines: +54 -15  
cleaned up default case and correct scene and user item issues  
-----------------------------  
revision 1.4  
date: 2004/12/09 22:29:09; author: mml2108; state: Exp; lines: +5 -14  
added second parameter (User u) to showScene method  
-----------------------------  
revision 1.3  
date: 2004/11/11 02:14:56; author: mml2108; state: Exp; lines: +87 -48  
Looks good except for the layout sizes. I don't think this is a big deal  
though  
-----------------------------  
revision 1.2  
date: 2004/11/04 04:35:34; author: mml2108; state: Exp; lines: +81 -29  
added more methods, cleaned up look and feel
4.5.8 Thing.java
Working file: Thing.java
head: 1.4
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 4; selected revisions: 4
description:

-----------------------------
revision 1.4
date: 2004/12/19 22:31:29; author: wl2139; state: Exp; lines: +2 -0
style update
-----------------------------
revision 1.3
date: 2004/12/18 20:02:13; author: wl2139; state: Exp; lines: +17 -3
Stylized according to style sheet
-----------------------------
revision 1.2
date: 2004/11/20 02:49:42; author: wl2139; state: Exp; lines: +5 -1
Fixed constructor inheritance problem
-----------------------------
revision 1.1
date: 2004/11/20 02:33:09; author: wl2139; state: Exp;
New base Thing class for Character, Item, Scene

4.5.9 Scene.java
Working file: Scene.java
head: 1.10
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 10; selected revisions: 10
description:

-----------------------------
revision 1.10
date: 2004/12/19 22:30:45; author: wl2139; state: Exp; lines: +2 -0
style update
-----------------------------
revision 1.9
date: 2004/12/19 22:19:31; author: wl2139; state: Exp; lines: +36 -36
Style guide fixes
-----------------------------
revision 1.8
date: 2004/12/18 20:23:26; author: wl2139; state: Exp; lines: +4 -0
Stylized according to style guide
-----------------------------
revision 1.7
Revision 1.6
Date: 2004/12/15 21:47:11; Author: mhn2102; State: Exp; Lines: +12 -3
Added error checking functions to user and character so that they can only
pickup and drop items that are in the scene or that they have.

Revision 1.5
Date: 2004/12/09 21:03:26; Author: mhn2102; State: Exp; Lines: +5 -0
The translator is pretty much completely working. I need to fix the user
inside boolean and other than that everything has been minimally tested and
is ready for the real test phase. I added a Makefile so now we can just type
make and have everything go.

Revision 1.4
Date: 2004/11/23 23:13:59; Author: mml2108; State: Exp; Lines: +12 -1
Further corrected constructor errors

Revision 1.3
Date: 2004/11/20 02:49:41; Author: wl2139; State: Exp; Lines: +1 -1
Fixed constructor inheritance problem

Revision 1.2
Date: 2004/11/20 02:32:39; Author: wl2139; State: Exp; Lines: +1 -7
Made class extend the base Thing class

Revision 1.1
Date: 2004/10/26 05:44:46; Author: wl2139; State: Exp;
Initial checkin

4.5.10 Item.java
Working file: Item.java
Head: 1.7
Branch:
Locks: strict
Access list:
Symbolic names:
Keyword substitution: kv
Total revisions: 7; selected revisions: 7
Description:
Revision 1.7
Date: 2004/12/19 22:31:09; Author: wl2139; State: Exp; Lines: +2 -0
Style update

Revision 1.6
Date: 2004/12/18 20:23:26; Author: wl2139; State: Exp; Lines: +4 -0
Stylized according to style guide

Revision 1.5
Date: 2004/12/18 20:18:58; Author: wl2139; State: Exp; Lines: +12 -6
Stylized according to style sheet

Revision 1.4
Date: 2004/12/09 21:03:26; Author: mhn2102; State: Exp; Lines: +7 -1
The translator is pretty much completely working. I need to fix the user inside boolean and other than that everything has been minimally tested and is ready for the real test phase. I added a Makefile so now we can just type make and have everything go.

-----------------------------------------------------
revision 1.3
date: 2004/11/20 02:49:41; author: wln2139; state: Exp; lines: +1 -1
Fixed constructor inheritance problem

----------------------------
revision 1.2
Made class extend the base Thing class

----------------------------
revision 1.1
date: 2004/10/26 05:44:46; author: wln2139; state: Exp;
Initial checkin

4.5.11 Character.java
Working file: Character.java
head: 1.5
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 5; selected revisions: 5
description:

----------------------------
revision 1.5
date: 2004/12/19 21:15:24; author: mhn2102; state: Exp; lines: +80 -42
I updated User and Character classes to print errors if they call invalid actions. Also made them fit style guide.

----------------------------
revision 1.4
date: 2004/12/15 21:47:10; author: mhn2102; state: Exp; lines: +20 -3
Added error checking functions to user and character so that they can only pickup and drop items that are in the scene or that they have.

----------------------------
revision 1.3
date: 2004/11/20 02:49:41; author: wln2139; state: Exp; lines: +2 -2
Fixed constructor inheritance problem

----------------------------
revision 1.2
Made class extend the base Thing class

----------------------------
revision 1.1
date: 2004/10/26 05:44:46; author: wln2139; state: Exp;

Here are the User and Character class files. They compile and should run correctly. -mariyaCVr: -----------------------------------------------

4.5.12 User.java
Working file: User.java
head: 1.5
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 5; selected revisions: 5
description:

revision 1.5
date: 2004/12/19 23:53:53; author: mml2108; state: Exp; lines: +4 -4
slightly cleaned up User class runtime error functions

revision 1.4
date: 2004/12/19 22:59:13; author: mhn2102; state: Exp; lines: +1 -1
I fixed the User class so it makes sure the scene is initialized before testing if the exits exist.

revision 1.3
date: 2004/12/19 21:15:24; author: mhn2102; state: Exp; lines: +108 -62
I updated User and Character classes to print errors if they call invalid actions. Also made them fit style guide.

revision 1.2
Added error checking functions to user and character so that they can only pickup and drop items that are in the scene or that they have.

revision 1.1
date: 2004/10/28 18:54:59; author: mhn2102; state: Exp;

Here are the User and Character class files. They compile and should run correctly. -mariyaCVr:

------

4.5.13 grimmtest.py
Working file: grimmtest.py
head: 1.5
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 5; selected revisions: 5
description:

revision 1.5
Reading stdout and stderr to compare test outputs to now because the parser uses stderr for some of its error msgs

revision 1.4
date: 2004/12/15 00:41:11; author: wl2139; state: Exp; lines: +8 -0
Add a check to make sure the GRIMM translator has been built

revision 1.3
date: 2004/12/15 00:02:42; author: wl2139; state: Exp; lines: +15 -10
Added ability to run the script from a path other than the current path. Use os.path library for better directory handling. Now only compares stdout to test output. Before, it was comparing stdout and stderr to the test output.

-----------------------------
revision 1.2
date: 2004/12/14 19:27:07;  author: wl2139;  state: Exp;  lines: +7 -11
Made it more efficient by only needing to specify outputs for tests that fail.

-----------------------------
revision 1.1
date: 2004/12/14 17:59:44;  author: wl2139;  state: Exp;
Initial checkin for test script engine
5. Architectural Design

Considering that GRIMM is a storybook creation language designed for ease of parents and teachers, not only does our project have all the aspects needed in a compiler, it also has a runtime environment. Thus you will find two parts to the architecture of our project. As described above, Becky implemented the lexer, parser, and AST generation; Mike and Billy created the AST Walker; Mariya wrote the translator; and Mike, Billy, and Mariya created the runtime environment.

5.1 GRIMM Compiler

The compiler contains a lexer, parser, AST walker, and translator as shown in the diagram below. In addition to these components we have also identified critical error points. These points define a stop in compilation if errors are found, there are three such points. The first point is after the parser, we decided if errors were found during parsing then compilation should not continue to check the static semantics. Similarly, we decided if errors were found while checking the static semantics then translation should not be launched.

**GRIMM**

The engine to the compiler is a java file that takes in the name of the GRIMM source file and then takes it through every step of its journey into Java code. It passes the filestream object into the lexer, the lexer splits the file up into a stream of tokens removing whitespace and comments, checks for errors if none the tokens generated by the lexer are passed into the parser, checks for errors if none passes the AST generated by the parser into the AST Walker, checks for errors if none it finally passes the AST generated by the AST Walker into the Translator. Each of these components will be discussed in detail within their respective sections.

**Lexer**

The lexer, developed in ANTLR by Becky Plummer, defines a simple set of tokens for the language. These tokens are described in detain in the language reference section above. The lexer takes a Java FileReader stream as input and outputs the GRIMM file as tokens. The lexer reads in the sequences of characters separated by white space and checks their spelling against the conventions we defined. An important job for the lexer is to remove whitespace (not including newline characters) and comments from the source.
**Parser**
From the specific tokens spit out by the lexer, the parser checks the syntax of sets of tokens developed in ANTLR by Becky. The importance of the parser is that it defines the syntac of the language and all other sections of the compiler depend on its accuracy. The parser checks it the tokens appear in the predicted order. For example:

```plaintext
user inside NOUN
```

When parsing this statement the parser would check that the token “user” appears first, then “inside”, and lastly a “NOUN” defined by the lexer as a sequence of characters and numbers starting with a character. The parser does not check the semantics of the statement, only that the tokens appear in the expected order.

**AST Walker**
The GRIMMWalker class verifies the semantic rules within a GRIMM program. The walker was implemented by Mike Lenner and Bill Liu using ANTLR and Java syntax. It works by walking through the Abstract Syntax Tree created by the Parser. The Walker uses the SymbolTable class, which implements most of the semantic checking functions the Walker uses.

The Walker's initial task is to build the symbol table. That is, the walker first goes through the declaration section of the GRIMM program, adding symbols with their correct type to the symbol table. The walker verifies that no two symbols are given the same identifier, because GRIMM has only one namespace. The walker also verifies that all Scene variables are assigned a name property as this is a rule of the GRIMM language. Finally, the walker also checks for type compatibility, as in making sure items are assigned to rooms or assigned to characters with the correct syntax. The Parser only checks that a noun has an item, but there is separate syntax for scene and character containing an item, both scenes and characters are nouns.

Once the walker has passed through the declaration nodes, it goes through the control flow of the program. Passing through these nodes, it verifies all referenced variables have been declared.

The walker implements a custom AST node class that allows access to the line number of each node. The walkers printError function takes advantage of this fact, printing out line numbers with each error message it displays.

**Translator**
The input into the translator is the Abstract Syntax Tree built by the GRIMM Parser and verified by the Walker. The translator was created by Mariya Nomanbhoy using ANTLR and Java instructions. The GRIMM Translator class walks the AST node by node to create the final Java source output, which will be the executable program once run through the java compiler. The output Java source file is named the same name as the input GRIMM file.
Each GRIMM program has two sections, as described above, the declarations & Assignments and the Action Loop. Once all declarations and assignments are complete, the Translator will move to the statements section of the GRIMM code. All control flow nodes map directly to Java statements with the same function. The while, if, and otherwise nodes cause the Translator to generate the correct java blocks, using the endwhile, and endif to end the block appropriately.

The GRIMM specific objects are the User, Scene, Character, and Item, described in the Runtime Environment section below. These were developed as classes by Mike, Bill, and Mariya. Methods within each object provide for conditional testing of an objects’ state as well as for manipulating the data inside each object. For example:

steven drops dragonBook

would be translated into a method call that would move the item “dragonBook” between the character class and the current scene class.

5.2 GRIMM Runtime Environment

5.2.1 Object Descriptions
The Java classes that make up the runtime library are divided into three parts. The first grouping includes the Thing class, along with its three derived classes, Scene, Character, and Item. These classes make up the basic building blocks of the runtime library.

Scene, Character, and Item
The Scene class contains all data associated with a scene (image, description, title) as well as methods to retrieve that data. It has the ability to contain Items objects, and of
course methods to manipulate those items. The Character class has less data associated with it as compared to the Scene class, yet it does also provide the ability to own Item objects. The Item class is very lightweight as its primary use is to determine its ownership. Both the Scene and the Character class may own Items, and the Item object is used so as to allow the Scene or the Character to report the existence of that Item.

**User**
The User class is used to maintain the state of the first person role player during the game. As with character, it has the ability to hold items. However, the User class also contains certain game-play information, such as the current scene, along with methods to change that data. It also maintains records of the game player’s input statements.

**GameFrame**
The final piece of the runtime library is the GameFrame class. This is the class that implements the graphical interface used to display the game play. All text and images are displayed via this class. Additionally, user input is controlled and recorded by this class, passing the data to the User class when appropriate. The user is only allowed to input instructions when the GRIMM program asks for it. At all other times the user is blocked from entering instructions.

**5.2.2 Translator Interaction**
As the translator encounters nodes of type Scene, Item, and Character, it generates code to instantiate an object from the class of that same name. All three of these classes inherit from the base Thing class. Each declaration of these variables in the GRIMM code corresponds directly to an object instantiation. As the translator encounters nodes representing property assignments of each of these GRIMM data types, it adds method calls from these objects. These method calls assign the passed in data to its owner object (e.g.: a scene object’s name or the items it contains).

The control flow nodes also contain direction for the way in which the translator causes the generated code to interact with the user. That is, nodes such as `goto` and `read` cause the translator to invoke methods from the GameFrame class. These calls determine at which point a user may enter input as well as what images and text are displayed to the screen.
6. Test Plan

6.1 Testing Description
GRIMM testing is somewhat difficult, because much of it relies on the user interaction portion of the language. Therefore there are many different kinds of program testing: automated and human interaction. At the heart of the GRIMM testing environment is the testing engine written in Python by Bill Liu. The testing engine, grimmtest.py, takes GRIMM file(s) as input, compiles .gmm files using the GRIMM compiler, and checks for error messages generated. If error messages were encountered, the testing engine compares the generated output file against the ideal output for the input file. Any differences are outputted as a “diff” to the terminal.

6.1.1 Categories of Tests
The GRIMM testing files are broken into four categories to test each phase of the compilation process:

- **Lexer Tests** – testing the errors in tokens
- **Parser Tests** – testing the language syntax
- **Static Semantics Tests** - testing the tree walker and symbol table.
- **Runtime Tests** – these are unautomated tests done by playing the game, checking for logic errors.

6.1.2 Automation
The testing engine has an option to automatically scan the testing directories and run each test file. This makes it simple to run all the tests quickly after a modification has been made to the language and to ensure there were no side effects.

6.1.3 Testing Choices
Each of the test scripts has a “:note:” in the first line of the program describing what is being tested. All of the examples are included in the Test Suites section below. We chose examples based on their importance to the language. For example, there are some simple examples such as spelling of identifiers and more complicated ones such as while loop structure. Each category listed above has various example tests that are pertinent to that area.

6.1.4 Test Format
Each test consist of a GRIMM .gmm file which contains the GRIMM code we are testing. And, it also consists of a .tst file, which contains the expected output of the associated GRIMM code.

6.2 Generated Source Code Samples
We each generated one complicated example with at least two rooms, characters, and items. The object of these examples is to test the logic of the runtime environment and how it reacts to the user’s actions.
6.2.1 Dangerous Aho

This is an example using the grimm language. The purpose of the game is to get from Butler library to Prof. Edwards office. In order to do this they must first get a key from the library. They pick up money or a book to exchange with the librarian. Doing this will give the user clues that they need the key. They can then move to Butler lawn, where the talk to a child or teacher. The child can give them a balloon to help them fly over construction to get into Mudd. Once in Mudd the user has a choice to talk to Prof. Aho and answer the question "What is the best language?" to which the proper response is AWK. However, if they get it right and ask him to explain the language he takes forever and the user loses. Otherwise, they can stay out of Aho's dangerous office and answer what is the best class ever, to which the proper response is obviously PLT. If they answer these questions correctly and have the key they can get to Edwards office. If they do not have the key they are returned to Butler.

:note: declaring scenes
scene butler
scene mudd
scene lawn
scene ahosOffice
scene edwardsOffice

character librarian
character child
character aho

child holds item balloon

butler name "Butler Library"
butler description "You are currently in Butler Library. There is a librarian in the library. To pick up an item in the scene type pickup item_name. To give something to a character type give item_name to character_name. To move to another scene type move to scene_name."
butler picture "butler.jpg"
butler contains item key
butler contains item money
butler contains item book
butler exit lawn

mudd name "Mudd"
mudd description "You are currently in Mudd. To pick up an item in the scene type pick up item_name. To give something to a character type give item_name to character_name. To move to another scene type go to scene_name."
mudd picture "mudd.jpeg"
mudd exit ahosOffice is hidden
mudd exit edwardsOffice is hidden

lawn name "Lawn"
lawn description "You are currently in Butler Lawn. To pick up an item in the scene type pick up item_name. To give something to a character type give item_name to character_name. To move to another scene type go to scene_name."
lawn picture "lawn.jpeg"
lawn exit mudd
lawn exit butler
ahosOffice name "Al Aho's Office"
ahosOffice description "You are currently in Al Aho's Office. To pick up an item in the scene type pick up item_name. To give something to a character type give item_name to character_name. To move to another scene type go to scene_name."
ahosOffice picture "aa.jpg"
ahosOffice exit mudd
ahosOffice exit butler is hidden
ahosOffice exit edwardsOffice is hidden

edwardsOffice name "Stephen Edwards Office"
edwardsOffice description "You are currently in Stephen Edwards Office."
edwardsOffice picture "se.jpg"

go to butler

while not user inside edwardsOffice
say "You need to make your way from Butler to Prof. Edwards office in order to demo the hard work you put into your term project."
say "The librarian has accused you of stealing a book."
say "Choose to pay fine by typing pay."
say "Otherwise choose to argue by typing argue"
while user inside butler
read user input
if user says "pay" then
  if user has money then
    user drops money
    librarian pickup money
    say "They now think you stole another book. Hint: Try picking up book..."
  otherwise
    say "You have to make some money first!"
endif
otherwiseif user says "argue" then
go to lawn
say "You have been ejected from Butler!"
say "At least you didn't pay."
otherwiseif user says "pick up book" and not user has book then
  user pickup book
otherwiseif user says "give book to librarian" and user has book then
  user drops book
  librarian pickup book
  say "Librarian: Thank you! You should pick up the key."
otherwiseif user says "pick up key" and not user has key then
  user pickup key
otherwiseif user says "pick up money" and not user has money then
  user pickup money
otherwiseif user says "go to Lawn" then
go to lawn
otherwise
  say "cannot understand what you are saying."
endif
endwhile

while user inside lawn
say "Would you like to talk to the child or the teacher?"
say "Type child or teacher"
read user input
if user says "child" then
    say "Child: Do you want a balloon or a pony?"
    say "Type balloon or pony."
    read user input
    if user says "balloon" and not user has balloon then
        child drops balloon
        user pickup balloon
        say "Child: Good choice. This will help you fly."
    otherwiseif user says "pony" then
        say "Child: You really think I will give you a pony?"
    otherwise
        say "I don't understand you."
    endif
otherwiseif user says "teacher" then
    say "Teacher: To get to mudd you must fly over construction."
    say "Now you must learn how to fly."
otherwiseif user says "go to Butler Library" then
    goto butler
otherwiseif user says "go to Mudd" then
    if user has balloon then
        goto mudd
    otherwise
        say "You need something to help you fly."
    endif
otherwise
    say "I don't understand what you are saying."
endif
endwhile
while user inside mudd
    say "You got into mudd! Now you need to find Edwards office."
    say "Would you like to ask a professor? Type yes or no."
    read user input
    if user says "yes" then
        goto ahosOffice
    otherwiseif user says "no" then
        say "Okay, then you must answer a question."
        say "What is the best class ever?"
        read user input
        if user says "PLT" then
            if user has key then
                goto edwardsOffice
            otherwise
                say "You got that right."
                say "Too bad you don't have the key!"
                goto butler
            endif
        otherwise
            say "NO NO NO!"
        endif
    otherwise
        say "I don't understand you."
    endif
endwhile
say "Aho: Hi!  To get directions you need to answer a question."
say "Aho: What is the best language ever?"
read user input
if user says "AWK" then
    say "Aho: Correct!  Would you like me to tell you about it?"
say "Type yes or no."
read user input
    if user says "yes" then
        say "You got him started talking!"
say "You missed your appointment!"
gameover
    elseif user says "no" then
        say "Good choice.  You know enough about it already!"
say "Aho: Alright then. Just type 'please let me pass PLT' and you will get to Edward's office."
read user input
    if user says "please let me pass PLT" then
        goto edwardsOffice
    otherwise
        say "Wrong choice."
    endif
    else
        say "I dont understand what you are trying to say."
    endif
elseif user says "go to Mudd" then
    goto mudd
else
    say "Wrong, try again."
endif
endwhile

while user inside edwardsOffice
    say "You've found your way to see Prof. Edwards!  He gives you an A+!  You win!"
gameover
endwhile
endwhile

dangerousAho.java
/*
 * File: GRIMM.java
 *
 * Authors: Mike Lenner, Billy Liu, Mariya Nomanbhoy, Becky Plummer
 */
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

/*
 * This class runs the lexer, parser, walker, and translator on
 * a .gmm file, outputting any compiler errors to the console.
 */
class GRIMM {
    public static void main(String[] args){

if(args.length > 0){
    try {
        /*get input stream to a file*/
        File fin = new File(args[0]);
        FileReader in = new FileReader(fin);

        /*create lexer and parser objects*/
        GRIMMLexer lexer = new GRIMMLexer(in);
        GRIMMParser parser = new GRIMMParser(lexer);
        GRIMMWalker walker = new GRIMMWalker();

        /*run lexer and parser on the input file*/
        parser.program();

        if (lexer.num_errors > 0) {
            System.out.println("Scanner Errors!");
            return;
        }

        if (parser.num_errors > 0) {
            System.out.println("Parser Errors!");
            return;
        }

        /*Get the AST from the parser*/
        CommonAST parseTree =
            (CommonAST)parser.getAST();

        /*Open a window in which the AST is* /
        /*displayed graphically*/
        //ASTFrame frame =
        //new ASTFrame("AST from the Simp parser", 
        //parseTree);

        walker.program(parseTree);

        if (walker.gRet < 0) {
            System.out.println("Walker Errors!");
            return;
        }

        //frame.setVisible(true);

        GRIMMtranslator gt =
            new GRIMMtranslator(args[0]);
        gt.program(parseTree);
    } catch(IOException e) {
        System.out.println("Error occured while reading your broom file.");
        e.printStackTrace();
        System.out.println("exception: "+e);
    } catch/antlr.RecognitionException re) {
        System.out.println("Recognition Exception occured while reading" 
            + " your grimm file.");
    }
}
re.printStackTrace();
System.out.println("exception: "+re);
} catch(antlr.TokenStreamException re) {
    System.out.println("Token Stream Exception" + " occurred while reading your grimm file");
    re.printStackTrace();
    System.out.println("exception: "+re);
}
else{
    System.out.println("ERROR 6111: No program name found" + " at the command prompt");
}
return;
}

6.2.2 Steal the Crown
This example shows a story with multiple scenes where the user must go to another scene and get money that will allow the user to open a secret door in the first room. Then inside the secret room there is a crown that the user must pickup to win the game. The user must talk to a character to get the key to the crown room.

:note: scene declarations
scene castleLobby
scene throneRoom
scene jewelRoom

:note: character declarations
character queen
character maid
maid holds item key

:note: castle lobby
castleLobby name "Entrance Hallway"
castleLobby picture "castle_lobby.jpg"
castleLobby description "The castle has many hidden rooms and talking to strangers may help you discover passage. Can you find the crown? The maid 'Jessica' is cleaning in the lobby. Say hello to her for her to respond"
castleLobby exit throneRoom
castleLobby exit jewelRoom is hidden

:note: throne room
throneRoom name "Throne Room"
throneRoom picture "throne_room.jpg"
throneRoom description "The throne room is where the queen has audiences with members of the public. The queen is expecting you. Say Your Majesty to talk to her."
throneRoom exit castleLobby
throneRoom contains item money

:note: jewel room
jewelRoom name "Hidden Jewel Vault"
jewelRoom picture "crown.jpg"
jewelRoom description "You have found the secret Jewel Room. Look at the crown jewels aren't they beautiful?"
jewelRoom exit castleLobby
jewelRoom contains item crown
go to castleLobby

while not user inside castleLobby or not user has crown

  :note: castleLobby
  if user inside castleLobby then
    read user input
    if user has key and user says "Jewel Room" then
go to jewelRoom
  otherwise if user says "hello Jessica" then
    say "no stranger has ever spoken to me. I have keys to every room!!"
say "Do you have any money? I can point you to palace secrets."
  otherwise if user says "where do I get money?" then
    say "Well you could ask the queen of course, but don't
tell her I sent you!"
  otherwise if user has money and user says "I have money" then
    user drops money
    maid pickup money
    maid drops key
    user pickup key
    say "You now have a key. There is a hidden Room door.
To open it type 'Jewel Room'"
  otherwise if user says "throne room" then
go to throneRoom
  otherwise
    say "I don't understand you. Did you use punctuation?"
  endif
endif

if user inside throneRoom then
read user input
  if user says "Your Majesty" then
    say "Greetings peasant, what is your distress?"
  otherwise if user says "i need money" then
    say "Of course!"
    user pickup money
    say "You may exit using the command Lobby"
  otherwise if user says "Lobby" then
go to castleLobby
  otherwise
    say "I don't understand you"
  endif
endif

if user inside jewelRoom then
read user input
  if user says "take the crown" then
    user pickup crown

otherwise if user says "Lobby" then
    goto castleLobby
otherwise
    say "I didn't understand you"
endif
endif
endwhile
say "good bye!"

becky_castle.java

/*
 * File: GRIMM.java
 *
 * Authors: Mike Lenner, Billy Liu, Mariya Nomanbhoy, Becky Plummer
 */

class GRIMM {
    public static void main(String[] args) {
        try {
            /*get input stream to a file*/
            File fin = new File(args[0]);
            FileReader in = new FileReader(fin);

            /*create lexer and parser objects*/
            GRIMMLexer lexer = new GRIMMLexer(in);
            GRIMMParser parser = new GRIMMParser(lexer);
            GRIMMWalker walker = new GRIMMWalker();

            /*run lexer and parser on the input file*/
            parser.program();

            if (lexer.num_errors > 0) {
                System.out.println("Scanner Errors!");
                return;
            }

            if (parser.num_errors > 0) {
                System.out.println("Parser Errors!");
                return;
            }

            /*Get the AST from the parser*/
CommonAST parseTree =
(CommonAST)parser.getAST();

/*Open a window in which the AST is*/
/*displayed graphically*/
//ASTFrame frame =
//new ASTFrame("AST from the Simp parser",
//parseTree);

walker.program(parseTree);

if (walker.gRet < 0) {
    System.out.println("Walker Errors!");
    return;
}

//frame.setVisible(true);

GRIMMtranslator gt =
    new GRIMMtranslator(args[0]);
    gt.program( parseTree );
} catch(IOException e) {
    System.out.println("Error occurred while reading your broom file.");
    e.printStackTrace();
    System.out.println("exception: "+e);
} catch(antlr.RecognitionException re) {
    System.out.println("Recognition Exception occurred while reading" + " your grimm file.");
    re.printStackTrace();
    System.out.println("exception: "+re);
} catch(antlr.TokenStreamException re) {
    System.out.println("Token Stream Exception" + " occurred while reading your grimm file");
    re.printStackTrace();
    System.out.println("exception: "+re);
}
else{
    System.out.println("ERROR 6111: No program name found" + " at the command prompt");
}
return;

6.2.3 Castle Quest
The story is a GRIMM program that takes the user to a castle and then to a choice of three doors. Only one door is the correct choice, while the other two will lead to the user's doom

: note: my scene declarations
scene road
scene gate
scene dead
scene door1
scene door2
scene bigdoor
scene doorlopen
scene treasureroom
scene lair

:note: build road scene
road name "Castle Path"
road description "You are standing on a dirt path. Ahead of you in the
distance you see the Castle of the North Kingdom. You think you here
something that sounds like trumpets blaring way off in the distance. To your
left is a small dagger."
road picture "images/castle.jpg"
road exit gate
road contains item dagger

:note: build gate scene
gate name "North Castle Gate"
gate description "The castle gate looms before you. The archway appears to
be some 30 feet high. It appears deserted which seems strange. To the left
and right of the gate are two smaller doors."
gate picture "images/gate.jpg"
gate exit road
gate exit door1
gate exit door2

:note: build end scene
dead name "You Have Perished!"
dead description "You quest is over and I am afraid you have failed. Let us
only pray that you were not the last hope. Please try again."
dead picture "images/tombstone.jpg"

:note: the door on the left
door1 name "Left Castle Entrance"
door1 description "You are standing in front of a locked door. Below you
here voices yelling but you can't make out if the yelling is out of joy or
out of pain. You notice that the hinges seem to be somewhat loose, as if
someone need only pry the door open with a metal object to open it."
door1 exit treasureroom
door1 exit gate
door1 exit doorlopen is hidden
door1 picture "images/Castle_gate2.jpg"

:note: the door on the right
door2 name "Right Castle Entrance"
door2 description "A ramp leads down this entrance. You feel a cool wind
coming up from beneath. Bursts of air come out, in almost a slow, rythmic
manner."
door2 exit lair
door2 exit gate
door2 picture "images/door2.jpg"

:note: the door in the middle. This will scene only be declared to use as an
:note: exit for the gate scene. If the user goes here, they're dead.
bigdoor name "Main Gate"
:note: build dungeon scene
treasure room name "The King's Treasure Room"
treasure room description "You have entered a room filled with gold and silver as far as they eye can see. There are jewels and other priceless items all over the place."
treasure room exit door1open
treasure room contains item TreasureChest
treasure room picture "images/Treasure.jpg"

:note: dragon's lair
lair name "Underground Lair"
lair description "You are in the underground den of a giant, firebreathing dragon. His giant head swings toward you as you enter the lair. His mouth begins to glow with the fire beneath."
lair picture "images/dragon.jpg"
lair contains item Map
lair contains item DragonTooth
lair exit dead is hidden

:note: this scene is identical to door1, except that it has an added exit to the treasure room. This scene is used so that I can check whether or not the user has opened the locked door

door1open name "Left Castle Entrance"
door1open description "The door has been pried open. It is no longer locked! You see a dark hallway ahead of you which leads to what looks like a treasure room!!"
door1open exit treasure room
door1open exit gate
door1open picture "images/Castle_gate.jpg"

goto road

while not user inside treasure room
    while user inside road
        while not user has dagger
            read user input
            if user says "take dagger" then
                say "You picked up the dagger"
                user pickup dagger
            else
                say "You are going to need some protection!"
            endif
        endwhile
        while not user says "castle gate" and not user says "castlegate"
            read user input
            say "OK, on to the castle gate!"
        endwhile
    goto gate
endwhile
goto road
otherwise if user says "go to left castle entrance" then
  :note: forcing user to drop dagger allows this
  if user has dagger then
    goto door1
  otherwise
    goto doorlopen
  endif
otherwise if user says "go to right castle entrance" then
  goto door2
otherwise if user says "go to the main gain" then
  goto bigdoor
otherwise
  say "You have to choose a door."
endif
endwhile

while user inside door1
  read user input
  if user says "go to castle gate" then
    goto gate
  elseif user says "pry open door with dagger" then
    say "You stick the dagger into the door and pull.  The door comes off!  The dagger breaks and you must leave it behind."
    user drops dagger
    goto doorlopen
  elseif user says "open door" then
    say "The door is locked"
  else
    say "Are you going to try to get in there or what?"
  endif
endwhile

while user inside doorlopen
  read user input
  if user says "go to gate" then
    goto gate
  elseif user says "go to treasure room" then
    goto treasureroom
  else
    say "Go and get that treasure!"
  endif
endwhile

while user inside treasureroom
  read user input
  if user says "go to left castle entrance" then
    goto doorlopen
  elseif user says "take treasure" then
    say "You have recovered the treasure.  You are rich beyond your wildest dreams.  You win!"
    user pickup TreasureChest
    gameover
  else
    say "All that treasure and no one guarding it.  Hmmm."
  endif
endwhile
while user inside door2
    read user input
    if user says "go to castle gate" then
        goto gate
    otherwise if user says "go down ramp" then
        goto lair
    otherwise
        say "Go down the ramp or retreat."
    endif
endwhile

while user inside lair
    read user input
    if user says "kill dragon with dagger" then
        say "The giant dragon uses your dagger as a toothpick. He then laughs at you and then eats you."
        goto dead
    otherwise
        say "The giant dragon has used his poweful dragon breath. You are burnt to a crisp."
        goto dead
    endif
endwhile

if user inside dead then
    gameover
endif

if user inside bigdoor then
    say "You are jumped by 20 guards hiding in the main entrance way. How could you make such an obvious move??! The guards beat you to death."
    goto dead
endif
endwhile

mike_example.java
//This is your grimm file translated into java
public class becky_castle {
    public static void main( String args[] ) {
        User player = new User();
        Scene curScene = new Scene(null, null, null);
        //initializing game frame
        gameFrame window = new gameFrame();
        Scene _castleLobby = new Scene(null, null, null);
        Scene _throneRoom = new Scene(null, null, null);
        Scene _jewelRoom = new Scene(null, null, null);
        Character _queen = new Character(null);
        Character _maid = new Character(null);
        Item _key = new Item( "key" );
        _maid.pickUp( _key );
        _castleLobby.setName( "Entrance Hallway" );
        _castleLobby.setPicture( "castle_lobby.jpg" );
_castleLobby.setDescr( "The castle has many hidden rooms and talking to strangers may help you discover passage. Can you find the crown? The maid 'Jessica' is cleaning in the lobby. Say hello to her for her to respond" );

_castleLobby.addExit( _throneRoom, false);
_castleLobby.addExit( _jewelRoom, true );
_throneRoom.setName( "Throne Room" );
_throneRoom.setPicture( "throne_room.jpg" );
_throneRoom.setDescr( "The throne room is where the queen has audiences with members of the public. The queen is expecting you. Say Your Majesty to talk to her." );

_item = new Item( "money" );
_throneRoom.addItem( _item , false );
_jewelRoom.setName( "Hidden Jewel Vault" );
_jewelRoom.setPicture( "crown.jpg" );
_jewelRoom.setDescr( "You have found the secret Jewel Room. Look at the crown jewels aren't they beautiful?" );
_jewelRoom.addExit( _castleLobby, false);
_item = new Item( "crown" );
_jewelRoom.addItem( _item , false );
player.moveTo( _castleLobby );
curScene = _castleLobby;
while( !(player.getCurScene() == _castleLobby ) || !player.hasItem( _crown ) )
{
    if( (player.getCurScene() == _castleLobby ) )
    {
        player.setSaid( window.getInput() );
        if( player.hasItem( _key ) &
(player.getSaid().matches("(?i).*Jewel Room.*") ) )
        {
            player.moveTo( _jewelRoom );
curScene = _jewelRoom;
window.showScene( player.getCurScene(), player);
        }
        else if( (player.getSaid().matches("(?i).*hello Jessica.*") ) )
            {
            window.setOutput( "no stranger has ever spoken to me. I have keys to every room!!" );
            window.showScene(player.getCurScene(), player);
            window.setOutput( "Do you have any money? I can point you to palace secrets." );
            window.showScene(player.getCurScene(), player);
        }
        else if( (player.getSaid().matches("(?i).*where do I get money?..*")) )
            {
            window.setOutput( "Well you could ask the queen of course, but don't tell her I sent you!" );
            window.showScene(player.getCurScene(), player);
        }
    } else if( (player.getCurScene() == _castleLobby ) )
    {
        player.setSaid( window.getInput() );
        if( player.hasItem( _key ) &
(player.getSaid().matches("(?i).*Jewel Room.*") ) )
        {
            player.moveTo( _jewelRoom );
curScene = _jewelRoom;
window.showScene( player.getCurScene(), player);
        }
        else if( (player.getSaid().matches("(?i).*hello Jessica.*") ) )
            {
            window.setOutput( "no stranger has ever spoken to me. I have keys to every room!!" );
            window.showScene(player.getCurScene(), player);
            window.setOutput( "Do you have any money? I can point you to palace secrets." );
            window.showScene(player.getCurScene(), player);
        }
        else if( (player.getSaid().matches("(?i).*where do I get money?..*")) )
            {
            window.setOutput( "Well you could ask the queen of course, but don't tell her I sent you!" );
            window.showScene(player.getCurScene(), player);
        }
    }
}
else if( player.hasItem( _money ) && (player.getSaid().matches("(?i).*I have money.*") ) )
{
    player.drop( _money );
    window.showScene(player.getCurScene(),
    player);
    _maid.pickUp( _money, curScene );
    window.showScene(player.getCurScene(),
    player);
    _maid.drop( _key, curScene );
    window.showScene(player.getCurScene(),
    player);
    player.pickUp( _key );
    window.showScene(player.getCurScene(),
    player);
    window.setOutput( "You now have a key.
There is a hidden Room door. To open it type 'Jewel Room'" );
    window.showScene(player.getCurScene(),
    player);
}
else if((player.getSaid().matches("(?i).*throne room.*") ) )
{
    player.moveTo( _throneRoom );
    curScene = _throneRoom;
    window.showScene( player.getCurScene(),
    player);

}
else
{
    window.setOutput( "I don't understand
you. Did you use punctuation?" );
    window.showScene(player.getCurScene(),
    player);
}

if( (player.getCurScene() == _throneRoom ) )
{
    player.setSaid( window.getInput() );
    if( (player.getSaid().matches("(?i).*Your
Majesty.*") ) )
    {
        window.setOutput( "Greetings peasant,
what is your distress?" );
        window.showScene(player.getCurScene(),
        player);
    }
else if( (player.getSaid().matches("(?i).*i
need money.*") ) )
    {
        window.setOutput( "Of course!" );
        window.showScene(player.getCurScene(),
        player);
        player.pickUp( _money );
        window.showScene(player.getCurScene(),
        player);

}
window.setOutput( "You may exit using the command Lobby" );
player;
} else if( (player.getSaid().matches("(?i).*Lobby.*") ) )
{
    player.moveTo( _castleLobby );
curScene = _castleLobby;
window.showScene( player.getCurScene(),
player);
}
else
{
    window.setOutput( "I don't understand you" );
window.showScene(player.getCurScene(),
player);
}

if( (player.getCurScene() == _jewelRoom ) )
{
    player.setSaid( window.getInput() );
    if( (player.getSaid().matches("(?i).*take the crown.*") ) )
    {
        player.pickUp( _crown );
window.showScene(player.getCurScene(),
player);
}
else if( (player.getSaid().matches("(?i).*Lobby.*") ) )
{
    player.moveTo( _castleLobby );
curScene = _castleLobby;
window.showScene( player.getCurScene(),
player);
}
else
{
    window.setOutput( "I didn't understand you" );
window.showScene(player.getCurScene(),
player);
}
}

window.setOutput( "good bye!" );
window.showScene(player.getCurScene(), player);


6.2.4 Pleasing your kid
:note: Make your kid happy before it's too late!
:note: This storyboard revolves around you trying to make your kid happy. There are two ways to do this as you'll discover, as well as a few pitfalls.

scene start
scene error
scene dead
scene editor
scene statements
scene mcdonalds
scene table
scene home
scene grimm

character teenager
teenager holds item BigMac
teenager holds item Chicken_Nuggets
teenager holds item Toy

start name "In front of the computer"
start description "You're at the computer monitor trying to write a GRIMM story board for your kid. He is sitting impatiently next to you. You need to decide on what kind of story to write, what editor to use, and other important things like that.\n
What editor do you want to use?"
start picture "computer.gif"
start contains item One_Dollar
start contains item Fifty_Cents
start exit dead is hidden
start exit editor is hidden

error name "Error encountered!"
error description "You've come across and error in your program"

finished name "Your kid is happy!"
finished description "You give your kid the toy and he is now happy. In fact, he's so happy that he forgot about the GRIMM storyboard and you're off the hook! Congradulations!"
finished picture "finished.jpg"

grimm name "Program Written!"
grimm picture "grimm.jpg"
grimm description "You've successfully written, compiled and run your GRIMM program! Your kid spent all night playing it and is happy. Congradulations!"

dead name "Poor Choice!"
dead description "Because of your choice, it takes you forever to properly write your GRIMM program. By the time you finish, your child is sad and has gotten tired of waiting for you to write a story. What kind of parent are you?"
dead picture "sad.jpg"

editor name "Using Vi as your editor"
editor description "You've fired up vim for your editor and now you're ready to start writing your GRIMM program. The first thing you need to do is declare some scenes, characters and items. Once you're finished declaring
them all, make sure to make a :note: to comment the end of your declarations so you can start writing your statements.

```
editor picture "vim.jpg"
```

editor exit start is hidden
editor exit statements is hidden

```
statements name "Hungry!"
statements description "You spend a while making all those declarations. You need to declare some statements. But before that, you've gotten hungry and need to get food. You start thinking of where to go to get food..."
statements picture "food.gif"
statements exit mcdonalds is hidden
```

```
mcdonalds name "McDonalds"
mcdonalds picture "mcdonalds.jpg"
mcdonalds description "You drive to your favorite McDonalds restaurant. It is a familiar sight: plastic counters, plastic chairs, pictures on the menu, poor high school students behind the cash registers. Maybe you should order quickly before your conscience gets to you."
mcdonalds exit home
mcdonalds exit table
mcdonalds exit dead is hidden
```

```
home name "Back at Home"
home description "You're drive back home, realizing you spent so much time at McDonalds while your kid is waiting for your GRIMM storyboard. He's still sitting here, waiting patiently for your to write the game."
home picture "home.gif"
home exit grimm is hidden
home exit finished is hidden
```

```
table name "Table at McDonalds"
```

```
:note: program starts here

```

```
goto start
user pickup One_Dollar
user pickup Fifty_Cents
```

```
while not user inside finished
  while user inside start
    read user input
    if user says "vim" or user says "vi" or user says "gvim" then
go to editor
  elseif user says "emacs" or user says "xemacs" then
go to dead
  otherwise
    say "Unfortunately, that editor is not installed on your machine. Maybe you should choose a more popular editor?"
  endif
  endwhile
```

```
while user inside editor
  read user input
  if user says "character" then
    say "you've declared a character"
  elseif user says "scene" then
```
say "you've declared a scene"
otherwiseif user says "contains" or user says "item" then
    say "you've declared an item"
otherwiseif user says ":note:" then
goto statements
otherwise
    say "you should declare characters, scenes and items first"
endif
endwhile

while user inside statements
read user input
if user says "mcdonalds" then
goto mcdonalds
otherwise
    say "no, you don't feel like going there. maybe you should go to your favorite fast food restaurant?"
endif
endwhile

while user inside mcdonalds
say "the teenager says: Welcome to McDonalds, how can I help you?"
read user input
if user says "big mac" and user has One_Dollar then
    say "the teenager says: Thank you for your order."
    say "you give the teenager $1 for the big mac"
    teenager drops BigMac
    user drops One_Dollar
    teenager pickup One_Dollar
    user pickup BigMac
otherwiseif user says "big mac" and not user has One_Dollar then
    say "you don't have enough money to buy a big mac"
otherwiseif user says "nuggets" and user has Fifty_Cents then
    say "the teenager says: Thank you for your order."
    say "you give the teenager $.50 for your chicken nuggets"
    teenager drops Chicken_Nuggets
    user drops Fifty_Cents
    teenager pickup Fifty_Cents
    user pickup Chicken_Nuggets
otherwiseif user says "nuggets" and not user has Fifty_Cents then
    say "you don't have enough money to buy the nuggets"
otherwiseif user says "toy" then
    if user has Fifty_Cents then
        say "the teenager says: Thank you for your order"
        say "you give the teenager $.50 for your toy."
        teenager drops Toy
        user drops Fifty_Cents
        teenager pickup Fifty_Cents
        user pickup Toy
    otherwise
        say "you don't have enough money to buy the toy"
endif
otherwise
    say "maybe you should order a big mac, chicken nuggets, or a kids toy?"
endif

if user says "home" then
    goto home
endif

if user says "table" then
    goto dead
endif

endwhile

while user inside home
    read user input
    if user has Toy and user says "toy" then
        goto finished
    otherwiseif user says "story" or user says "write" or user says "program" or user says "grimm" then
        goto grimm
    otherwise
        say "you need to do something quick to satisfy your kid"
    endif
endwhile

if user inside dead then
    gameover
endif

endwhile

kid.java
//This is your grimm file translated into java
public class kid {
    public static void main( String args[] ) {
        User player = new User();
        Scene curScene = new Scene(null, null, null);
        //initializing game frame
        gameFrame window = new gameFrame();
        Scene _start = new Scene(null, null, null);
        Scene _error = new Scene(null, null, null);
        Scene _finished = new Scene(null, null, null);
        Scene _dead = new Scene(null, null, null);
        Scene _editor = new Scene(null, null, null);
        Scene _statements = new Scene(null, null, null);
        Scene _mcdonalds = new Scene(null, null, null);
        Scene _table = new Scene(null, null, null);
        Scene _home = new Scene(null, null, null);
        Scene _grimm = new Scene(null, null, null);
        Character _teenager = new Character(null);
        Item _BigMac = new Item( "BigMac" );
        _teenager.pickUp( _BigMac );
        Item _Chicken_Nuggets = new Item( "Chicken_Nuggets" );
_teenager.pickUp( _Chicken_Nuggets );
Item _Toy = new Item( "Toy" );
_teenager.pickUp( _Toy );
_start.setName( "In front of the computer" );
_start.setDescr( "You're at the computer monitor trying to write a GRIMM story board for your kid. He is sitting impatiently next to you. You need to decide on what kind of story to write, what editor to use, and other important things like that.\n\nWhat editor do you want to use?" );
_start.setPicture( "computer.gif" );
Item _One_Dollar = new Item( "One_Dollar" );
_start.addItem( _One_Dollar , false );
Item _Fifty_Cents = new Item( "Fifty_Cents" );
_start.addItem( _Fifty_Cents , false );
_start.addExit( _dead , true );
_start.addExit( _editor , true );
_error.setName( "Error encountered!" );
_error.setDescr( "You've come across and error in your program" );
_finished.setName( "Your kid is happy!" );
_finished.setDescr( "You give your kid the toy and he is now happy. In fact, he's so happy that he forgot about the GRIMM storyboard and you're off the hook! Congratulations!" );
_finished.setPicture( "finished.jpg" );
grimm.setName( "Program Written!" );
grimm.setPicture( "grimm.jpg" );
grimm.setDescr( "You've successfully written, compiled and run your GRIMM program! Your kid spent all night playing it and is happy. Congratulations!" );
_dead.setName( "Poor Choice!" );
_dead.setDescr( "Because of your choice, it takes you forever to properly write your GRIMM program. By the time you finish, your child is sad and has gotten tired of waiting for you to write a story. What kind of parent are you?" );
_dead.setPicture( "sad.jpg" );
_editor.setName( "Using Vi as your editor" );
_editor.setDescr( "You've fired up vim for your editor and now you're ready to start writing your GRIMM program. The first thing you need to do is declare some scenes, characters and items. Once you're finished declaring them all, make sure to make a :note: to comment the end of your declarations so you can start writing your statements." );
_editor.setPicture( "vim.jpg" );
_editor.addExit( _start , true );
_editor.addExit( _statements , true );
_statements.setName( "Hungry!" );
_statements.setDescr( "You spend a while making all those declarations. You need to declare some statements. But before that, you've gotten hungry and need to get food. You start thinking of where to go to get food..." );
_statements.setPicture( "food.gif" );
_statements.addExit( _mcdonalds , true );
mcdonalds.setName( "McDonalds" );
mcdonalds.setPicture( "mcdonalds.jpg" );
mcdonalds.setDescr( "You drive to your favorite McDonalds restaurant. It is a familiar sight: plastic counters, plastic chairs, pictures on the menu, poor high school students behind the cash registers. Maybe you should order quickly before your conscience gets to you." );
mcdonalds.addExit( _home , false );
_mcdonalds.addExit( _table, false);
_mcdonalds.addExit( _dead, true );
_home.setName( "Back at Home");  
_home.setDescr( "You're drive back home, realizing you spent so
much time at McDonalds while your kid is waiting for your GRIMM storyboard.
He's still sitting here, waiting patiently for your to write the game." );
_home.setPicture( "home.gif" );
_home.addExit( _grimm, true );
_home.addExit( _finished, true );
_table.setName("Table at McDonalds");
player.moveTo( _start );
curScene = _start;
window.showScene( player.getCurScene(), player);
player.pickUp( _One_Dollar );
window.showScene(player.getCurScene(), player);
player.pickUp( _Fifty_Cents );
window.showScene(player.getCurScene(), player);
while( !(player.getCurScene() == _finished ) ) {
    while( (player.getCurScene() == _start ) )
    {
        player.setSaid( window.getInput() );
        if( (player.getSaid().matches("(?i).*vim.*") ) ||
            (player.getSaid().matches("(?i).*vi.*") ) ||
            (player.getSaid().matches("(?i).*gvim.*") ) )
        {
            player.moveTo( _editor );
            curScene = _editor;
            window.showScene( player.getCurScene(),
            player);
        }
        else if( (player.getSaid().matches("(?i).*emacs.*") )
            || (player.getSaid().matches("(?i).*xemacs.*") ) )
        {
            player.moveTo( _dead );
            curScene = _dead;
            window.showScene( player.getCurScene(),
            player);
        }
        else
        {
            window.setOutput( "Unfortunately, that editor
is not installed on your machine.  Maybe you should choose a more popular
editor?" );
            window.showScene(player.getCurScene(), player);
        }
    }
    while( (player.getCurScene() == _editor ) )
    {
        player.setSaid( window.getInput() );
        if( (player.getSaid().matches("(?i).*character.*") )
        {
            window.setOutput( "you've declared a character"
        );
            window.showScene(player.getCurScene(), player);
        }
else if( (player.getSaid().matches("(?i).*scene.*") ) )
{
    window.setOutput( "you've declared a scene" );
    window.showScene(player.getCurScene(), player);
}
else if( (player.getSaid().matches("(?i).*contains.*") ) ||
(player.getSaid().matches("(?i).*item.*") ) )
{
    window.setOutput( "you've declared an item" );
    window.showScene(player.getCurScene(), player);
}
else if( (player.getSaid().matches("(?i).*:note:*") ) )
{
    player.moveTo( _statements );
    curScene = _statements;
    window.showScene( player.getCurScene(), player);
}
else
{
    window.setOutput( "you should declare characters, scenes and items first" );
    window.showScene(player.getCurScene(), player);
}
while( (player.getCurScene() == _statements ) )
{
    player.setSaid( window.getInput() );
    if( (player.getSaid().matches("(?i).*mcdonalds.*") ) )
    {
        player.moveTo( _mcdonalds );
        curScene = _mcdonalds;
        window.showScene( player.getCurScene(), player);
    }
    else
    {
        window.setOutput( "no, you don't feel like going there. maybe you should go to your favorite fast food restaurant?" );
        window.showScene(player.getCurScene(), player);
    }
}
while( (player.getCurScene() == _mcdonalds ) )
{
    window.setOutput( "the teenager says: Welcome to McDonalds, how can I help you?" );
    window.showScene(player.getCurScene(), player);
    player.setSaid( window.getInput() );
    if( (player.getSaid().matches("(?i).*big mac.*") ) &&
player.hasItem( _One_Dollar ) )
    {
        window.setOutput( "the teenager says: Thank you for your order." );
    }
window.showScene(player.getCurScene(), player);

window.setOutput("you give the teenager $1 for

the big mac");

window.showScene(player.getCurScene(), player);
_teenager.drop( _BigMac, curScene );
window.showScene(player.getCurScene(), player);
player.drop( _One_Dollar );
window.showScene(player.getCurScene(), player);
_teenager.pickUp( _One_Dollar, curScene );
window.showScene(player.getCurScene(), player);
player.pickUp( _BigMac );
window.showScene(player.getCurScene(), player);

} else if( (player.getSaid().matches("(?i).*big mac.*")
 ) && !player.hasItem( _One_Dollar ) )
{
    window.setOutput("you don't have enough money
to buy a big mac");
}
else if( (player.getSaid().matches("(?i).*nuggets.*")
 ) && player.hasItem( _Fifty_Cents ) )
{
    window.setOutput("the teenager says: Thank you
    for your order.");
window.showScene(player.getCurScene(), player);
window.setOutput("you give the teenager .50
for your chicken nuggets");
window.showScene(player.getCurScene(), player);
    _teenager.drop( _Chicken_Nuggets, curScene );
window.showScene(player.getCurScene(), player);
    player.drop( _Fifty_Cents );
window.showScene(player.getCurScene(), player);
    _teenager.pickUp( _Fifty_Cents, curScene );
window.showScene(player.getCurScene(), player);
    player.pickUp( _Chicken_Nuggets );
window.showScene(player.getCurScene(), player);
}
else if( (player.getSaid().matches("(?i).*nuggets.*")
 ) && !player.hasItem( _Fifty_Cents ) )
{
    window.setOutput("you don't have enough money
to buy the nuggets");
}
else if( (player.getSaid().matches("(?i).*toy.*") ) )
{
    if( player.hasItem( _Fifty_Cents ) )
    {
        window.setOutput("the teenager says:
Thank you for your order");
    player);
    window.showScene(player.getCurScene(),
.
.50 for your toy.");
}
else if( (player.getSaid().matches("(?i).*toy.*") ) )
{
    if( player.hasItem( _Fifty_Cents ) )
    {
        window.setOutput("the teenager says:
Thank you for your order");
    player);
    window.showScene(player.getCurScene(),
.
.50 for your toy.");
}
else if( (player.getSaid().matches("(?i).*toy.*") ) )
{
    if( player.hasItem( _Fifty_Cents ) )
    {
        window.setOutput("the teenager says:
Thank you for your order");
    player);
    window.showScene(player.getCurScene(),
.
.50 for your toy.");
}
else if( (player.getSaid().matches("(?i).*toy.*") ) )
{
    if( player.hasItem( _Fifty_Cents ) )
    {
        window.setOutput("the teenager says:
Thank you for your order");
    player);
    window.showScene(player.getCurScene(),
.
.50 for your toy.");
}
_teenager.drop( _Toy, curScene );
window.showScene(player.getCurScene(), player);
player.drop( _Fifty_Cents );
window.showScene(player.getCurScene(), player);
_teenager.pickUp( _Fifty_Cents, curScene );
window.showScene(player.getCurScene(), player);
player.pickUp( _Toy );
window.showScene(player.getCurScene(), player);
}
else
{

window.setOutput( "you dont have enough money to buy the toy" );
window.showScene(player.getCurScene(), player);
}
else
{

window.setOutput( "maybe you should order a big mac, chicken nuggets, or a kids toy?" );
window.showScene(player.getCurScene(), player);
}
if( (player.getSaid().matches("(?i).*home.*") ) )
{

player.moveTo( _home );
curScene = _home;
window.showScene( player.getCurScene(), player);
}
if( (player.getSaid().matches("(?i).*table.*") ) )
{

player.moveTo( _dead );
curScene = _dead;
window.showScene( player.getCurScene(), player);
}
while( (player.getCurScene() == _home ) )
{
player.setSaid( window.getInput() );
if( player.hasItem( _Toy ) &&
(player.getSaid().matches("(?i).*toy.*") ) )
{

player.moveTo( _finished );
curScene = _finished;
window.showScene( player.getCurScene(), player);
}
else if( (player.getSaid().matches("(?i).*story.*") ) ||
(player.getSaid().matches("(?i).*write.*") ) ||
(player.getSaid().matches("(?!.*program.*)") ||
(player.getSaid().matches("(?!.*grimm.*)") ) )
{
    player.moveTo( _grimm );
    curScene = _grimm;
    window.showScene( player.getCurScene(),
    player);
}
else
{
    window.setOutput( "you need to do something quick to satisfy your kid" );
    window.showScene(player.getCurScene(), player);
}

if( (player.getCurScene() == _dead ) )
{
    window.setOutput(" GAMEOVER");
    window.showScene(player.getCurScene(), player);
    while(true) {}; // infinite loop
}

6.3 Test Suites
We created tests scripts that test each aspect of our language. Each test consists of a .gmm file containing the GRIMM source and a .tst file which contains the expected error output. In the case of successful compilation we expect no error message therefore we have no .tst file. In the case of unsuccessful compilation we get an error message which is checked against the expected output stored in the .tst file. Under each category we have various tests listed in the order .gmm file followed by the .tst file.

6.3.1 Lexer Test Scripts
bad.gmm
:note: this should fail because tokens must start with an alpha
12say

bad.tst
Token Stream Exception occurred while reading your grimm file
line 3:1: unexpected char: '1'
at GRIMMLexer.nextToken(GRIMMLexer.java:155)
at antlr.TokenBuffer.fill(TokenBuffer.java:69)
at antlr.TokenBuffer.LA(TokenBuffer.java:80)
at antlr.LLkParser.LA(LLkParser.java:52)
at GRIMMParser.decls(GRIMMParser.java:1237)
at GRIMMParser.program(GRIMMParser.java:1305)
at GRIMM.main(GRIMM.java:31)
exception: line 3:1: unexpected char: '1'
bad2.gmm
:note: common semicolon check

scene foo;

say "there are no semicolons in GRIMM"

bad2.tst
Token Stream Exception occured while reading your grimm file
line 3:10: unexpected char: ';' 
   at GRIMMLexer.nextToken(GRIMMLexer.java:155)  
   at antlr.TokenBuffer.fill(TokenBuffer.java:69)  
   at antlr.TokenBuffer.LA(TokenBuffer.java:80)  
   at antlr.LLkParser.LA(LLkParser.java:52)  
   at GRIMMParser.scene_decl(GRIMMParser.java:84)  
   at GRIMMParser.decls(GRIMMParser.java:1240)  
   at GRIMMParser.program(GRIMMParser.java:1305)  
   at GRIMM.main(GRIMM.java:31)
exception: line 3:10: unexpected char: ';' 

good.gmm
:note: this is simply a print statement that should pass the lexer

say "test successful"

underscore.gmm
:note: this test makes sure the lexer accepts underscores

character edwards_teacher

say "this should pass"

6.3.2 Parser Test Scripts

boolean.gmm
:note: boolean expressions

if user says "foo" and not user says "bar" or user says "foo2" 
then
  gameover
endif

character.gmm
:note: this tests that we can declare characters

character Edwards

say "character is declared"

gameover_bad.gmm
:note: this tests a common mistake with gameover.  it should be 'gameover'
game over

gameover_bad.tst
line 3:1: unexpected token: game
Parser Errors!

gameover.gmm
:note: this tests the gameover statement
gameover
goto.gmm
:note: this tests the goto statement
scene classroom
classroom name "Classroom"
goto classroom

hidden_exit.gmm
:note: this tests that the hidden is parsed correctly
scene classroom
scene office
classroom name "Classroom"
classroom exit office is hidden
office name "Edward's office"
say "office is a hidden exit"

hidden_key.gmm
:note: this script tests that we can hide items
scene classroom
classroom name "Classroom"
classroom contains item key is hidden
say "hidden key in classroom"

holds.gmm
:note: this tests that characters can hold items
character edwards
edwards holds item grades
say "edwards holds the item grades"

if_fail.gmm
:note: this tests a common syntax errors with the if statement

if user says "foo" then
    say "you said foo"

say "these is missing the 'endif' "

if_fail.tst
line 7:1: unexpected token: null
line 7:1: unexpected token: null
Parser Errors!

if_fail2.gmm
:note: this test common syntax errors with the if statement

if user says "foo" then
    say "you said foo"
otherwise if user says "bar" then
    say "you said bar"
endif

say "otherwise if should be one word"

if_fail2.tst
line 5:11: unexpected token: if
Parser Errors!

if_fail3.gmm
:note: this test common syntax errors with the if statement

if user says "foo" then
    say "you said foo"
otherwiseif user says "bar"
    say "you said bar"
endif

say "otherwiseif is missing 'then'"

if_fail3.tst
line 5:28: expecting "then", found '

Parser Errors!

if_fail4.gmm
:note: this test common syntax errors with the if statement

if user says "foo" then
    say "you said foo"
otherwiseif user says "bar" then
    say "you said bar"
otherwise then
    say "you said nothing"
endif
say "otherwise should not have 'then'"

if_fail4.tst
line 7:11: unexpected token: then
Parser Errors!

if_fail5.gmm
:note: this test common syntax errors with the if statement

if user says "foo" then
  if user says "foo2" then
    say "you said foo2"
otherwise if user says "bar" then
  say "you said foo"
endif

say "nested if is missing 'endif'"

if_fail5.tst
line 11:1: unexpected token: null
line 11:1: unexpected token: null
Parser Errors!

if_fail6.gmm
:note: this test common syntax errors with the if statement

if user says "foo"
  say "you said foo"
endif

say "these is missing the 'then' "

if_fail6.tst
line 3:19: expecting "then", found '

line 5:1: expecting EOF, found 'endif'
Parser Errors!

if.gmm
:note: this tests a complex, proper if statement

if user says "fool" then

  say "fool"
otherwise if user says "foo2" then
  say "foo2"
otherwise if user says "foo bar" then
  if user says "bar1" then
    say "bar1"
  otherwise if user says "bar2" then
    say "bar2"
  otherwise
    say "bar3"
endif
otherwise
  say "foo3"
endif

say "this is a proper if statement"

**item_fail.gmm**

:note: parser error on item

scene classroom

classroom name "Classroom"
classroom contains key

say "it should be: classroom contains item key"

**item_fail.tst**

line 6:20: expecting "item", found 'key'

Parser Errors!

**keywords.gmm**

:note: this test makes sure keywords cant be used as ids

character character
character scene
character holds
character item
character contains
character is
character hidden
character name
character description
character picture
character exit
character user
character says
character has
character inside
character and
character if
character then
character endif
character elseif
character otherwise
character while
character endwhile
character goto
character read
character user
character input
character pickup
character drops
character gameover

scene character
scene scene
scene holds
scene item
scene contains
scene is
scene hidden
scene name
scene description
scene picture
scene exit
scene user
scene says
scene has
scene inside
scene and
scene if
scene then
scene endif
scene otherwiseif
scene otherwise
scene while
scene endwhile
scene goto
scene read
scene user
scene input
scene pickup
scene drops
scene gameover

gameover

keywords.tst
line 3:11: expecting NOUN, found 'character'
line 4:11: expecting NOUN, found 'scene'
line 5:11: expecting NOUN, found 'holds'
line 6:11: expecting NOUN, found 'item'
line 7:11: expecting NOUN, found 'contains'
line 8:11: expecting NOUN, found 'is'
line 9:11: expecting NOUN, found 'hidden'
line 10:11: expecting NOUN, found 'name'
line 11:11: expecting NOUN, found 'description'
line 12:11: expecting NOUN, found 'picture'
line 13:11: expecting NOUN, found 'exit'
line 14:11: expecting NOUN, found 'user'
line 15:11: expecting NOUN, found 'says'
line 16:11: expecting NOUN, found 'has'
line 17:11: expecting NOUN, found 'inside'
line 18:11: expecting NOUN, found 'and'
line 19:11: expecting NOUN, found 'if'
line 20:11: expecting NOUN, found 'then'
line 21:11: expecting NOUN, found 'endif'
line 22:11: expecting NOUN, found 'otherwiseif'
line 23:11: expecting NOUN, found 'otherwise'
line 24:11: expecting NOUN, found 'while'
line 25:11: expecting NOUN, found 'endwhile'
no_stmt.gmm
:note: this tests that a program needs at least one statement

character Edwards

no_stmt.tst
line 4:1: unexpected token: null
Parser Errors!

pickup.gmm
:note: this tests pickup

scene classroom

classroom name "A classroom"
classroom contains item chalk
user pickup chalk

**scene_decl.gmm**
:note: this simply declares a scene

scene classroom
classroom name "A Classroom"
say "test finished with success"

**scene descr.gmm**
:note: declares a scene and sets a description for the scene

scene classroom
classroom name "A classroom"
classroom description "The PLT classroom is on the 5th floor of Mudd"
say "test finished with success"

**scene_exit.gmm**
:note: declares two scenes and makes one scene an exit for the other

scene classroom
scene office

classroom name "A Classroom"
classroom exit office

office name "An office"
say "test finished with success"

**scene_item.gmm**
:note: declares a scene and gives it an item

scene classroom
classroom name "A classroom"
classroom contains item chalk

say "test finished with success"

**scene_name.gmm**
:note: declares a scene and sets a name for the scene

scene classroom
classroom name "COMSW4115 Classroom"
say "test finished with success"

**scene_pic.gmm**
:note: declares a scene and gives it a picture
scene classroom

classroom name "A classroom"
classroom picture "classpic.jpg"

say "test finished with success"

**strconst.gmm**
:note: this tests that string constants must be used in certain places

scene classroom

classroom name The Classroom
classroom description The Description
classroom picture file
if user says foo then
    gameover
endif

**strconst.tst**
line 5:16: expecting STRCONST, found 'The'
line 5:20: unexpected token: Classroom
line 6:1: unexpected token: classroom
line 7:1: unexpected token: classroom
line 8:14: expecting STRCONST, found 'foo'
Parser Errors!

**while_fail1.gmm**
:note: this has some common while parse errors

:note: there has to be at least one statement in a while
while user says "foo"
endwhile

:note: missing an 'endwhile'
while user says "foo"
    read user input

**while_fail1.tst**
line 5:1: unexpected token: endwhile
line 10:1: expecting "endwhile", found 'null'
line 10:1: unexpected token: null
Parser Errors!

**while_fail2.gmm**
:note: this makes sure there is a boolean in the while statement

character edwards
scene classroom

classroom name "Classroom"
classroom contains item chalk
while user
  say "foo"
endwhile

while edwards
  say "bar"
endwhile

while classroom
  say "boo"
endwhile

while chalk
  say "zoo"
endwhile

while read user input
  say "poo"
endwhile

while not read user input
  say "doo"
endwhile

while_fail2.tst
line 9:7: unexpected token: user
line 13:7: unexpected token: edwards
line 17:7: unexpected token: classroom
line 21:7: unexpected token: chalk
line 25:7: unexpected token: read
line 29:11: unexpected token: read
Parser Errors!

while.gmm
:note: this tests some proper, complex while statements

character edwards

edwards holds item grades

while user has grades
  say "GRIMM gets an A+
endwhile

while user says "foo"
  read user input
endwhile

while not user says "foo"
  read user input
endwhile
6.3.3 Walker Test Scripts

**char_item_typechk1.gmm**

:note: type checking between character and item

scene classroom
character edwards

classroom name "Classroom"
classroom contains item edwards

user pickup edwards
user drops edwards
if user has edwards then
    say "foo"
endif
while user has edwards
    say "foo"
endwhile

say "these fails because edwards is a character, not an item"

**char_item_typechk1.tst**

Semantic Error on 7: edwards already declared
Semantic Error on 9: edwards not declared or not of type item
Semantic Error on 10: edwards not declared or not of type item
Semantic Error on 11: edwards not declared or not of type item
Semantic Error on 14: edwards not declared or not of type item
Walker Errors!

**char_item_typechk2.gmm**

:note: type checking between character and item

scene classroom
character edwards

classroom name "Classroom"
classroom contains item chalk

chalk holds item edwards
chalk holds item chalk
chalk holds item classroom

say "these fail because chalk is an item type, not a character type"

**char_item_typechk2.tst**

Semantic Error on 9: chalk not declared or is not of type character
Semantic Error on 10: chalk not declared or is not of type character
Semantic Error on 11: chalk not declared or is not of type character
Walker Errors!

**exit_dup.gmm**
:note: foo has the bar exit twice. this is a semantic error.

scene foo
scene bar

bar name "Bar"

foo name "Foo"
foo exit bar
foo exit bar

gameover

**exit_dup.tst**
Semantic Error on 10: foo already has bar as an exit
Walker Errors!

**exit_fail.gmm**

:note: this test fails because the exit is not defined before it is used

scene classroom

classroom name "A classroom"
classroom exit office

say "bad test"

**exit_fail.tst**
Semantic Error on 6: office not declared or is not of type scene.
Walker Errors!

**goto_fail.gmm**

:note: the goto statement fails because the scene has not been declared

scene classroom
goto office

**goto_fail.tst**
Semantic Error on 5: office not declared or not of type scene
Semantic Error: Scene classroom doesn't have a name defined
Walker Errors!

**goto_typechk.gmm**

:note: type check for scene, item and character

scene classroom
character edwards

classroom name "Classroom"
classroom contains item chalk
goto classroom
goto edwards
goto chalk

say "goto chalk and goto edwards fails because they are not scenes"

**goto_typechk.tst**
Semantic Error on 10: edwards not declared or not of type scene
Semantic Error on 11: chalk not declared or not of type scene
Walker Errors!

**hidden_fail.gmm**
:note: the hidden keyword fails because the scene is not declared

scene classroom

classroom name "Classroom"
classroom exit office is hidden

say "hidden fails because office is not declared"

**hidden_fail.tst**
Semantic Error on 6: office not declared or is not of type scene.
Walker Errors!

**holds_bad.gmm**
:note: this fails because the character is not defined

edwards holds item grades

say "edwards is not defined as a character"

**holds_bad.tst**
Semantic Error on 3: edwards not declared or is not of type character
Walker Errors!

**id_dup.gmm**
:note: this test fails because all Things must have unique identifiers

scene classroom

classroom name "Classroom"
classroom character classroom

say "id problem"

**id_dup.tst**
Semantic Error on 6: Identifier classroom already used
Walker Errors!
item_assign.gmm
:note: checks the holds keyword

scene foo
category bar

foo name "Foo"
foo holds item zoo

say "it should be foo contains item zoo"

item_assign.tst
Semantic Error on 7: foo not declared or is not of type character
Walker Errors!

item_test1.gmm
:note: this test makes sure we can't have the same id

scene classroom
category edwards

classroom name "Classroom"
classroom contains item chalk

edwards holds item chalk

say "this is bad"

item_test1.tst
Semantic Error on 9: chalk already declared
Walker Errors!

name_bad.gmm
:note: this fails the semantic analysis because all scenes must
declare names

scene classroom

say "test fails"

name_bad.tst
Semantic Error: Scene classroom doesn't have a name defined
Walker Errors!

name_undecl.gmm
:note: the scene is not named

scene classroom

say "classroom needs a name"

name_undecl.tst
Semantic Error: Scene classroom doesn't have a name defined
Walker Errors!
recur_scene.gmm
:note: GRIMM allows for recursive scenes

scene foo

foo name "Foo"
foo exit foo

say "foo has an exit to itself"

scene_char_typechk1.gmm
:note: type checking for scenes and characters

scene foo
character bar

foo name "Foo"
foo exit bar

say "bar is a character, not a scene"

scene_char_typechk1.tst
Semantic Error on 7: bar not declared or is not of type scene.
Walker Errors!

scene_char_typechk3.gmm
:note: type checking between character and scene

character foo
scene bar

bar name "Bar"

foo name "Foo"
foo description "Foo Description"
foo picture "/pic.jpg"
foo contains item bar
foo exit bar

say "these are errors because foo is not a scene"

scene_char_typechk3.tst
Semantic Error on 8: Cannot assign name "Foo" to scene foo because foo is not declared or is not of type scene.
Semantic Error on 9: foo not declared or is not of type scene.
Semantic Error on 10: foo not declared or is not of type scene.
Semantic Error on 11: foo not declared or is not of type scene
Semantic Error on 12: foo not declared or is not of type scene
Walker Errors!

scene_dup.gmm
:note: this test fails because all Things must have unique identifiers
scene classroom

classroom name "Classroom"
character classroom

say "id problem"

scene_dup.tst
Semantic Error on 6: Identifier classroom already used
Walker Errors!

scene_full.gmm
:note: full scene declaration with name, description, picture, exit and items

scene classroom
scene office

classroom name "W4115 Classroom"
classroom description "The classroom is on the 5th floor of Mudd"
classroom picture "classpic.jpg"
classroom exit office
classroom contains item chalk

office name "Steve Edwards Office"
office description "Office is on the 4th floor of CS building"
office picture "mugshot.jpg"
office contains item compiler

say "success"

scene_item_typechk1.gmm
:note: type checking for scenes and items

scene foo
scene bar

foo name "Foo"
foo contains item bar

say "bar is a scene, not an item"

scene_item_typechk1.tst
Semantic Error on 7: bar already declared
Semantic Error: Scene bar doesn't have a name defined
Walker Errors!

scene_item_typechk2.gmm
:note: type check between item and scene

scene foo

foo name "Foo"

user drops foo
say "this should fail because foo is not at item"

scene_item_typechk2.tst
Semantic Error on 7: foo not declared or not of type item
Walker Errors!

6.3.4 Translator Tests
javakeyword.gmm
:note: this tests that the translator is mangling the variables
so it doesn't
:note: conflict with the java compiler

class character

gameover
7. Lessons Learned

7.1 Individual Lessons

7.1.1 Mike
The most important piece of knowledge that I wished I had possessed when beginning this project was the overall, big picture of the compiler design. Knowing beforehand how each component worked and what functionality it needed to implement would have been a huge help. Instead, I found myself trying to build components by feel, adding code in what looked to be right, only to discover afterward what that code should have been doing. If given the chance to repeat this project, I would have first begun with an outline of all code to be written. Having each component's functionality sketched out before beginning development would have been the better way to approach this.

In designing the GUI GameFrame class, I was forced to do an almost complete redesign of the layout towards the end of the project. Unfortunately, when I had begun development, my knowledge of the Java Swing libraries and the manner in which layout is done is not what it is now. In my first iteration of the GameFrame class, my layout was sloppy and subject to unpredictable behavior when resizing occurred. Having now learned proper techniques for making such layout designs, I was able to rewrite a much tighter, predictable GUI window.

One of the best parts of this project was my fellow design team members and the manner in which we worked together. Based on the experiences of others, it seems that having good team chemistry is as important as any other factor. Our team excelled in work delegation, communication, meeting self-imposed deadlines, and work collaboration. This was a major part of our success, more so than I thought it would be.

7.1.2 Bill
When first developing the testing engine, I was only comparing standard out with the ideal test output. However, some parts of ANTLR output things to standard error (like parse error messages). I had to change the testing engine to grab both standard out and standard error and compare that to the test output (which was generated by grabbing both standard out and standard error).

To report the line number of the error during a tree walk, we had to create a custom AST class to store the line number every time a new token was initialized. The custom AST class also had a getLineNumber() method which was called during the tree walk to return that line number. Also, the way we did the walker could have been done easier if we had assumed the parser did its job for us. Also, it's much easier to manage the symbol table if you use a custom symbol table class. We tried to reuse our application classes and it was overly complicated.

We found in some cases, such as “scene class” that the translator needs to mangle
variable names, otherwise it could clash with the output language's keywords. Therefore, we had to change the variable names by adding an “_” to all variables.

Finally, A very important note is to write a style guide before everyone starts coding

7.1.3 Mariya
My advice is to divide up the parts of the compiler early. Look ahead in the lectures or the reading to understand what the specific components of the compiler are and give each person something to work on immediately. I think we tried to do one part at a time, so some people ended up having nothing to do while others were trying to get their part done, and then the workload would switch. I don’t think this is the most efficient way to do the project. Also, in the beginning plan out the language, style, and structure specifics before writing lots of code. This will save a lot of time in the end.

7.1.4 Becky
I feel that the hardest part of this project was understanding each component and what it did at the beginning of the course. As a group we spent a lot of time trying to figure out what each piece was and its function in the compiler. I found that we were trying to build a part of the compiler and we did not understand what we were doing, then we would go to class with the intention of asking Professor Edwards, and then the topic of the lesson for that day would be the part of the compiler we were trying to implement. Thus, one lesson I learned was that the lectures are a little behind where the project progression should be.

Secondly, we were a great group, but I think that if we had understood each section better we could have split things up a little earlier and we might have been able to do some more complicated things with our language. We ended up learning a lot together and therefore we did each part one after another. So, there was one or two people coding with others watching and learning. Whereas we could have had everyone working on different aspects while meeting as a group.

There are a few interesting features in working with runtime environments expecting a user. For example, it is impossible to know which scene a user is in at compile time when checking if the exits being used are valid. Or another case is items, it is impossible to know at compile time what items a user has. Thus when the programmer specifies for a user to pickup an item we don’t know at compile time if the scene actually has the item or not.

7.2 Group Advice
1. Choose group members that you can work well with, not necessarily your friends. We found our working environment functional as well as pleasurable.

2. Read ahead in the slides/book/past projects (especially the Architecture section) to find out what a compiler is made of so that you understand each part and can split up the project more effectively.
3. Create a style guide as well as set up CVS early as you will need both.

4. Start as early as possible. Do not wait for in class deadlines, set your own.
Appendix

A1. Makefile

.DEFAULT:

# makes everything that we have so far (default, just type "make")
all: grammars main extra

% : %.gmm
   java antlr.Tool grimm.g
   java antlr.Tool GRIMMTranslator.g
   javac GRIMM.java
   javac Character.java Scene.java GRIMM.java SymbolTable.java
   GRIMM Lexer.java Thing.java User.java GRIMM Parser.java GRIMM Token Types.java
   gameFrame.java GRIMM Walker.java GRIMM Translator.java
   GRIMM Translator Token Types.java outputFrame.java Item.java CustomAST.java
   java GRIMM $*.gmm
   javac $*.java
   java $*

main: GRIMM.java
   javac GRIMM.java

grammars: grimm.g GRIMMTranslator.g
   java antlr.Tool grimm.g
   java antlr.Tool GRIMMTranslator.g

extra: Character.java Scene.java GRIMM.java SymbolTable.java GRIMM Lexer.java
   User.java GRIMM Parser.java GRIMM Token Types.java gameFrame.java
   GRIMM Walker.java GRIMM Translator.java GRIMM Translator Token Types.java
   outputFrame.java Item.java CustomAST.java
   javac Character.java Scene.java GRIMM.java SymbolTable.java
   GRIMM Lexer.java Thing.java User.java GRIMM Parser.java GRIMM Token Types.java
   gameFrame.java GRIMM Walker.java GRIMM Translator.java
   GRIMM Translator Token Types.java outputFrame.java Item.java CustomAST.java

test:
   ./grimmtest.py testscripts/lexer/* testscripts/parser/*
   testscripts/walker/* testscripts/translator/* testscripts/app/*

clean: testclean
   rm -f *~ *.class GRIMM Lexer.* GRIMM Parser.* GRIMM Translator*
   *Token Types* GRIMM Walker.java

testclean:
   rm -f testscripts/lexer/*.java testscripts/lexer/*~
   testscripts/parser/*.java testscripts/parser/*~ testscripts/walker/*.java
   testscripts/walker/*~

A2. GRIMM.java

/*
 * File: GRIMM.java
 *
 * Authors: Mike Lenner, Billy Liu, Mariya Nomanbhoy, Becky Plummer
 */
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

/*
 * This class runs the lexer, parser, walker, and translator on
 * a .gmm file, outputting any compiler errors to the console.
 */
class GRIMM {
    public static void main(String[] args) {
        if (args.length > 0) {
            try {
                /* get input stream to a file */
                File fin = new File(args[0]);
                FileReader in = new FileReader(fin);

                /* create lexer and parser objects */
                GRIMMLexer lexer = new GRIMMLexer(in);
                GRIMMParser parser = new GRIMMParser(lexer);
                GRIMMWalker walker = new GRIMMWalker();

                /* run lexer and parser on the input file */
                parser.program();

                if (lexer.num_errors > 0) {
                    System.out.println("Scanner Errors!");
                    return;
                }

                if (parser.num_errors > 0) {
                    System.out.println("Parser Errors!");
                    return;
                }

                /* Get the AST from the parser */
                CommonAST parseTree = (CommonAST) parser.getAST();

                /* Open a window in which the AST is */
                /* displayed graphically */
                // ASTFrame frame =
                // new ASTFrame("AST from the Simp parser",
                // parseTree);

                walker.program(parseTree);

                if (walker.gRet < 0) {
                    System.out.println("Walker Errors!");
                    return;
                }

                // frame.setVisible(true);

                GRIMMtranslator gt =
            }
        }
    }
}
new GRIMMtranslator(args[0]);
gt.program( parseTree );
} catch (IOException e) {
    System.out.println(
        "Error occurred while reading your broom file.");
    e.printStackTrace();
    System.out.println("exception: "+e);
} catch (antlr.RecognitionException re) {
    System.out.println(
        "Recognition Exception occurred while reading" + " your grimm file.");
    re.printStackTrace();
    System.out.println("exception: "+re);
} catch (antlr.TokenStreamException re) {
    System.out.println("Token Stream Exception" + " occurred while reading your grimm file");
    re.printStackTrace();
    System.out.println("exception: "+re);
}
} else {
    System.out.println("ERROR 6111: No program name found" + " at the command prompt");
} return;
}
protected ALPHA : 'a'..'z' | 'A'..'Z' | '_';
protected DIGIT : '0'..'9';

WS : ( (' ' | '	') { $setType(Token.SKIP); } )+;
STRCONST : '"'! ( ~('"' | '
') | ('"'! '"') )* '"'!;
NEWLINE : (( '\n' | '\r' '\n' ){ newline(); } )+;
COMMENT : ( ":note:" ( ~( '
' | '\r' ) )* NEWLINE )
          { $setType(Token.SKIP); };

NOUN options { testLiterals = true; }
          : ALPHA (ALPHA | DIGIT)*;

//
****************************************************************************
class GRIMMParser extends Parser;
//
/* keywords: goto, say, says, user, pickup, has, while, if, then, endif,
  * otherwise, otherwiseif, and, or, endwhile, inside, scene, exit, picture,
  * description, name, item, character, hidden, read, input, gameover,
  * contains
  */

options {
  k = 2;
  buildAST = true;
  exportVocab = GRIMM;
}

tokens{
  STMTS;
  DECLS;
  PROGRAM;
  BOOLEXPR;
}
{
  int num_errors = 0;
  public void reportError( String s) {
    super.reportError(s);
    num_errors++;
  }

  public void reportError(RecognitionException e) {
    super.reportError(e);
    num_errors++;
  }
}
/* declarations */
scene_decl: 
    "scene"^ NOUN (NEWLINE!)+;

char_decl:
"character"^ NOUN (NEWLINE!)+;

/* assignment */
char_item_assign:
   NOUN "holds"^ "item" NOUN (NEWLINE!)+;

item_assign:
   NOUN "contains"^ "item" NOUN ("is"! "hidden")? (NEWLINE!)+;

name_assign:
   NOUN "name"^ STRCONST (NEWLINE!)+;

desc_assign:
   NOUN "description"^ STRCONST (NEWLINE!)+;

picture_assign:
   NOUN "picture"^ STRCONST (NEWLINE!)+;

exit_assign:
   NOUN "exit"^ NOUN ("is"! "hidden")? (NEWLINE!)+;

/* user boolean expressions */
user_says:
   "user"! "says"^ STRCONST;

person_has:
   ("user" | NOUN ) "has"^ NOUN;

user_inside:
   "user"! "inside"^ NOUN;

/* control structures */
conditional:
   ("not")? ( user_says | person_has | user_inside );

bool_expr:
   and_expr ( "or"^ and_expr )*  
   { #bool_expr = # ( [BOOLEXPR, "bool_expr"], bool_expr ); } ;

and_expr:
   conditional ( "and"^ conditional )*;

if_stmt:
   "if"^ bool_expr "then"! (NEWLINE!)+ stmts  
   (owiseif)*  
   (owise)?  
   "endif"!;

owiseif:
   "otherwiseif"^ bool_expr "then"! (NEWLINE!)+ stmts ;

owise:
   "otherwise"^ (NEWLINE!)+ stmts ;
while_stmt:
  "while"^ bool_expr (NEWLINE!)+ stmts "endwhile"!;

/* actions */
action:
  jump
  | say
  | user_input
  | game_over
  | pick_up
  | drop
;

jump:
  "goto"^ NOUN;

say:
  "say"^ STRCONST;

user_input:
  "read" "user"! "input"!;

pick_up:
  ("user" | NOUN ) "pickup"^ NOUN;

drop:
  ("user" | NOUN ) "drops"^ NOUN;

game_over:
  "gameover";

/* putting it together */
stmt:
  if_stmt
  | while_stmt
  | action;

stmts:
  (stmt (NEWLINE!)+)
  {
    #stmts = #([STMTS, "stmts"], stmts);
  }
  ;

decls:
  {
    scene_decl
    | char_decl
    | char_item_assign
    | item_assign
    | name_assign
    | desc_assign
    | picture_assign
    | exit_assign
  }*
  {
    #decls = #([DECLS, "decls"], decls);
  }
  ;
program:
/* add custom AST node to make line numbers available in walker */
{ getASTFactory().setASTNodeType("CustomAST"); }

decls
stmts
EOF!

/*program = #([PROGRAM, "program"], program ); */

/* globals */
SymbolTable gSym_Tab = new SymbolTable();
int gRet = 0;

/* print error function */
public void printError(String msg, int line) {
    System.out.println("Semantic Error on "+line+: "+msg);
}

program:
#(PROGRAM declaration statement)
{
    if (!gSym_Tab.checkIntegrity()) {
        gRet = -1;
    }
}

declaration:
#(DECLS ( (scene_decl  |
exit_assign |
char_decl   |
char_item_assign |
name_assign |
desc_assign |
item_assign |
picture_assign)* ) } );

statement:
#(STMTS (action | if_stmt | while_stmt)+ );

while_stmt:
#("While" bool_exp statement);

if_stmt:
other_expr:
  #("otherwiseif"
   {gSym_Tab.printDebug("In otherwiseif");}
  bool_exp statement
  |#("otherwise"
    {gSym_Tab.printDebug("In otherwise");}
    statement);

bool_exp:
  #(BOOLEXPR log_expr )
  {gSym_Tab.printDebug("bool expr");};

log_expr:
  #("and" log_expr log_expr)
  {gSym_Tab.printDebug("and expr");}
  |#("or" log_expr log_expr)
  {gSym_Tab.printDebug("or expr");}
  |#("not" cond_pred)
  | cond_pred;

cond_pred:
  #("says" STRCONST)
  {gSym_Tab.printDebug("says expr");}
  | #("inside" scene_name:NOUN)
    {gSym_Tab.printDebug("inside expr");
     if (!gSym_Tab.isDeclared(scene_name.getText(),SymbolTable.SCENE))
       {
         printError (scene_name.getText()+" not declared"+
                     " or not of type scene",
                     scene_name.getLineNumber());
         gRet = -1;
       }
    }
  | #("has" has_expr);

has_expr:
  ("user" item_user:NOUN)
  {gSym_Tab/printDebug("User has");
   if (!gSym_Tab.isDeclared(item_user.getText(),SymbolTable.ITEM))
     {
       printError (item_user.getText()+ " not declared"+
                   " or not of type item",
                   item_user.getLineNumber());
       gRet = -1;
     }
  }
  | (char_name:NOUN item_char:NOUN)
    {gSym_Tab/printDebug("Char has");
     if (!gSym_Tab.isDeclared(item_char.getText(),SymbolTable.ITEM))
       {
         printError (item_char.getText()+ " not declared"+
                     " or not is of type item",
                     item_char.getLineNumber());
       }
    }
gRet = -1;
}
if (!gSym_Tab.isDeclared(char_name.getText(), SymbolTable.CHAR))
{
perror (char_name.getText() + " not declared" +
   " or is not of type character",
   char_name.getLineNumber());
gRet = -1;
}

action:
   #("goto" scene_name:NOUN)
   {
      gSym_Tab.printDebug ("goto");
      if (!gSym_Tab.isDeclared(scene_name.getText(), SymbolTable.SCENE))
      {
         perror (scene_name.getText() + " not declared" +
                  " or not of type scene",
                  scene_name.getLineNumber());
         gRet = -1;
      }
   }
   | #("pickup" pickup_stmnt)
   | #("drops" drop_stmnt)
   | ("gameover")
   | ("read")
   | ("say")
      {gSym_Tab.printDebug ("Game Over");}

pickup_stmnt:
   ("user" item_user:NOUN)
   {
      gSym_Tab.printDebug ("user pickup");
      if (!gSym_Tab.isDeclared(item_user.getText(), SymbolTable.ITEM))
      {
         perror (item_user.getText() + " not declared or" +
                  " not of type item",
                  item_user.getLineNumber());
         gRet = -1;
      }
   }
   | (char_name:NOUN item_name:NOUN)
   {
      gSym_Tab.printDebug ("char pickup");
      if (!gSym_Tab.isDeclared(item_name.getText(), SymbolTable.ITEM))
      {
         perror (item_name.getText() + " not declared or" +
                  " not of type item",
                  item_name.getLineNumber());
         gRet = -1;
      }
   }
   if (!gSym_Tab.isDeclared(char_name.getText(), SymbolTable.CHAR))
      {perror (char_name.getText() + " not declared or " +
             " not of type character",}
char_name.getLineNumber());
    gRet = -1;
} 
};
drop_stmnt:
  ("user" item_user:NOUN)
  
  
  
  
  gSym_Tab.printDebug ("user drops");
  if (!gSym_Tab.isDeclared(item_user.getText(),SymbolTable.ITEM))
  {
      printError (item_user.getText()+ " not declared or"+
                  " not of type item",
               item_user.getLineNumber());
      gRet = -1;
  }
  
  | (char_name:NOUN item_name:NOUN)
  
  
  
  
  gSym_Tab.printDebug ("char drop");
  if (!gSym_Tab.isDeclared(item_name.getText(),SymbolTable.ITEM)) {
      printError (item_name.getText()+ " not declared or"+
                  " not of type item",
               item_name.getLineNumber());
      gRet = -1;
  }
  if (!gSym_Tab.isDeclared(char_name.getText(),SymbolTable.CHAR)) {
      printError (char_name.getText()+ " not declared or "+
                  " declared of type other than character",
              char_name.getLineNumber());
      gRet = -1;
  }
};
say:
  #("say" s:STRCONST) {gSym_Tab.printDebug(s.getText());};
scene_decl:
  #("scene" n:NOUN)
  
  
  
  
  int temp;
  gSym_Tab.printDebug("Scene Decl");
  temp = gSym_Tab.addSymbol(n.getText(),SymbolTable.SCENE);
  if (temp == -1) {
      printError("Identifier "+n.getText()+" already used",
              n.getLineNumber());
  }
  gRet += temp;
};
char_decl:
  #("character" n:NOUN)
  
  
  
  
  int temp;
  temp = gSym_Tab.addSymbol(n.getText(),SymbolTable.CHAR);
  if (temp == -1) {
printError("Identifier "+n.getText()+" already used", n.getLineNumber());

gRet += temp;
}
}
}

exit_assign:
"exit" scene_name:NOUN exit_name:NOUN ("hidden")?
{
if (!gSym_Tab.isDeclared(exit_name.getText(),SymbolTable.SCENE))
{
printError(exit_name.getText()+" not declared"+
" or is not of type scene.", exit_name.getLineNumber());
gRet = -1;
}
if (!gSym_Tab.isDeclared(scene_name.getText(),SymbolTable.SCENE))
{
printError(scene_name.getText()+" not declared or is not of type scene", scene_name.getLineNumber());
gRet = -1;
}
if (!gSym_Tab.addExit(scene_name.getText(),
exit_name.getText()))
{
printError(scene_name.getText()+" already has "+
exit_name.getText()+" as an exit", scene_name.getLineNumber());
gRet = -1;
}
}

name_assign:
"name" scene_name:NOUN name:STRCONST)
{
if (!gSym_Tab.isDeclared(scene_name.getText(),SymbolTable.SCENE))
{
printError ("Cannot assign name "+name.getText()+
" to scene "+scene_name.getText()+
" because "+scene_name.getText()+" is not"+
" declared or is not of type scene.",
scene_name.getLineNumber());
gRet = -1;
}
}

gSym_Tab.nameDefined(scene_name.getText());

picture_assign:
"picture" scene_name:NOUN STRCONST)
{
if (!gSym_Tab.isDeclared(scene_name.getText(),SymbolTable.SCENE))
{
printError (scene_name.getText()+
" not declared or is not of type scene. ",

desc_assign:
  #("description" scene_name:NOUN STRCONST)
  {
      if (!gSym_Tab.isDeclared(scene_name.getText(), SymbolTable.SCENE))
      {
          printError (scene_name.getText() +
          " not declared or is not of type scene.",
          scene_name.getLineNumber());
          gRet = -1;
      };
  }

item_assign:
  #("contains" scene_name:NOUN "item" item_name:NOUN)
  {
      if (!gSym_Tab.isDeclared(scene_name.getText(),
          SymbolTable.SCENE)) {
          printError(scene_name.getText() +
          " not declared or is not of type scene",
          scene_name.getLineNumber());
          gRet = -1;
       } else {
          int temp;
          temp = gSym_Tab.addSymbol(item_name.getText(),
              SymbolTable.ITEM);
          if (temp == -1) {
              printError(item_name.getText() +
              " already declared",
              item_name.getLineNumber());
          }
          gRet += temp;
      }
  }

char_item_assign:
  #("holds" char_name:NOUN "item" item_name:NOUN)
  {
      if (!gSym_Tab.isDeclared(char_name.getText(), SymbolTable.CHAR)) {
          printError (char_name.getText() +
          " not declared or is not of type character",
          char_name.getLineNumber());
          gRet = -1;
      } else {
          int temp;
          temp = gSym_Tab.addSymbol(item_name.getText(),
              SymbolTable.ITEM);
          if (temp == -1) {
              printError(item_name.getText() +
              " already declared",
          }
A4. SymbolTable.java

/*
 * File: SymbolTable.java
 * Authors: Mike Lenner, William Liu
 */
import java.util.*;

/*
 * Symbol Table class stores symbols for the walker
 */
public class SymbolTable {
    /* Flag to print debug messages to console */
    public static final boolean DEBUG = false;

    /* Vector of symbols for our symbol table */
    Vector symbols = new Vector();

    /* Types of objects in the symbol table */
    public static final String SCENE = "scene";
    public static final String CHAR = "character";
    public static final String ITEM = "item";

    /* Constructor */
    public SymbolTable() {
    }

    /* Adds a symbol of type to the symbol table using name as the id.
     * Returns 0 on success, -1 on error.
     */
    public int addSymbol(String name, String type) {
        Symbol newSym = new Symbol();
        newSym.Id = name;
        newSym.Type = type;

        if (isDeclared(name)) {
            printDebug("Scene \"+name\" already declared");
            return -1;
        }
        symbols.add(newSym);

        return 0;
    }

    /* Tests whether a certain symbol is already in the symbol table.
     * Returns true if it exists, false if it does not.
     */
public boolean isDeclared(String name) {
    for (int i=0; i<symbols.size(); i++) {
        Symbol temp = (Symbol) symbols.get(i);
        if (temp.Id.equals(name))
            return true;
    }
    return false;
}

public boolean isDeclared(String name, String type) {
    for (int i=0; i<symbols.size(); i++) {
        Symbol temp = (Symbol) symbols.get(i);
        if (temp.Id.equals(name) && temp.Type.equals(type))
            return true;
    }
    return false;
}

public boolean nameDefined(String name) {
    for (int i=0; i<symbols.size(); i++) {
        Symbol temp = (Symbol) symbols.get(i);
        if (temp.Id.equals(name)) {
            temp.Name_Defined = true;
            return true;
        }
    }
    return false;
}

public boolean checkIntegrity() {
    for (int i=0; i<symbols.size(); i++) {
        Symbol temp = (Symbol) symbols.get(i);
        if ((temp.Type == SCENE) && (!temp.Name_Defined)) {
            System.out.println("Semantic Error: Scene "+ temp.Id+" doesn't have a name defined");
            return false;
        }
    }
    return true;
}
/*
 * This method is used to add an exit to a scene symbol. It is used
 * when an exit is assigned to a scene. It allows to check later to
 * verify a scene has not been added as exit twice to the same scene
 */
public boolean addExit(String scene, String exit) {
    for (int i=0; i<symbols.size(); i++) {
        Symbol temp = (Symbol) symbols.get(i);
        if (temp.Id.equals(scene)) {
            if (temp.Exits.contains(exit))
                return false;
            else {
                temp.Exits.addElement(exit);
                return true;
            }
        }
    }
    System.out.println("Semantic Error: scene"+
                        " has not been declared and was not"+
                        " caught in isDeclared(). Something"+
                        " is very wrong.");
    return false;
}

/*
 * Outputs a message to the console if DEBUG is true.
 */
public void printDebug(String msg) {
    if (DEBUG)
        System.out.println(msg);
}

/*
 * Symbol class for each id
 */
class Symbol {
    public String Type;  /* symbol type (SCENE, CHAR, ITEM) */
    public String Id;  /* unique id for symbol */
    public boolean Name_Defined; /* flag for SCENE types */
    public Vector Exits;

    /*
    * Constructor
    */
    public Symbol() {
        Type = "";
        Id = "";
        Name_Defined = false;

        /* initialize exit vector */
        Exits = new Vector();
    }
A5. CustomAST.java

/*
 * File: CustomAST.java
 *
 * Author(s): Mike Lenner, Billy Liu
 *
 */

import antlr.*;
import antlr.collections.AST;

/*
 * AST class that will store the line number so our walker can report line
 * numbers when a semantic error occurs
 */
public class CustomAST extends CommonAST {
    private int Line_Number;

    /*
     * Constructor
     */
    public CustomAST() {}

    /*
     * Constructor that initializes with token
     */
    public CustomAST(Token tok) { super(tok); }

    /*
     * Initializes the AST
     */
    public void initialize(AST t) {
        super.initialize(t);
        if (t.getClass().getName().compareTo("CustomAST") == 0) {
            CustomAST t2 = (CustomAST)t;
            Line_Number = t2.Line_Number;
        }
    }

    /*
     * Initializes a token
     */
    public void initialize(Token tok) {
        super.initialize(tok);
        /* store the line number from the token */
        Line_Number = tok.getLine();
    }

    /*
     * Used in tree walker to get the token's line number
     * Returns the line number.
     */
/*
   public int getLineNumber() {
       return Line_Number;
   }
}

A6. GRIMMTranslator.g
/*
 * File: GRIMMTranslator.g
 *
 * Author: Mariya Nomanbhooy
 */

{import java.io.*;}

/* This is the translator class. A translator object is made in the Main
 * file. It will then translate grimm file into proper java code using
 * the AST. The constructor takes a String parameter which is the name
 * of the java file to be created.
 */
class GRIMMtranslator extends TreeParser;

options {
    importVocab = GRIMM;
    //buildAST = true;
}

{ /* Class member variables */
    public String Class_Name;
    public String Java_Code;
    int Tabs = 0;

    /* Constructor takes the name of output file as a String and
     * creates a new translator object.
    */
    public GRIMMtranslator( String filename ) {
        super();
        Class_Name = filename;
    }

    /* This function prints the tabs based on how the current indentation
     * of the file.
    */
    public void printTabs() {
        for( int j = 0; j < Tabs; j++ ) {
            Java_Code += "\t";
        }
    }
}

/* This method does the translation of the grimm file into a java
* file. It takes a parse tree as a parameter and returns void.
* The method will print out the java code to Class_Name.java
*/
program {
    Java_Code = "//This is your grimm file translated into java\n";
    Tabs = 0;
}
#endif

/*
* Writing out header to string value.
* This includes initializing a game frame and user
*/
PROGRAM {
    /* creating a java class */
    Java_Code += "public class " + Class_Name.substring(0, Class_Name.length() - 4) + " {";
    printTabs();
    Tabs++;
    printTabs();
    /* starting main function */
    Java_Code += "public static void main( String args[]
";
    Java_Code += " )\n\t{\n"
    Tabs++;
    printTabs();
    /* initializing user */
    printTabs();
    Java_Code += "User player = new User();\n";
    printTabs();
    Java_Code += "Scene curScene = new Scene(null, null, null);\n";
    printTabs();
    /* initializing graphics */
    printTabs();
    Java_Code += "//initializing game frame"
    Java_Code += "\n\tgameFrame window = new gameFrame();\n";
    /*
    The program is divided up into two main parts.
The first part has all declarations and assignments. There can be 0 or more of these
    */
    declarations
    /*
    The second part of the code is the statements which there can be one or more off
    */
statements {
    /* closing main and class */
    Java_Code += "\t}\n};
    try{
        BufferedWriter out = new BufferedWriter(
            new FileWriter(
                Class_Name.substring(0,Class_Name.length() -3)
                + "java"));
        out.write(Java_Code);
        out.close();
    } catch(IOException e) {
        System.out.println(e);
    }

    */
    * This will translate all declarations into java.
    */
    declarations:
    /* first we check what kind of a declaration we have */
    #(dec:DECLS
        (sceneDecl  |
        exitAssign  |  
        charDecl  |  
        charItemAssign  |  
        nameAssign  |  
        descAssign  |  
        itemAssign  |  
        pictureAssign  ))
    );

    */
    * Printing a scene declaration. *
    */
    sceneDecl:
        #("scene" n:NOUN ) {
            printTabs();
            /*printing new scene */
            Java_Code += "Scene _" + n.getText()
            + " = new Scene(null, null, null);\n";
        }

    */
    * Printing a character declaration. *
    */
    charDecl:
        #("character" n:NOUN ) {
            printTabs();
            /* printing new scene */
            Java_Code += ("Character _" + n.getText()
            + " = new Character(null);\n"");
        }

    */
    * Assigning and exit to a scene.
exitAssign:
  #(dec:"exit" scene_name:NOUN exit_name:NOUN ("hidden")? ) {
    printTabs();
    if( dec.getNumberOfChildren() == 3 ) {
      /* adding hidden exit */
      Java_Code += "_." + scene_name.getText() + ".addExit( _." + exit_name.getText() + ", true );\n";
    }
    else {
      /* adding visible exit */
      Java_Code += "_." + scene_name.getText() + ".addExit( _." + exit_name.getText() + ", false);\n";
    }
  }

nameAssign:
  #("name" var:NOUN name:STRCONST ) {
    printTabs();
    Java_Code += "_." + var.getText() + ".setName( " + name.getText() + ");\n";
  }

pictureAssign:
  #("picture" scene_name:NOUN name:STRCONST) {
    printTabs();
    Java_Code += "_." + scene_name.getText() + ".setPicture( " + name.getText() + ");\n";
  }

descAssign:
  #("description" scene_name:NOUN descr:STRCONST) {
    printTabs();
    Java_Code += "_." + scene_name.getText() + ".setDescr( " + descr.getText() + ");\n";
  }

itemAssign:
  #(dec:"contains" scene_name:NOUN "item" item_name:NOUN) {
    printTabs();
    Java_Code += "Item _" + item_name.getText() + "+ _ = new Item( " + item_name.getText() + ";\n";
    printTabs();
if( dec.getNumberOfChildren() == 4 ) {
    Java_Code += "_" + scene_name.getText() + ".addItem( _" + item_name.getText() + ", true );\n";
} else {
    Java_Code += "_" + scene_name.getText() + ".addItem( _" + item_name.getText() + ", false );\n";
}

/*
 * Assigning an item to a character
 */
charItemAssign:
    #("holds" char_name:NOUN "item" item_name:NOUN)
    {
        printTabs();
        Java_Code += "Item _" + item_name.getText() + " = new Item( " + item_name.getText() + ");\n";
        printTabs();
        Java_Code += "_" + char_name.getText() + ".pickUp( _" + item_name.getText() + ");\n";
    }
/*
 * Going through all statements and calling appropriate function.
 */
statements:
    #(STMTS (action | ifStmt | whileStmt)+ )
/*
 * We have found an action and need to act appropriately
 */
action:
    #("goto" scene_name:NOUN)
    {
        printTabs();
        Java_Code += "player.moveTo( _" + scene_name.getText() + ");\n";
        printTabs();
        Java_Code += "curScene = _" + scene_name.getText() + ";\n";
        printTabs();
        Java_Code += "window.showScene( player.getCurScene(), player);\n";
    }
| #("pickup" pickupStmt)
| #("drops" dropStmt)
| ("gameover")
    {
        printTabs();
        Java_Code += "window.setOutput(" GAMEOVER");\n";
        printTabs();
        Java_Code += "window.showScene( player.getCurScene(), player);\n";
        Java_Code += "window.showScene( player.getCurScene(), player);\n";
        Java_Code += "while(true) {;\n";
    }
| ("read")
    {
        printTabs();
        Java_Code += "player.setSaid( window.getInput()) ;\n";
    }

boolExp {
    Java_Code += " )\n";
    printTabs();
    Java_Code += "{\n"
    Tabs++;
}

statements {
    Tabs--;
    printTabs();
    Java_Code += "}\n";
}));

/*
* Found an if statement
*/
ifStmt:
    #("if" {
        printTabs();
        Java_Code += "if( ";
    }

boolExp {
    Java_Code += " )\n";
    printTabs();
    Java_Code += "{\n"
    Tabs++;
}

statements {
    /* ending previous if statement*/
    Tabs--;
    printTabs();
    Java_Code += "}\n";
}

(otherExpr)*
}

/*
* Found an otherwise if or otherwise statement
*/
otherExpr:
    #("otherwiseif" {
        printTabs();
        Java_Code += "else if( ";
    }

boolExp {
    Java_Code += " )\n";
    printTabs();
    Java_Code += "{\n"
    Tabs++;
}

statements {
    Tabs--;
    printTabs();
A7. gameFrame.java

/*

*/
boolExp:
   #(BOOLEXP logExpr);

logExpr:
   #("and" logExpr {
      Java_Code += " && ";
   } logExpr)
   | #("or" logExpr {
      Java_Code += " || ";
   } logExpr)
   | #("not" {
      Java_Code += " !";
   } condPred)
   | condPred;

condPred:
   #("says" words:STRCONST) {
      Java_Code += "(player.getSaid().equals("" + words.getText() + "") ");
   }
   | #("inside" scene_name:NOUN) {
      Java_Code += "(player.getCurScene() == _" + scene_name.getText() + ")";
   }
   | #("has" hasExpr);

hasExpr:
   (person:"user" item_user:NOUN) {
      Java_Code += "player.hasItem( _" + item_user.getText() + " ) ";
   }
   | (char_name:NOUN item_char:NOUN) {
      Java_Code += "_.hasItem( _" + char_name.getText() + ".hasItem( _" + item_char.getText() + " ) "");
   }
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.table.AbstractTableModel;
import java.awt.*
import java.util.Vector;
import java.io.*;

/* The GameFrame class is responsible for displaying the console on which the
 * GRIMM story will be displayed and played. */

public class GameFrame extends JFrame {

    /* components */
    static private JTextArea Terminal;
    private JLabel Title;
    private JTextArea Desc;
    private JTextField Input;
    private JTextArea Exits;
    private JLabel Image;
    private JSplitPane Item_Pane;
    private JTable Scene_Items;
    private JTable User_Items;
    private JScrollPane Term_Scroll;
    private JScrollPane Exit_Scroll;

    private static final String NONE = "noimg";

    /* private members of GameFrame */
    private boolean Cmd_Entered = false;
    private String Cmd_Text;

    /* Constructor for GameFrame class */
    /* Return Value: none */
    public GameFrame() {
        initComponents();
        setVisible(true);
    }

    /* This function is responsible for building the various components of
    * the frame and setting their layout correctly. This function also
    * assigns all required listeners. */
    /* Return Value: none */
    private void initComponents() {
        /* build default objects for top panel */
        Title = new JLabel();
    }
}
Desc = new JTextArea();
Exits = new JTextArea();
Image = new JLabel();
Terminal = new JTextArea();
Input = new JTextField();

/* set all preferred sizes */
Title.setPreferredSize(new Dimension(700,40));
Desc.setPreferredSize(new Dimension(700,100));
//Exits.setPreferredSize(new Dimension(700,45));
Image.setPreferredSize(new Dimension(700,425));
Input.setPreferredSize(new Dimension(575,20));

/* build default image */
setImg(NONE);

/* create borders */
Border empty_5 = BorderFactory.createEmptyBorder(5,5,5,5);
Border line_black1 =
    BorderFactory.createLineBorder(Color.black, 1);
Border line_black2 =
    BorderFactory.createLineBorder(Color.black, 2);
Border compund1 =
    BorderFactory.createCompoundBorder(line_black1, empty_5);

/* add borders */
Image.setBorder(line_black2);
Desc.setBorder(compund1);
Exits.setBorder(empty_5);

/* set fonts */
Title.setFont(new Font("",Font.BOLD, 24));
Desc.setFont(new Font("", Font.BOLD, 12));
Exits.setFont(new Font("", Font.BOLD, 12));

/* set properties of text areas */
Desc.setEditable(false);
Desc.setLineWrap(true);
Desc.setWrapStyleWord(true);
Exits.setEditable(false);
Title.setHorizontalAlignment(JLabel.CENTER);
Image.setHorizontalAlignment(JLabel.CENTER);

/* build two tables for user and room items with default */
/* items */
User_Items = new JTable(new CustomTabMod("User Items"));
Scene_Items = new JTable(new CustomTabMod("Scene Items"));

/* create scoll bar panes for two item tables */
JSplitPane user_scroll = new JScrollPane(User_Items);
JSplitPane scene_scroll = new JScrollPane(Scene_Items);

/* split pane for item lists */
Item_Pane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
    user_scroll, scene_scroll);
Item_Pane.setOneTouchExpandable(true);
Item_Pane.setPreferredSize(new Dimension(125,220));
Item_Pane.setDividerLocation(90);

/* set properties of terminal text area and input field */
Terminal.setEditable(false);
Terminal.setLineWrap(true);
Terminal.setWrapStyleWord(true);
Input.setEditable(false);
Input.setBorder(line_black1);
Input.setFocusable(true);
Terminal.setFocusable(false);

/* add terminal into scroll bar pane */
Term_Scroll = new JScrollPane(Terminal,
   JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
   JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
Term_Scroll.setPreferredSize(new Dimension(575,200));

/* add exit into scroll bar pane */
Exit_Scroll = new JScrollPane(Exits,
   JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
   JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
Exit_Scroll.setPreferredSize(new Dimension(700,45));

/* determine layout */
getContentPane().setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
gbc.insets = new Insets(5,20,5,20);
gbc.ipadx = 0;
gbc.ipady = 0;
gbc.gridx = 0;
gbc.gridy = 0;
gbc.weighty = 0;
gbc.weightx = 1;
gbc.gridwidth = 1;
gbc.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(Title,gbc);

getContentPane().add(Image,gbc);

getContentPane().add(Desc,gbc);
gbc.weighty = 0.5;
gbc.weightx = 1;
gbc.gridwidth = 3;
gbc.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(Exit_Scroll, gbc);

gbc.insets = new Insets(5,20,0,5);
gbc.gridx = 0;
gbc.gridy = 4;
gbc.weighty = 0.5;
gbc.weightx = 0;
gbc.gridwidth = 2;
gbc.fill = GridBagConstraints.BOTH;
getContentPane().add(Term_Scroll, gbc);

gbc.insets = new Insets(5,5,5,20);
gbc.gridx = 2;
gbc.gridy = 4;
gbc.weighty = 0.5;
gbc.weightx = 1;
gbc.gridwidth = 1;
gbc.gridheight = 2;
gbc.fill = GridBagConstraints.BOTH;
getContentPane().add(Item_Pane, gbc);

gbc.insets = new Insets(0,20,5,5);
gbc.gridx = 0;
gbc.gridy = 5;
gbc.weighty = 0;
gbc.weightx = 0;
gbc.gridwidth = 2;
gbc.gridheight = 1;
gbc.fill = GridBagConstraints.BOTH;
getContentPane().add(Input, gbc);

/* Set location of frame */
setLocation(100,10);
pack();

/*
 * set up listeners
 */

/* set up exit event */
addWindowListener(new java.awt.event.WindowAdapter() {
    public void
        windowClosing(java.awt.event.WindowEvent evt) {
            exitForm(evt);
        }
});

/* set up listener for terminal input from user */
Input.addActionListener(new java.awt.event.ActionListener() {
    public void
        actionPerformed(java.awt.event.ActionEvent evt) {
            /* record cmd txt */
            Cmd_Text = evt.getActionCommand();
        }
});
/* record cmd avail */     Cmd_Entered = true;
            }  
        });

/*
 * This is a wrapper function to allow graceful closing of the frame.
 * Return Value: none
 */
private void exitForm(java.awt.event.WindowEvent evt) {
    System.exit(0);
}

/*/  
 * This function assigns the description text.  
 * Return Value: none  
 */
public void setDescription(String text) {
    Desc.setText(text);
}

/*/  
 * This function assigns the title text  
 * Return Value: none  
 */
public void setTitle(String text) {
    Title.setText(text);
}

/*/  
 * This function takes all exits passed in via the parameter and  
 * assigns the text name associated with those exits to the Exits text  
 * area.  
 * Return Value: none  
 */
public void setExits(Vector ext) {
    String list = "";
    for (int i=0; i < ext.size(); i++) {
        Scene temp = (Scene) ext.get(i);
        list += "There is an exit to " + temp.getName() + "\n";
    }
    Exits.setText(list);
}

/*
 * This function takes all items passed in via the parameter and  
 * assigns the text name associated with those items to the User_Items  
 * table.  
 * Return Value: none  
 */
public void setUserItems(Vector items) {
    /* clear all items */
    ((CustomTabMod)User_Items.getModel()).clear();
    for (int i=0; i < items.size(); i++) {
        Item temp = (Item) items.get(i);
        User_Items.setValueAt(temp.getName(), i, 0);
    }
    /* This function takes all items passed in via the parameter and
    * assigns the text name associated with those items to the Scene_Items
    * table.
    * Return Value: none
    */
    public void setSceneItems(Vector items) {
        /* clear all items */
        ((CustomTabMod)Scene_Items.getModel()).clear();
        for (int i=0; i < items.size(); i++) {
            Item temp = (Item) items.get(i);
            Scene_Items.setValueAt(temp.getName(), i, 0);
        }
        /* This function assigns the image passed in via the parameter to the
        * image label. If the path cannot be resolved, or a null path is
        * passed, the label will display the text"No Image Found." All paths
        * are assumed to be relative.
        * Return Value: none
        */
        public void setImg(String name) {
            if (name == null) {
                Image.setText("Image Not Found");
                return;
            }
            java.net.URL imgURL = gameFrame.class.getResource(name);
            if (imgURL != null) {
                ImageIcon newImg = new ImageIcon(imgURL);
                Image.setText(null);
                Image.setIcon(newImg);
            } else {
                Image.setText("Image Not Found");
            }
        }
        /* This function is the main interface for the GameFrame class. A
        * Scene and a User object are passed in via parameter and all relevant
public void showScene(Scene s, User u) {
    if (s != null) {
        setTitle(s.getName());
        setDescription(s.getDescr());
        setExits(s.getVisibleExits());
        setImg(s.getPicture());
        setSceneItems(s.getVisibleItems());
        setUserItems(u.items());

        /* update GUI */
        repaint();
    }
}

public String getInput() {

    String cmd;

    /* allow user to enter commands */
    Input.setEditable(true);
    Input.setEnabled(true);

    /* request keyboard focus */
    Input.requestFocusInWindow();

    /* keep control of program here until user enters cmd */
    while (!Cmd_Entered);

    /* save command string */
    cmd = Cmd_Text;

    /* output to the terminal */
    setOutput(cmd);

    /* reset for next command */
    Cmd_Entered = false;
    Cmd_Text = "";

    /* clear text field */
Input.setText(null);
Input.setCaretPosition(0);

/* disallow user to enter commands until next call */
Input.setEditable(false);
Input.setEnabled(false);

return cmd;
}

/*
 * This function will output the passed in String to the terminal text
 * area. A newline is placed before and after the output.
 *
 * Return Value: none
 */
public static void setOutput(String text) {
Terminal.append("\n" + text + "\n");
Terminal.setCaretPosition(Terminal.getDocument().getLength());
}

/*
 * This inner class is used to implement custom table model. This
 * allows for the scene items and user items to be easily stored,
 * displayed, and manipulated in the desired fashion.
 */
class CustomTabMod extends AbstractTableModel {

private Vector Data = new Vector();
private String Header;

/*
 * Constructor. Uses the passed in string as the header title
 * of the column
 *
 * Return Value: none
 */
public CustomTabMod(String header) {
    this.Header = header;
}

/*
 * Returns the size of the Data vector, which will translate
 * into the number of rows in our list.
 *
 * Return Value: number of items in Data vector (int)
 */
public int getRowCount() {
    return Data.size();
}

/*
 * Returns the number of columns for our list, which is always
 * one since our table is only one column wide. This function
 * must be implemented as part of the AbstractTableModel
 * abstract class.
 */
public int getColumnCount() {
    return 1;
}

public String getColumnName(int column) {
    return Header;
}

public Object getValueAt(int row, int column) {
    return Data.get(row);
}

public void setValueAt(Object aValue, int rowIndex, int columnIndex) {
    Data.addElement(aValue);
    fireTableDataChanged();
}

public void clear() {
    Data.removeAllElements();
    fireTableDataChanged();
}

public static void main(String[] args) {
    /*Scene s1 = new Scene("Scene #1","Scene 2.1", 800, 600, null);*/

}
"This is scene #1",
"img");*/
//User u = new User();
//s1.addItem(new Item("Key"),false);

gameFrame gf = new gameFrame();
//gf.showScene(s1,u);
while(true)
    gf.getInput();
}
} // end gameFrame class

A8.Thing.java
/*
 * File: Thing.java
 * Author: William Liu
 */

/*
 * Base class for all objects in GRIMM
 */
public class Thing
{
    private String name;  /* name of the object */

    /* Constructor initializes the name */
    public Thing(String n) {
        name = n;
    }

    /* Returns the name */
    public String getName() {
        return name;
    }

    /* Sets the name of the Thing */
    public void setName(String n) {
        name = n;
    }
}
A9. Scene.java

/*
* File: Scene.java
*
* Author: William Liu
*/

import java.util.Vector;

/*
* Scene class for all scenes in GRIMM
*/
public class Scene extends Thing
{
    private String Description;  /* description of scene */
    private String Image;   /* image for the scene */
    private Vector Hidden_Items = new Vector(); /* hidden items */
    private Vector Visible_Items = new Vector(); /* visible items */
    private Vector Hidden_Exits = new Vector(); /* hidden exits */
    private Vector Visible_Exits = new Vector(); /* visible exits */

    /*
     * Constructor takes the name, description, and picture
    */
    public Scene(String n, String descr, String img) {
        super(n);
        Description = descr;
        Image = img;
        return;
    }

    /*
     * Constructor takes just initializes description and image
    */
    public Scene(String n) {
        super(n);
        Description = Image = n;
    }

    /*
     * Returns the description of the scene
    */
    public String getDescr() {
        return Description;
    }

    /*
     * Sets the description for the scene
    */
    public void setDescr(String s) {
        Description = s;
    }

    /*
     * Returns all the hidden items in the scene
    */
}
*/
public Vector getHiddenItems() {
    return Hidden_Items;
}

/*
 * Returns all the visible items in the scene
 */
public Vector getVisibleItems() {
    return Visible_Items;
}

/*
 * Returns all visible and hidden items in the scene
 */
public Vector getAllItems() {
    Vector all_items = new Vector();
    all_items.addAll(Hidden_Items);
    all_items.addAll(Visible_Items);
    return all_items;
}

/*
 * Deletes an item from the scene
 * Returns true if successfully deleted, otherwise false
 */
public boolean delItem(Item i) {
    if (Visible_Items.contains(i))
        return Visible_Items.remove(i);
    else if (Hidden_Items.contains(i))
        return Hidden_Items.remove(i);
    else
        return false;
}

/*
 * Adds an item to the scene with the option of it being hidden
 */
public boolean addItem(Item i, boolean hidden) {
    if (hidden)
        return Hidden_Items.add(i);
    else
        return Visible_Items.add(i);
}

/*
 * Returns the number of visible items
 */
public int numVisibleItems() {
    return Visible_Items.size();
}

/*
 * Returns the visible exits
 */
public Vector getVisibleExits() {
    return Visible_Exits;
public Vector getHiddenExits() {
    return Hidden_Exits;
}

public Vector getAllExits() {
    Vector all_exits = new Vector();
    all_exits.addAll(Hidden_Exits);
    all_exits.addAll(Visible_Exits);
    return all_exits;
}

public boolean addExit(Scene s, boolean hidden) {
    if(hidden)
        return Hidden_Exits.add(s);
    else
        return Visible_Exits.add(s);
}

public boolean delExit(Scene s) {
    if(Hidden_Exits.contains(s))
        return Hidden_Exits.remove(s);
    else if(Visible_Exits.contains(s))
        return Visible_Exits.remove(s);
    else
        return false;
}

public String getPicture() {
    return Image;
}

public void setPicture(String s) {
    Image = s;
}
public boolean hasExit(Scene s) {
    return (getAllExits().contains(s));
}

public boolean hasItem(Item i) {
    if(Hidden_Items.indexOf(i) == -1 && Visible_Items.indexOf(i) == -1)
        return false;
    return true;
}

A10. Character.java
/*
 * File: Character.java
 *
 * Author: Mariya Nomanbhoy
 */

import java.util.Vector;

/*
 * This is a Character class. It holds the information for the
 * characters in the game such as the items they hold and their name.
 * It does not hold their scene as characters can be members of multiple
 * scenes at once.
 */

public class Character extends Thing {

    private Vector Items; /* items being held */

    /* Constructor takes the name as a String and creates a new
     * character holding 0 items.
     */
    public Character(String n) {
        super(n);
        Items = new Vector();
    }

    /* Pick up an item, therefore add it to the items being held.
     * This takes an item as a parameter and returns void. This
     * method will only be used for declarations, as it is only
     * during declarations that the Character can pick up an item
     * not in the current scene. This usage is guaranteed by
     * translator.
     */
public void pickUp( Item i ) {
    Items.add( i );
}

/*
 * Pick up an item, therefore add it to the items being held.
 * This takes an item and a Scene as parameters and returns
 * void. Will exit game if item is not in the scene passed and
 * therefore cannot be picked up. This method is used at all
 * times except during declarations.
 */
public void pickUp( Item i, Scene s ) {
    if( s.hasItem( i ) ) {
        Items.add( i );
        s.delItem( i );
    } else {
        System.out.println( "Runtime Error: Character "
            + "trying to pick up item " + i.getName()
            + " which is not in the scene." );
    }
}

/*
 * Drop an item, therefore remove it from the items being held.
 * This takes an item as a parameter and returns void. Will
 * exit game if item is not being held by the character and
 * therefore cannot be dropped.
 */
public void drop( Item i, Scene s ) {
    if( hasItem(i) ) {
        Items.remove(i);
        s.addItem( i, false );
    } else {
        System.out.println( "Runtime Error: Character "
            + "trying to drop an item " + i.getName()
            + " which they do not have." );
    }
}

/*
 * returns number of items held by character as an int.
 */
public int numItems() {
    return Items.size();
}

/*
 * returns the items held by user as a vector of items.
 */
public Vector items() {
    return Items;
}
* This method takes an item as a parameter and returns a boolean. * returns true if Item i is being held by Character, otherwise * returns false.
*/
public boolean hasItem( Item i ) {
    if( Items.indexOf( i ) == -1 )
        return false;
    return true;
}
}

A11. Item.java
/*
 * File: Iteme.java
 *
 * Authors: William Liu
 */
/*
 * Item Class
 */
public class Item extends Thing {

    /*
     * Constructor initializes name of item and hidden flag
     */
    public Item(String n, boolean hid) {
        super(n);
        boolean hidden = hid;
    }

    /*
     * Constructor initializes the item with just the name
     */
    public Item( String n ) {
        super(n);
    }
}

A12. User.java
/*
 * File: User.java
 *
 * Author: Mariya Nomanbhoy
 */
import java.util.Vector;

/*
 * This is a user class. It holds the information for the person
 * playing the storybook game such as the items and the scene
 * they is in.
 */
public class User {
    private Scene Cur_Scene;  /* the scene the user is currently in */
    private String Said;      /* Last input by the user */
private Vector Items;   /* List of items held by the User */

/*
 * Constructor for user. This initializes the user to have 0
 * items and sets the scene and the last thing said to NULL
 */
public User() {
    Items = new Vector();
    Said = null;
    Cur_Scene = null;
}

/*
 * Get the last thing said by the user. Takes no parameters and
 * returns a string.
 */
public String getSaid() {
    return Said;
}

/*
 * Set the last thing said by the user. Takes a string as a
 * parameter and returns void.
 */
public void setSaid( String s ) {
    Said = s;
}

/*
 * Pick up an item, therefore add it to the items being held.
 * This takes an item as a parameter and returns void. Will
 * exit game if item is not in current scene and therefore
 * cannot be picked up.
 */
public void pickUp( Item i ) {
    if( Cur_Scene.hasItem( i ) == true ) {
        Items.add( i );
        Cur_Scene.delItem(i);
    } else {
        System.out.println("Runtime Error: User picking "+"up item " + i.getName() + " that is not in "
+"the scene.");
        System.exit(0);
    }
}

/*
 * Drop an item, therefore remove it from the items being held.
 * This takes an item as a parameter and returns void. Will
 * exit game if item is not being held by the user and therefore
 * cannot be dropped.
 */
public void drop( Item i ) {
    if( hasItem( i ) ) {
        Items.remove(i);
        Cur_Scene.addItem( i, false );
    }
else {
    System.out.println("Runtime Error: User dropping" + " item "+ i.getName()+ " that it does not " +"have.");
    System.exit(0);
}

    */
    * returns number of items held by User as an int.
    */
    public int numItems() {
        return Items.size();
    }

    */
    * returns the items held by user as a vector of items.
    */
    public Vector items() {
        return Items;
    }

    */
    * This method takes an item as a parameter and returns a boolean. 
    * returns true if Item i is being held by user, otherwise returns 
    * false.
    */
    public boolean hasItem( Item i ) {
        if( Items.indexOf( i ) == -1 )
            return false;
        return true;
    }

    */
    * This method returns the current scene of the user. It takes 
    * no parameters and returns a Scene object.
    */
    public Scene getCurScene() {
        return Cur_Scene;
    }

    */
    * This method moves the character to new scene by changing the 
    * current scene. It takes a Scene as a parameter and returns 
    * void. If the Scene passed is not an exit of the current scene 
    * a runtime error is printed and the game is exited.
    */
    public void moveTo( Scene s ) {
        if( Cur_Scene == null || Cur_Scene.hasExit( s ) ) {
            Cur_Scene = s;
        } else {
            System.out.println( "User trying to move " + " to exit " + s.getName() + " which is " +"not a valid exit for this scene.");
            System.exit(0);
        }
import sys, os, string
import difflib, pprint
# import fileinput, glob

#extension of test scripts
extt = "gmm"

#extension of test script outputs
exto = "tst"

#GRIMM translator class after build
grimm_class = "GRIMM.class"

#command to execute grimm compiler
cmd = "java GRIMM "

#############################################################################
##
#parse command line arguments
if len(sys.argv) < 2:
    print "Usage: " +sys.argv[0]+ " <test_scripts>"
    sys.exit(0)

#Make sure GRIMM translator exists
if not os.path.isfile(os.path.join(os.path.dirname(sys.argv[0]),
grimm_class)):
    print "GRIMM translator does not exist. Make it first."
    sys.exit(0)

files = sys.argv[1:]

for f in files:
    typed_input = f #saved the user inputted path for better output
    f = os.path.abspath(f)
    #split the filename and extension
    try: name, ext = os.path.basename(f).split('')
    except: continue

    if ext != extt: continue #file is not a grimm test file
    try: os.stat(f) #make sure the input file exists
    except:
        print f + " does not exist"
        continue

    try: #read in the test output file if it exists
        test_output = os.path.join(os.path.dirname(f), name+'.tst')
        output_ideal = open(test_output).readlines()
except: #otherwise, it's a successful test and open the default output
test_output = os.path.join(os.path.dirname(f), 'success.tst')
output_ideal = open(test_output).readlines()

working_dir = os.getcwd() #save the current working directory
os.chdir(os.path.dirname(sys.argv[0])) #change into GRIMM directory
#run test and save it's output
cmd_inf, cmd_outf = os.popen4(cmd + f)
os.chdir(working_dir) #go back to the working dir
os.wait()
output_test = cmd_outf.readlines()

#compare the outputs
if output_test == output_ideal:
    print typed_input + ' passed'
continue
else:
    diff = difflib.Differ() # instantiate a differ object
    #diff the two outputs
    result = list(diff.compare(output_ideal, output_test))
    print '-------------------------------------------------------'
    print f + ' test failed with diff:'
    for l in result:
        if l[0:2] != '  ':
            pprint.pprint(l.rstrip())
    print '-------------------------------------------------------'