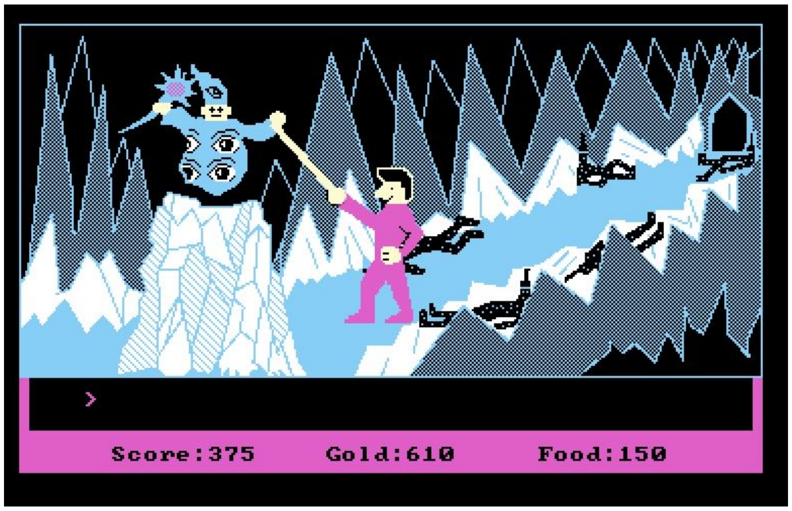

TBAG: The Text-Based (Maze) Game Language

Language Reference Manual

Gregory Luan Chen <glc2121>
Yu Chun (Julie) Chien <yc2937>
Maria Van Keulen <mv2482>
Brian Slakter <bjs2135>
Iris Zhang <iz2140>

1 Lexical Elements

1.1 Identifiers

Identifiers are strings used for naming different elements, such as variables, functions, classes. These identifiers are case sensitive, and can involve letters, digits, and underscore '_', but should always start with a letter. These rules are described by the definitions involving regular expressions below:

identifier	:= (letter) (letter | digit | underscore)*
digit	:= '0' - '9'
letter	:= uppercase_letter | lowercase_letter
uppercase_letter	:= 'A' - 'Z'
lowercase_letter	:= 'a' - 'z'

1.2 Keywords

These keywords are reserved for use in the language and therefore cannot be used as identifiers. These keywords are case sensitive:

int	room	if	true	func
string	adj	else	false	return
bool		while		print

1.3 Literals

1.3.1 String Literals

String literals are a sequence of zero or more letters, spaces, digits, other ASCII characters numbers 32 to 126, excluding the single quote (number 39). These strings should be enclosed in single quotes.

Example: 'happy'.

1.3.2 Integer Literals

Integer literals are a sequence of one or more digits. Only numbers in decimal format are recognized.

Example: 42

1.4 Operators

Operators are tokens that are utilized for performing actions on different elements. Common operations performed by such operators are addition, subtraction, and other mathematical processes. These will be discussed further in section 3.

Example: +

1.5 Delimiters

Delimiters are special characters that separate elements of a program, commonly used to help improve the visual clarity of the written code.

1.5.1 Parentheses and Braces

Parentheses are used to force evaluation of parts of a program in a specific order. They are also used to enclose arguments for a function.

1.5.2 Commas

Commas are used to separate function arguments.

1.5.3 Brackets

Brackets are used for arrays, including creation of arrays as well as, assignment and access of array elements.

1.5.4 Semicolon

Semicolons are used to terminate a sequence of code.

1.5.5 Curly Braces

Curly braces are used to enclose function definitions and while loops, as well was room and adjacency declarations.

1.6 Whitespace

Whitespace is used to separate tokens, but has no other special meaning otherwise.

2 Data Types

Data types are statically typed. They are assigned a type upon their declaration, and maintain that type throughout the life of the program.

2.1 Primitive Data Types

There will be only three primitive data types, keeping the language simple while still providing what is necessary to implement a somewhat complex program.

2.1.1 Integers

All numbers will be of type int. This 4 byte data type will range in value from −2,147,483,648 to 2,147,483,647.

2.1.2 Strings

All text-based values will be of type string. There is no need for a char data type as these can simply be treated as single character strings.

2.1.3 Booleans

Boolean values true and false can be used, with type boolean.

2.2 Non-Primitive Data Types

2.2.1 Arrays

Arrays are list-like containers that can be used to hold both primitive and non-primitive data-types. All elements of an array must be of the same type.

2.2.1.1 Declaring Arrays

You can declare an array by indicating the type of the elements that the array will contain, followed by an identifier for the array, and finally the brackets enclosing the number of elements an array will hold. For example:

```
int my_array [5];
```

2.2.1.2 Accessing and setting array elements

Array elements can be accessed by providing the desired index of the element in the array you wish to access enclosed within brackets next to the identifier of the array. For example:

```
my_array $ [0]
```

Array elements can be set by accessing the desired index in which to place the item, and then assigning it to the desired value. For example:

```
my_array $ [0] = 4;
```

To get an array's length, use the arr_length() function. For example:

```
int size = arr_length(my_array)
```

2.2.2 Rooms

Rooms are created using a "room" keyword followed by a block of code that contains that room's fields. The syntax to then create a room called "Home" with fields "desc" and "welcome_msg" defined:

```
room {
      string desc;
      string welcome_msg;
}

room home {
      desc = 'My House';
      welcome_msg = 'You wake up in your bed.'';
}
```

Repeat as needed to create additional rooms. Rooms automatically come with a field 'adj' that is an array of rooms. To add rooms to a room's adjacency list, use the adj keyword:

```
adj {home, final};
```

This simple line will perform the following function under the hood:

```
home.adj[arr_length(home.adj)] = final;
final.adj[arr_length(final.adj)] = home;
```

3 Expressions and Operators

3.1 Expressions

Expressions are made up of one or more operands and zero or more operators. Innermost expressions are evaluated first, as determined by grouping into parentheses, and operator precedence helps determine order of evaluation. Expressions are otherwise evaluated left to right.

3.2 Operators

The table below presents the language operators (including assignment operators, mathematical operators, logical operators, and comparison operators), descriptions, and associativity rules. Operator precedence is highest at the top and lowest at the bottom of the table.

Operator	Description	Associativity
* / %	Multiplication, division, modulo	Left-to-right
+ -	Addition, subtraction	
< <=	Inequality Operators: Less Than, Less Than Or Equal	
> >=	Inequality Operators: Greater Than, Greater Than Or Equal	
== !=	Equal, Non-Equal	
=	Assignment	Right-to-left

4 Statements

4.1 The if Statement

The if statement is used to execute a statement if a specified condition is met. If the specified condition is not met, the statement is skipped over. The general form of an if statement is as follows:

```
if (condition)
      action1;
else
    defaultaction;
```

"if" must be followed with "else," although the else may have no statement associated with it:

```
if (condition)
      action1;
else
```

4.2 The while Statement

The while statement is used to execute a block of code continuously in a loop until the specified condition is no longer met. If the condition is not met upon initially reaching the while loop, the code is never executed. The general structure of a while loop is as follows:

```
while (condition) {
    action1;
    action2;
    action3;
}
```

5 Functions

5.1 Function Definitions

Function definitions consist of an initial keyword “func,” a return-type, a function identifier, a set of parameters and their types, and then a block of code to execute when that function is called with the specified parameters. An example of the GCD function definition is as follows:

```
func int sum (int a, int b){
      return a + b
}
```

5.2 Calling Functions

A function can be called its identifier followed by its params in parentheses.
for example:

```
sum(1, 2)
```

6 Program Structure and Scope

6.1 Program Structure

A TBAG program must live entirely within one source file.

The three parts of a TBAG program are room declarations, adjacency specifications, and function declarations. They must come in that order.

6.2 Scope

Any declarations made within the program that are not within one the block of an if statement, a while statement, and a function definition are available for reference any point later in the program. Declarations made within blocks of an if statement, a while statement, or a function definition are only available for reference within that block. Declarations are never visible to any code that comes before it in the program.

7 Built-in Functions

7.1 The print function

The print function can be used to print out strings to the command line. The general structure for calling the print function is as follows:

```
print(‘welcome to the jungle’);
```

Anything within the parentheses will be printed; it must be of type string.

7.2 int_to_string

takes an int and turns it into a string

```
int a = 1;
string aString = int_to_string(a);
```

7.3 string_to_int

takes an string and turns it into a int

```
string myStr = ‘1’;
int myInt = string_to_int(myStr);
```

7.4 arr_length

returns the number of elements in an array. for example”

```
int size = arr_length(myArr);
```

7.5 next_input

takes the next input from i/o, waiting for newline to terminate

```
string token = next_input();
```

8 CFG

program	→	roomdecl adj fdecl EOF
roomdecl	→	manrdecl rdecl
manrdecl	→	ROOM LBRACE vdecl_list RBRACE
rdecl	→	|
		ROOM ID LBRACE vdecl_list stmt_list RBRACE rdecl
adj	→	|
		ADJ RBRACE adj_stmt_LBRACE adj
adj_stmt		→ ID COMMA ID SEMI
array_access	→	LBRACE INT RBRACE
array_decl	→	type LBRACE RBRACE
type	→	INT | STRING | BOOL
fdecl	→	FUNC BOOL ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
fdecl		
formals_opt	→	|
		formal_list
formal_list	→	ID |
		formal_list COMMA ID
vdecl_list	→	|
		vdecl_list vdecl
vdecl	→	INT ID SEMI |
		STRING ID SEMI |
		BOOL ID SEMI |
		array_decl ID SEMI
stmt_list		→ | stmt_list stmt
stmt	→	expr SEMI |
		RETURN expr SEMI |
		LBRACE stmt_list RBRACE |
		IF LPAREN expr RPAREN stmt_list ELSE |
		IF LPAREN expr RPAREN stmt_list ELSE stmt_list |
		WHILE LPAREN expr RPAREN stmt_list
expr	→	LITERAL |
		ID |
		ID DOLAR_SIGN array_access |
		expr PLUS expr |
		expr MINUS expr |
		expr TIMES expr |
		expr DIVIDE expr |
		expr EQ expr |
		expr NEQ expr |
		expr LT expr |
		expr LEQ expr |
		expr GT expr |
		expr GEQ expr |
		ID ASSIGN expr |
		LPAREN expr RPAREN

9 Sample Program: sampleGame1.tbag

```
room {
       string desc;
       string welcome_message;
}

room Home {    desc = 'Home';
               welcome_message = ‘You wake up in your bed.’;
}

room Final {   desc = 'Final';
               welcome_message = ‘Congratulations! You made it outside.’;
}

room location = Home;

adj {Home, Final}

func bool win(){
       if(location == Final)
               return true;
       else
               return false;
}

func bool main(){
       print(‘You have begun your journey in ‘ + location.desc);
       print(location.welcome_message);
       print(‘Here are your choices for where to proceed:’);
       print(‘’);
       int i = 0;
       while (i < arr_length(location.adj)) {
               print(int_to_string(i) + ‘ ‘ + location.adj[i].desc);
               i = i + 1;
       }
       while( win() == false && (input=next_input()) != 'Q') {
               location = location.adj[string_to_int(input)];
               int i = 0;
               while (i < arr_length(location.adj)) {
                       print(location.welcome_message);
                       print(int_to_string(i) + ‘ ‘ + location.adj[i].desc);
                       i = i + 1;
               }
       }
       if (input == 'Q')
               print(‘you quit, quitter’);
       else
             print('you won');
}
```