Grimm: Language Reference Manual

1. Lexical Conventions

1.1. ALPHA

An ALPHA is a Roman alphabet character that can be either uppercase or lowercase. ALPHA's are a protected type, and therefore they are not a valid identifier in the Grimm language. Instead they are used to define other identifiers.

1.2. DIGIT

A DIGIT consists of any numerical digit from 0 to 9. DIGIT is a protected type, and therefore they are not a valid identifier in the Grimm language. Instead they are used to define other identifiers.

1.3. Tokens

There are 6 classes of tokens: NOUN, STRCONST, NEWLINE, Whitespace, Comments, and keywords.

1.3.1. NOUN

An identifier can be any alpha numeric combination beginning with an alphabet character. Upper and lowercase letters are considered to be different. A noun may not have the same spelling as a key word.

1.3.2. STRCONST

A series of charcters surrounded by double quotes.

1.3.3. NEWLINE

A line terminator signals the end of a line. Empty lines are ignored, therefore one NEWLINE is treated the same as multiple NEWLINE characters in a row.

1.3.4. Whitespace

Grimm is a partially white space sensitive language. NEWLINE characters indicate the end of a line are used to organize the text within a script. No statements can extend for more than one line. All other whitespace listed below is ignored.

1.3.5. Comments

Comments are used by the programmer to leave notes for himself and others who may use the code. Therefore, comments are appropriately indicated by the character sequence `:note:` and end when a NEWLINE is reached, therefore comments may only be one line long.

1.3.6. Keywords

The following words are keywords in the Grimm language and may not be used as NOUN's.

```
and             gameover        item            read
```

character	goto	name	say
contains	has	not	says
description	hidden	or	scene
drops	holds	otherwise	then
endif	if	otherwiseif	user
endwhile	input	pickup	while

1.3.7. User keyword

The user keyword is a variable that containes 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. The user keyword is combined with keywords has, says, pickup, and drops which are described below. The user keyword can also be used in the expression `read user input` to extract input from the terminal.

2. Statements

Statements are the elements of the program, which executed during runtime. The statements are executed sequentially unless otherwise indicated by the flow-control statements. The end of a statement is indicated by a NEWLINE, and therefore no statement can be longer than one line. There are three basic types of statements: action statements, if statements, and while statements.

2.1. Action statements

An action statement indicates that some action should be executed. The jump, say, and pickup actions are user actions. They can be typed into the console by the user, and then executed if they are valid commands. The user input, char says, and gameover actions are strictly used by the programmer and only used explicitly in the code.

2.1.1. Jump

A jump statement allows the user to move from one scene to another. This is indicated by using the goto keyword and then followed by a NOUN, which is the name of the scene that the user will move to.

```
goto hallway
```

2.1.2. Say

A say statement allows the user to speak to a character in the scene. The user may say anything, although only certain questions or sentences may produce a meaningful response. The action is indicated by the keyword say followed by a STRCONST.

```
say "hello"
```

2.1.3. Pick up

The user may pick up items in the scene. This is indicated by the pickup keyword followed by a NOUN, which is the item to be picked up. There is an optional NOUN before the pickup key word so that the programmer can indicate if a character or a user will pickup the item. If no beginning NOUN is specified the pickup action is associated with the user by default.

```
pickup key
```

2.1.4. Drop

The programmer may allow for a user or a character to drop an item, therefore no longer possessing it. An optional NOUN preceding the `drop` command specifies whether the user of the character is executing the drop actions. If this NOUN is not included, the `drop` command defaults to the user.

2.1.5. User input

The programmer may read the user input and act accordingly using the user input action. This input will be stored in memory and can be used in the flow-control statements described is section 2.2 and 2.3. This is indicated by the sequence of key words read user input.

```
read user input
```

2.1.6. gameover

The game over action is used when the user finishes the story. It is indicated by using the keyword gameover.

```
gameover
```

2.2. Conditional Structures

2.2.1. Boolean Expressions

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

2.2.1.1. user says STRCONST

The user says expression is used to evaluate whether the last input by the user is equal to a specific STRCONST.

```
user says "hello"
```

2.2.1.2. user inside scene

This expression is used to evaluate which scene the user is currently in. It returns true if the user is currently in scene.

```
user inside kitchen
```

2.2.1.3. (user | char) has item

This expression evaluates to true if the user (or char) has the item.

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

2.2.2. If Statements

If statements are part of the flow-control of the language. The first line must always be indicated by the `if` followed by a boolean statement, followed by the `then`. On successive lines there can be a series of statements. Next there may be an `otherwiseif,` followed by a boolean statement or a series of statements. There can be as many `otherwiseif`'s as needed. The end of an `if` statement must be indicated on a new line using the keyword `endif`.

Using boolean expressions we can create complicated if statements. Here is an example of code:

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

2.2.3. While Statements

While statements are the second part of the flow-control of the GRIMM language. The evaluate a boolean expression and then execute a sequence of statements. The boolean expression will then be evaluated. If it is still true the sequence will be executed again. Otherwise it will not. 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 on a newline.

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

3. Structure of a GRIMM script

A GRIMM script has three main parts: the declarations, the assignments, and the action loop. A program must contain the action loop section(although it does not necessarily have to be a loop) but does not have to contain the declarations or assignments section. An empty file is not a valid GRIMM script.

3.1. Declarations

The declarations section is where the programmer declares all of it scenes, characters, and items. There does not have to be declarations of all three different types.

3.1.1. Scenes

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 scenes available to the storybook. 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
```

3.1.2. Characters

The programmer may choose to put characters in all or some of their scenes. Characters can appear in multiple scenes. These characters are declared using the keyword character and then the variable name.

```
character steven
```

3.1.3. Items

The programmer may also choose to put items in all or some of their scenes. Again, items can appear in multiple scenes. These items are declared using the keyword item and then the variable name.

```
item key
```

3.2. Assignments

Following the declarations section each scene, character, and item is assigned certain attributes.

3.2.1. 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 at the top of the console. Beneath this text the image will be displayed. There are a number of defined exits which will be displayed at the bottom of the console. Additonally, any items contained in the room will be displayed. The user can either type in an action to be executed in the scene, or choose to move on to another scene by choosing one of the exits. The option to leave the game will always be an exit. As a user may need an item to use a 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.

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

3.2.2. Characters

Each character is defined by specifying its name and the items that character possesses. For example:

```
steven name "Stevie"
steven holds item knife
```

3.3. Action Loop

The action loop is where all actions specified either by the programmer or the user are executed. All scenes, character, and items used here must be declared in the declarations section. This section can be any sequence of valid statements. For example:

```
while user inside hallway
     read user input
     if user says "pickup key" then
          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 don't know what that means"
     endif
endwhile
```