BGGL (Board Game Generator Language)

COMS W4115: Programming Languages and Translators
Professor Stephen Edwards
Final Report

@cs.columbia.edu
{
Matt Chu (mwc2110)
Steve Moncada (sm2277)
Hrishikesh Tapaswi (hat2107)
Vitaliy Shchupak (vs2042)
}

Contents

Chapter 1: Introduction

1.1 Overview

Coding any of the familiar board games using a standard object-oriented programming language like C++ or Java would prove decidedly tedious. Even the most expert OOL-user would waste a large amount of time specifying the (often repetitive) characteristics of traditional board games. BGGL seeks to dramatically cut down on the time a board game programmer would traditionally spend tweaking the class structure of his code while significantly enhancing the way gameplay terms (i.e. rules) are implemented. Programmers using BGGL will never spend time devising the most efficient board game framework, instead, they'll start with a palette of boards, regions, pieces, and rules and begin implementing their idea immediately.

1.2 Goals

The goal of BGGL is to abstract board games' traditional frameworks and provide the programmer with a rich set of language constructs that can be used to intuitively implement the rules and flow of a game. Throughout the development of BGGL, the abstraction and reduction of non-essential or monotonous tasks as a primary focus. Ideally, BGGL will not only be used by programmers seeking to mimic the functionality of existing board games electronically, but utilized as an instrument for developing new games. Indeed, what we find most exciting about BGGL's potential relates most closely to the development of new games. BGGL will quickly elucidate possible pitfalls and inspire potential enhancements.

1.3 BGGL-Specific Language Conventions

1.3.1 Game

A game in BGGL is the equivalent of the main method in traditional programming languages. Inside the game block the BGGL compiler expects to find variable initialization, console input/output, control flow, and Piece movements.

1.3.2 Board

The BGGL board variable is the region on which all other BGGL types perform their tasks. Most other components reference the board in some way. The board is initialized inside the game block of a *.bggl file. Using a combination of empty pieces '_' , invalid pieces '#' , and existing user-defined pieces, any type of board is possible. By this we mean boards of any shape may be specified, as the Chinese checkers example will demonstrate below. Currently, the BGGL does not explicitly support certain complex board configurations, as the internal global variable board is represented as two array-like structures (hence, rectangular).

1.3.3 Piece
Pieces in BGGL are very simple. Their name is basically their only attribute, as the board keeps track of piece location and BGGL moves and rules dictate whether or not a piece may be manipulated in some meaningful board game manner.

1.3.4 Move
A move in BGGL is defined by a special tuple syntax delimited by colons ':'. A move acts on a piece (and consequently, the board) in one of three ways: 1) adding a piece to the board, 2) removing a piece from the board, or 3) moving a piece from one board coordinate to another. Using the custom syntax, most moves get their inputs interactively from the user or from the specific context of each game block. The BGGL interpreter does automatic bounds checking on each move as it is executed.

1.3.5 Rule
Rules in BGGL test piece movement. They are specialized versions of functions, but are syntactically very similar. A rule returns a boolean in BGGL by performing a series of 'tests' on a certain piece movement, specified in a rule block by a 4-tuple which is discussed at length below.

Chapter 2: Tutorial

2.1 Getting Started

2.1.1 Environment Considerations

BGGL requires a Java 1.5 environment to run. We found that the best way to execute BGGL code would be to write scripts for compilation and execution. These scripts were tested on the three major operating systems and each compiled and ran smoothly.

2.1.2 Running a BGGL Program

To run the BGGL demo, which contains three brief programs (prime-number validator, tic-tac-toe, and Chinese checkers), cd into the bggl /bin directory and run the file named run-demo.sh.

2.2 BGGL Basics

Every BGGL game has four key components: rules, a game block, the specification of a board, and moves. The tic-tac-toe example in section 2.3 illustrates the application of these four components.

RULES

Rules are specialized functions in BGGL which define allowable piece movements. They act just like functions but their syntax is a little different. Each rule may be defined as follows:

rule <*identifier*>() : *target-piece-list* {
    return test *length* , *direction* , *jump* , *emptysquare* ;
}

where

target-piece-list is a comma-separated list of preivously-initialized pieces.
length is an integer which specifies how many valid squares a piece a may move.
direction is one of three keywords (diag | ortho | orthodiag) or an array of custom movement values.
jump is a boolean which specifies if a piece must jump another piece to make a move (true) or not jump another piece to make a move (false).
emptysquare is a boolean which specifies if a piece must land on an empty square at the end of its move.

Rules implicitly return boolean values and are composed of tests. Tests are the 4-tuple outlined above. The default value for each of the components of a test are outlined in Section 3.

In tic-tac-toe, the one and only rule-declaration that needs to be made explicit is below:

```
rule no_overwrites() : X, O {
     return test , , , true;
}
```

THE GAME BLOCK
The game block is BGGL's equivalent of the main method. Inside the game block the programmer initializes board specifications, determines control flow, wires the program to take input from the console, and manipulates the global board variable with moves. The game block always proceeds rule and function declarations. All of the board game's variables are also declared within the game block.

THE BOARD
Every game block contains a formal intialization of the global board variable. Moves are subsequently applied to this board. The compiler will automatically do checks on the bounds of the board for the rest of the game block. The initialization of the board for tic-tac-toe is simple:

```
board =  < [ _ , _ , _ ] ,
           [ _ , _ , _ ] ,
           [ _ , _ , _ ] >;
```

The '_' symbol means 'empty square.' Other initializations call for invalid '#' regions or actually set pieces onto the board at initialization. BGGL keeps a rectangular matrix as the internal board representation, but this doesn't limit the programmer's ability to specify a polygonally-shaped board. The example of Chinese checkers in section 6.2 illustrates this point.

MOVES
Move syntax is a special feature of BGGL. Since rules are defined to check if a move is valid, once a move is initialized, it is passed into a rule and the specified movement is tested in the rule blocks test structure(s). A move can be one of three things: 1) the addition of a piece, 2) the removal of a piece, or 3) a move of a piece from a source board coordinate to a destination board coordinate. A move is a 4-tuple in the first two cases and a 6-tuple in the third. A move has the following syntax:

move : < *move-type* > : < *piece* > : < *row1* > : < *col1* > (: < *row2* > : < *col2* >);

move-type is a special token which distinguishes each move, '+' is an add, '-' is a remove, and '^' is a movement from one board coordinate to another.
piece is the identifier of an already-initialized piece.

row1, col1 is the target set of coordinates in the add '+' and '-' remove contexts, but in the move '^' it doubles as the source set of coordinates, with *row2, col2* set as the targets.

The BGGL compiler will do automatic bounds checking on each rule, but the rules specified by the programmer are also easily checked using BGGL syntax. Usually, this type of statement is executed as a boolean inside a control flow structure like an if () block. The syntax for applying a rule on a movement is below, followed by the tic-tac-toe example inside the if statement.

rule-invocation : *move-identifier*
apply move-identifier ;

```
row = input "Enter row coordinate: ", int;
col = input "Enter col coordinate: ", int;
move m;
currpiece = getpiece(thisplayer);
m = :+:currpiece:row:col;

if (no_overwrite():m) {

      // additional game control flow
      apply m;
}
```

2.3 Example: Tic-tac-toe

```
piece X;
piece O;
player p1;
player p2;

rule no_overwrite(): X, O {
      return test , , , true;
}

func getpiece(player p) returns piece {
      if (p == p1) {
            return X;
      } else {
            return O;
      }
}

func getwinner() returns player {
```

```
	int i;
	player winner;
	for (i = 0 to 2) {
		if (<_i> == [X,X,X] || <|i> == [X,X,X] ||
			</0> == [X,X,X] || <\0> == [X,X,X]) {
			winner = p1;
		} else {
			if (<_i> == [O,O,O] || <|i> == [O,O,O] ||
				</0> == [O,O,O] || <\0> == [O,O,O]) {
				winner = p2;
			}
		}
	}
	return winner;
}

game
{
	//empty tic tac toe board
	board =
	<[_,_,_]
	 [_,_,_]
	 [_,_,_]>;

	boolean done = false;
	player thisplayer = p1;
	int row;
	int col;
	piece currpiece;
	print board;
	int countmoves=0;
	while (!done) {
		print "Player " + thisplayer + ": " + getpiece(thisplayer);
		row = input "Enter row coordinate: ", int;
		col = input "Enter col coordinate: ", int;

		move m;
		currpiece = getpiece(thisplayer);
		m = :+:currpiece:row:col;

		// check if it is legal
		if (no_overwrite():m) {
			apply m;
			if (thisplayer == p1) {
				thisplayer = p2;
			}
			else {
				thisplayer = p1;
```

```
                }
                countmoves = countmoves + 1;
            }
            else {
                print "Invalid coordinate";
            }
            print board;

            player winner = getwinner();
            if (winner == p1 || winner == p2) {
                print "" + winner + " won!";
                done = true;
            }
            else {
                if (countmoves == 9) {
                    print "It's a draw!";
                    done = true;
                }
            }
        }
}
```

Chapter 3: Language Reference Manual

3.1 Lexical Conventions

There are five classes of tokens used in BGGL. They are identifiers, keywords, numbers, string literals, and other tokens. White space is used to separate different tokens in BGGL. In the context of this language, 'white space' refers to spaces, tabs, and new lines. If the input stream has been parsed into tokens up to a given character, the next token will be the longest string of characters which could possibly constitute a new token.

3.1.1 Comments

- Multi-line comment-style introduced by '/*' and terminated by '*/' causes the compiler to ignore the characters within the two tokens.
- Single line comment-style introduced by '//' causes the compiler to ignore the rest of the line.
- Multi-line comments have precedence over single-line comments, *see example*.

EXAMPLES:

```
/* this is a                          // this is a single line comment.
       multi-line
          // comment */
```

3.1.2 Identifiers

Identifiers in BGGL may be composed of any number of letters, digits, and underscores whose first character is a letter. Identifiers are case-sensitive.
All identifiers in BGGL, including ones for variables, functions, and domain-specific constructs, inhabit the same namespace—which is the same namespace as all keywords in BGGL. Identifiers will hereafter be denoted ***identifier*** in the code examples below.

3.1.3 Keywords

The following keywords are reserved and therefore may not be used as identifiers:

`array`	`false`	`input`	`piece`	`string`	`width`
`apply`	`for`	`int`	`player`	`test`	
`boolean`	`func`	`move`	`print`	`to`	
`break`	`game`	`nothing`	`return`	`true`	
`diag`	`height`	`ortho`	`returns`	`turn`	
`else`	`if`	`orthodiag`	`rule`	`while`	

3.1.4 Numbers
Numbers (integer constants) in BGGL are represented by a sequences of one or more digits. There are no floating-point numbers in BGGL.

3.1.5 String Literals
A string literal is a sequence of characters enclosed by double quotes ' ” '. A double quote inside the string is represented by two adjacent double quotes ' “” '.

3.1.6 Other Tokens
The following characters and character pairs are also used in BGGL, their functionality will be discussed later in this chapter:

_	{	+	%	@	=	>=	||
#	}	-	^	|	!	<=	,
[	(	*	:	>	!=	&&	\
]	)	/	;	<	==	\\	

3.2 Types
The names of the types below will hereafter be denoted by ***type***.

```
array      :    static int-index-based collections of types
                default value: []

boolean    :    Boolean values which evaluate to true or false
                default value: false

int        :    standard 32-bit integers
                default value: 0

string     :    a sequence of one or more characters
                default value: “”

board      :    BGGL-specific board type, creates coordinate plane
                default value: <>

move       :    BGGL-specific movement type
                default value: ‘ : : : : ’ (the empty move)

piece      :    BGGL-specific game piece type
                default value: identifier name

player     :    BGGL-specific player type
                default value: identifier name
```

3.3 Expressions

The following section explains the behavior of expressions (henceforth denoted *expression*) in BGGL. The order of the subsections in this section indicates expression operator precedence (from highest to lowest) and the specific linear associativity will be explicitly stated within each subsection. Within each section, the operators have the same precedence. For the most part, expressions in BGGL almost identically follow the conventions adopted by Java.

3.3.1 Primary Expressions

Primary expressions include identifiers, parenthesized expressions, function calls, and BGGL-specific constructs which are not arithmetic, logical, or relational. A function call contains a primary expression followed by parentheses containing a possibly empty, comma-separated list of expressions which constitute the arguments to a function. The associativity of primary expressions is left-to-right (primary expressions are right-value expressions, which means they can appear on the right side of assignment statements).

SYNTAX: *identifier* | *number* | *string* | (*expression*) | true | false
APPLICABILITY: all types, literals

3.3.2 Array-Element Lookup Operator

The syntax for retrieving an element from an array in BGGL is different from standard programming languages. The at symbol '@' is used to reference array elements. The syntax follows:

SYNTAX: *array expression* @ *integer expression*

where array expression = an expression which resolves to an array
where integer expression = an expression which resolves to an int

EXAMPLE: [1,2,3]@1
(returns 2)

3.3.3 Multiplicative Operators

Multiplicative operators include the tokens *, /, and %. The binary * triggers multiplication. The binary / triggers division. The binary % triggers the modular division operator, which gives the remainder from the division of the first expression by the second expression. The unary – triggers the multiplication by (-1). The associativity of multiplicative operators is left-to-right.

SYNTAX: *expression* * *expression*
expression / *expression*
expression % *expression*
- *expression*
APPLICABILITY: int

3.3.4 Additive Operators

The additive operators include the tokens + and -. The binary + yields the sum of the expressions and the binary – yields the difference of the expressions. The associativity of additive operators is left-to-right.

SYNTAX: *expression* + *expression*
expression - *expression*

APPLICABILITY: int
string (only addition—performs concatenation)

3.3.5 Relational and Equality Operators

The relational and equality operators include the tokens <, >, <=, >=, ==, and !=. Both sets of operators yield 0 if the binary relation is false and 1 if it is true. The associativity of relational and equality operators is left-to-right or right-to-left.

SYNTAX: *expression* < *expression*
expression > *expression*
expression <= *expression*
expression >= *expression*
expression == *expression*
expression != *expression*

APPLICABILITY: int
boolean (only == and !=),
string (only == and !=),
piece (only == and !=)

3.3.6 Unary '!' Operator

The unary operator includes the ! token. It produces the negation of the adjacent expression. The associativity of the unary operator is right-to-left.

SYNTAX: ! *expression*

APPLICABILITY: boolean

3.3.7 Logical Operators

The Boolean AND operator includes the token &&. It returns 1 if both expressions surrounding the token are non-zero. It returns 0 is both expressions are 0. The associativity of the Boolean AND is left-to-right.

The Boolean OR operator includes the token ||. It returns 1 if either expression surrounding the token are non-zero. It returns 0 is both expressions are 0. The associativity of the Boolean OR is left-to-right. If the first expression is non-zero, the second expression is not evaluated.

SYNTAX: *expression* && *expression*
expression || *expression*

APPLICABILITY: boolean

3.3.8 Assignment Operator
The assignment operator includes the token =. It requires that an l_value be its left operand and that the type of the assignment expression is the same as the left operand. After assignment, the resulting value is stored in the left operand. The associativity of the assignment operator is right-to-left.

SYNTAX: *identifier* = *expression*
APPLICABILITY: all types, literals

3.4 Statements
All statements in BGGL end with the ';' token.

3.4.1 Expression Statements
A majority of statements in BGGL are expression statements, of the following form:

```
expression ;
```

3.4.2 Conditional Statements
There are two forms of BGGL's conditional statement. In both cases the expression is evaluated and if it is non-zero, the first sub-statement is executed. In the second case the second sub-statement is executed if the expression is 0. As per convention, ambiguity regarding the else is resolved by connecting an else with the last-encountered "elseless if."

SYNTAX:
```
if ( expression ) {
       statement  * ;
}

if ( expression ) {
       statement * ;
} else {
       statement  * ;
}
```

3.4.3 Iterative Statements
BGGL supports while and for loops.

The while statement will execute the commands and statements inside the open and closed brackets while the expression contained in parenthesis evaluates to a non-zero number using BGGL's Boolean support. This expression is checked when the statement is first encountered in the program's sequence, and at each time execution reaches the final close-bracket. The loop will continue to repeat until the expression evaluates to 0, at which time the execution will bypass the block and proceed sequentially through the rest of the program.

SYNTAX:
```
while ( expression ) {
       statement  * ;
```

```
}
```

The for statement in BGGL does not follow most for-loop conventions.

SYNTAX:
```
for ( identifier = expression to expression ) {
      statement  * ;
}
```

EXAMPLE:
```
for (x = 6 to 7) {
      print x;
}
```

3.4.4 Return Statements

A function returns to the invocation token via a return. It may have either of the forms described below. In the first case nothing is returned. In the second case, expression is returned to the caller of the function.

SYNTAX:
```
return ;
return expression ;
```

3.4.5 Declaration Statements

An identifier is associated with any of BGGL's eight data types through a declaration statement. Declaration statements initialize identifiers as variables. The syntax for the two types of declaration statements is as follows:

SYNTAX:
```
type identifier;
type identifier = expression;
```

3.4.6 Assignment Statements

A variable, named with an identifier, is given a value through an assignment statement. The assignment may occur when the identifier is given its type in the declaration statement, or it may occur separately. The syntax for the latter situation is outlined below:

SYNTAX:
```
identifier = expression;
```

3.5 Functions

3.5.1 Overview

Functions in BGGL are evaluated in applicative order. Function declarations may not be nested. Function declarations must take place before the game block of a BGGL program. No function declarations may take place during or after the game block. The BGGL compiler implements static scoping.

3.5.2 Declaration
Functions in BGGL are defined using the following syntax:

SYNTAX:
```
func identifier ( parameter list ) returns type

func identifier ( parameter list ) returns nothing
```

where parameter list = type *identifier* *
(comma- separated)

3.5.3 Invocation
Functions in BGGL may be called anywhere in the code where an expression would be suitable. Invoking a function is an expression. It may be paired with a declaration or assignment statement to give an appropriately-typed value. The syntax for invocation expressions is below:

SYNTAX:
```
function-identifier ( expression list );
```

where expression list = *expression* *
(comma-separated)

3.5.4 Internal Functions
The following internal functions interact with the console's input and output streams exhibiting expected behavior. The print function simply takes an expression and prints it to the screen, and each input function takes the specified expression, prints it the screen, and then waits for the user to enter a value of the type specified in the internal function declaration (string or int).

SYNTAX:
```
print expression;
identifier = input outputexpression , int;
identifier = input outputexpression , string;
```

where outputexpression = string literal | *identifier*

3.6 BGGL-Specific Language Constructs

3.6.1 Program Layout
All BGGL files must contain a game block. A game block is defined as the BGGL-keyword 'game' followed by curly braces ' { , } ' which enclose the game's code.

A BGGL file can contain function and rule declarations which may only be defined before the game block in a *.bggl file's order.

3.6.2 Board Coordinate Syntax

The global board variable may be initialized at any location in BGGL code. It calls for an array of piece arrays. The BGGL compiler will set up the dimensions of the board based on the quantity and length of the arrays in the board declaration. An empty board with dimensions *r* * *c* can be specified by providing *r* arrays each filled with *c* 'empty pieces.' An empty piece in BGGL is specified using the underscore '_' character. An invalid piece in BGGL is specified using the hash '#' character. Invalid pieces are used to chisel away at the rectangular matrix to create custom boards.

In the example that follows, a 2 by 2 board is created whose lower-left and upper-right regions are invalid. This board would have two valid squares aligned to look like a small backslash.

SYNTAX: `board = < [ pieces list ] * > ;`

where pieces list = *piece* *
(comma-separated)

EXAMPLE:
```
board = <   [ _ , # ]
            [ # , _ ] > ;
```

The piece at a specific coordinate on the board can be identified using the expression below. The following expression returns a value of type piece:

SYNTAX: `< row coordinate , column coordinate >`

where row coordinate and column coordinate = 0-based board references of type int.

EXAMPLE: < 2 , inputcol >

An array of pieces at four different locations on the board can be identified all at once using the expressions below. An entire row is returned using the underscore '_' followed by the 0-based target row. A entire column is returned using the vertical bar '|' followed by the 0-based target column. The forwards and backwards diagonals in a square board are returned using the forward slash ' / 'and backslash ' \ ' followed by the 0-based target diagonal, respectively. The expressions below return arrays of pieces:

SYNTAX:
```
< _ row coordinate > | < column coordinate >
< / diagonal coordinate > | < \ diagonal coordinate >
```

where row coordinate , column coordinate , diagonal coordinate = 0-based board references of type int.

EXAMPLE: < | 2 >

The two built-in keywords height and width can be used anywhere in the code to return these two dimensions of the board. If a board has not been formally initialized yet, height and width will return 0.

3.6.3 Move Syntax

Pieces in BGGL move about the board coordinate system via move instructions. There are three types of allowable movements: add, remove, and move. To create a move instruction follow the syntax outlined below. Essentially, a move is a 4- or 6- tuple which takes in a move-type (+, -, or ^ for add, remove, move, respectively), a piece, and coordinates.

SYNTAX:

```
move identifier = : move type : piece identifier : t-row : t-col ;
move identifier = : ^ : piece identifier : s-row : s-col : t-row :
                    t-col ;
```

where move type = '+' or '-' for adding/deleting pieces, respectively
where piece identifier = an identifier of an already-initialized piece
where s-row and s-col are the 'source' coordinates (integers) and
where t-row and t-col are the 'target' coordinates (integers)

EXAMPLES:

: + : X : 0 : 0 ; (adds piece X to 0,0)
: - : O : row : col ; (removes piece O from row, col)
: ^ : curr_piece : 7 : 1 : row : col ;
(moves piece curr_piece from 7,1 to row, col)

In order to apply a move to the board, the following syntax is used:

SYNTAX:

```
apply move identifier ;
apply move declaration ;
```

where *move identifier* = the name of a previously initialized move
where *move declaration* = the full move syntax specified above

3.6.4 Rule Syntax

Rules in BGGL are simply specialized functions. Rules are named in the same manner as functions, by providing an identifier before the parentheses, and they can accept a list of parameters. Additionally, a list of piece identifiers should be provided to specify the pieces which the current rule applies to. The syntax for these structures is outlined below. They are specialized functions in that they may only return booleans.

SYNTAX:

```
rule identifier ( parameter list ) : piece list { }
       (implicitly returns boolean)
```

where parameter list = type *identifier* *
(comma- separated)

where piece list = *identifier* *
(comma- separated)

EXAMPLE: rule no_overwrite ( ) : X, O { ... return test syntax ; ... }

A special feature of the Rule syntax is that the programmer has access to the move which the rule is being applied to. This gives a programmer the ability to write custom rules if the 'test' predicate doesn't apply. The move variable names are thispiece, movetype, r1, c1, r2, and c2. For example, from within the rule block, a programmer can call (<r2,c2> == W), testing whether or not the destination square has piece W.

3.6.5 Test Syntax

Rules in BGGL primarily concern themselves with piece movement. Therefore, within each rule block, there are a series of test statements which very concretely define allowable piece movements. The syntax for such test statements is outlined below:

SYNTAX: `test length , direction , jump , emptysquare ;`

where length is an integer which specifies how many valid squares a piece a may move.

where direction is one of three keywords (diag | ortho | orthodiag) or an array of custom movement values.

where jump is a boolean which specifies if a piece must jump another piece to make a move (true) or not jump another piece to make a move (false).

where *emptysquare* is a boolean which specifies if a piece must land on an empty square at the end of its move.

EXAMPLE: test 1 , diag, false , false ; (code for allowable pawn-capture move)

Chapter 4: Project Plan

4.1 Responsibilities

Roughly, the breakdown of responsibilities for this project was as follows: Vitaliy assumed the role of group manager and did extensive work on with the grammar and walker of BGGL. Matt organized the file structure, set up our directories, test suites, and spent a great deal of time working with the interpreter and walker. Hrishikesh devoted time to the BGGL-specific conventions, the meat of our specialization. Steve compiled the documentation portions of the project, including the white paper, LRM, this final report and the PowerPoint presentation for the group. He was involved in testing in the later phases of the project as well.

Vitaliy	Group leader, grammar, walker
Matt	Configured directory structure, test suites, interpreter and walker
Hrishikesh	BGGL-specific conventions
Steve	Documentation, grammar, semantic testing

Our group met weekly on Mondays at 7:30 p.m. in the CLIC laboratory to discuss the project. Each of us used our personal computers for the development of BGGL.

4.2 Our Code

BGGL is a strongly-typed, applicatively ordered, top-down parsing, statically scoped programming language. In this section you'll find the specifics of our code's implementation.

Inheritance

By using inheritance in our testing framework we minimize the effort required to write new tests because all behavior is specified by parent classes (such as pretest, location, appropriate components of compiler.)

The type system we used implements inheritance extensively, specifically as follows: the abstract parent of all BGGL data types is ADataType. It defines abstract methods for all the common operations such as add(), sub(), div(), eq(), neq(), etc. ADataType declares that they all throw an OperationNotSupported exception.

The motivation of such a design was to ensure that many of the operations are automatically type-safe. For example, if (`string * string`) is not allowed, then it simply does not implement the mult() method. It also simplified the walking code by allowing us to call the correct method without having to check the types.

All concrete BGGL types (BGGLInt, BGGLBoolean, etc) extend directly from ADataType except for BGGLRule.

Package Structure
edu.bggl is the root package for our project.
edu.bggl.exception is where we keep all of our BGGL custom exceptions.
edu.bggl.grammar contains the Java classes generated by ANTLR.
edu.bggl.interpreter is the "executable" package: the interpreter and the demo are stored here.
edu.bggl.semantic stores all the BGGL datatypes and the symbol table (and its helper classes, Variable or Scope).

Symbol Table Implementation
The symbol table is simply a queue of Scope variables. It provides the obvious methods: add, get, and set. Each of these throws an IdentifierNotFoundException, useful for static semantic analysis. For convenience, we also define addNoCheck( ), getNoCheck( ), setNoCheck( ). These methods are useful because we have several implicit variables, and because we guarantee their existence in the symbol table, there is no need to check for the exception.

Static Scoping
A new scope is created each time the walker encounters a control flow statement (if blocks), iterative statements (for or while blocks), function calls, rule calls, or unassociated curly braces.

Global Scoping
We implement a global scope in the following way: all variables (with the exception of functions and rules) are added to the global scope if they are defined outside the game scope. If an identifier is not found in the symbol table, it is checked for in the global scope before throwing an IdNotFoundEx.

Error Handling: We disabled ANTLR's default exception handling so we could use our own. Whenever there's an exception at static semantic analysis or at runtime, our exception handler catches it and rethrows it as an ANTLR exception in order to facilitate debugging. We also execute the entire program inside a try-catch block.

4.3 Development Environment

4.3.1 ANTLR 2.0
ANTLR, short for ANother Tool for Language Recognition is a framework that allowed us to pretty painlessly construct a lexer, parser, and AST walker. ANTLR took our code and almost tripled it to create approximately 4,500 lines of Java to completely describe how to build and walk an abstract syntax tree constructed from the BGGL language.

4.3.2 Java 1.5
The vast majority of the code written by our group was in Java. Sun Microsystems describes Java as, "Java: A simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language." Java was an ideal language for the creation of BGGL in that it supports heavy object-orientation and inheritance. Java 5.0 brought immense improvement over Java 1.4 for BGGL, in its implementation of generics.

4.3.3 Eclipse
Eclipse is an open-source IDE designed specifically for the Java platform. As we mentioned above, the SVN plugin (Subclipse) allowed us to seamlessly edit the source tree, commit our changes, and integrate other member's changes into our working copy. Similarly, the ANTLR plugin for Eclipse provided us with the means not only to compile our grammars within Eclipse with a single click (or every time we saved), but with an integrated means to see and respond to errors in our grammar's code. We used several ANT scripts to achieve various combinations of code-generation.

4.3.4 Subversion (SVN)
Matt set up Subversion, which is a replacement to CVS as a source code revision control system. Many of CVS's ailments have been fixed in Subversion, as the new system tracks changes in the file tree as opposed to individual files. This tool allowed us to concurrently work on our source tree and easily transmit changes to the entire group. SVN has a great plug-in for Eclipse which Matt helped us set up and use in both Windows and Mac OSX environments.

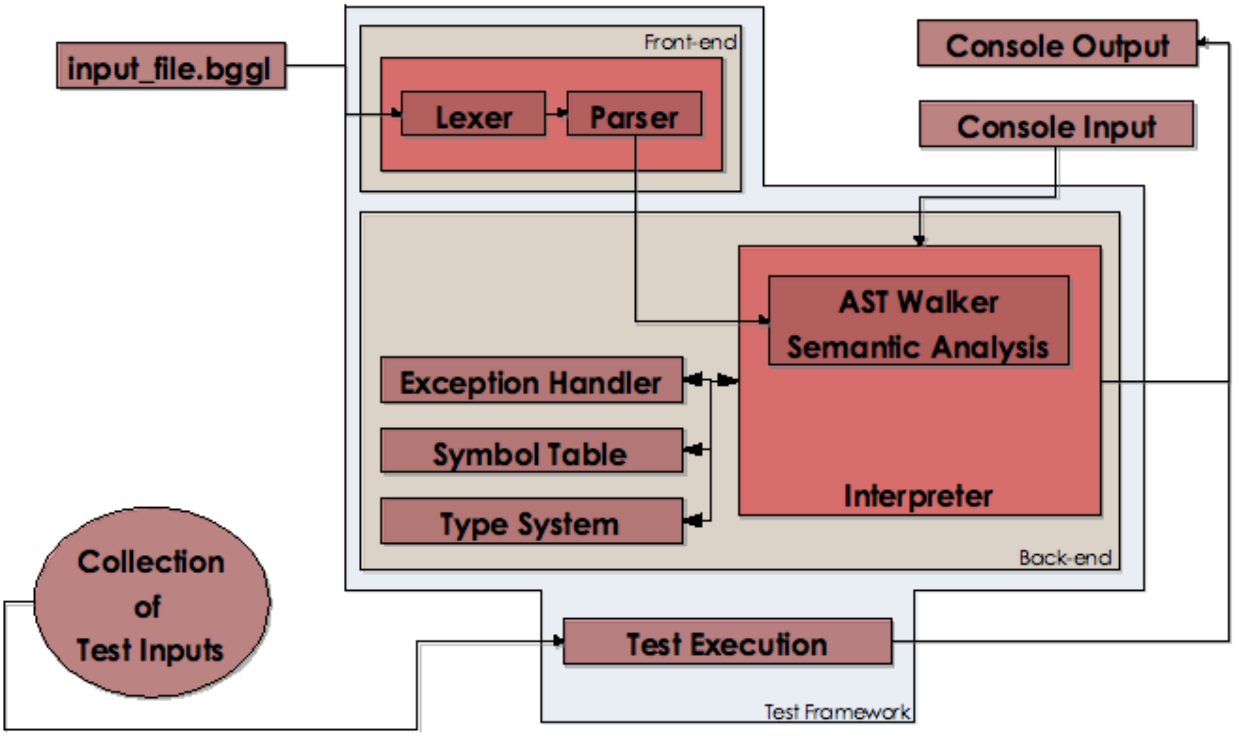
4.3.5 Junit 4.1
The JUnit 4.1 regression testing framework allowed us to make concise lexical, semantic, and output tests to be run through various components of our compiler. We specified an elaborate directory structure and ran all our tests on basically every generation of our code. The JUnit tests were essential to checking our language as they provided automated, continuous testing directly within the Eclipse IDE.

4.4 Project Timeline: Important Dates

Mon., 9/11	BGGL Conceived, Wiki Configured, Whitepaper Preparation
Mon., 9/18	SVN Support and IDE Configuration on all 4 laptops
Mon., 10/16	Preliminary LRM Finalized, Directory Structure Finalized
Mon., 11/20	Parser, Lexer, and Walker Almost Done
Mon., 12/4	Tic-Tac-Toe Created, Test Suite Begins Growing
Mon., 12/11 – Fri., 12/15	Continued Testing, Scoping Issues Resolved,
Sat., 12/16 – Mon., 12/18	Chinese Checkers Coded, Debugged, Final Testing

Chapter 5: Architecture

Overview
The architecture of BGGL's compiler is depicted below:

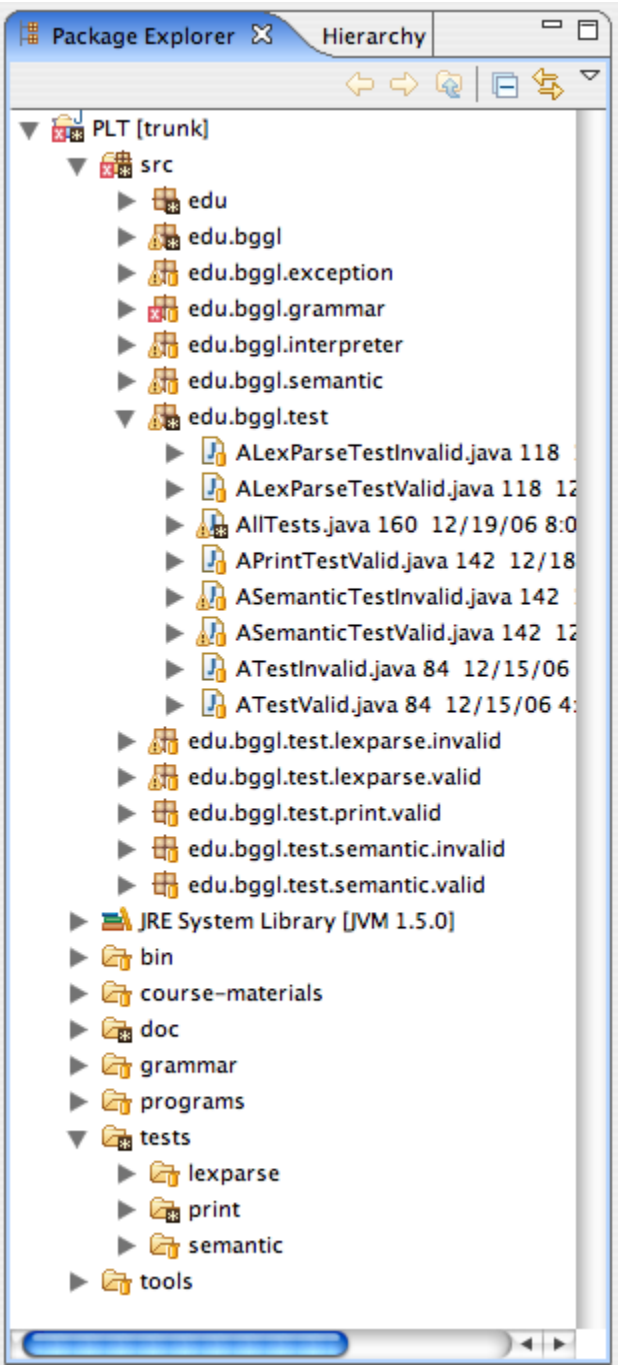


Our architecture is essentially that of a standard compiler wrapped within a flexible, easy-to-use unit testing framework.

The dataflow is as follows: the BGGL program is input to the lexer and output as a stream of tokens which is then fed into the parser and output as an AST. We choose to wrap the walker into the Interpreter. In order to perform static semantic analysis, we actually invoke the interpretation with a flag set so that interpretation does not happen. If static semantic analysis passes, we invoke the Interpreter again, this time with the interpretation flag on.

All components of the system have direct access to the symbol table via the Singleton design pattern, and the exception handler is globally accessible as a public static method as well.

Chapter 6: Project Tests



6.1 Test Suites

The way we organized our testing process is described below. Each test class represented one test BGGL program. All test classes reside in the edu.bggl.test package. The hierarchy is as follows:

AValidTest: The abstract parent of all tests classes that are meant to pass.
AInvalidTest: The abstract parent of all tests classes that are meant to fail.

Each of these parent classes provides so-called "tear-up" code which initializes the environment before the start of each test. Our initialization process was to: 1) reset the symbol table, 2) …

For a valid test to be asserted as passing, it had to have thrown no exceptions. For an invalid test, it HAD to throw an exception.

AValidLexParseTest, AInvalidLexParseTest: The abstract parents of all test classes that check syntax.

For a lexparse test to pass, it must throw no lex-parse exceptions. For it to fail, it must throw an exception.

Most of our testing at this level was valid testing; obviously there are infinitely many programs which are syntactically invalid, so there was no attempt at coverage. However, every piece of syntax has at least 1 valid test to prove that the grammar does indeed recognize it.

For more complicated syntactic constructs, multiple tests were written. Note that because this is only syntactic testing, many of the programs were (purposefully) written to be semantically invalid.

AValidSemanticTest, AInvalidSemanticTest: Performed static semantic analysis. Also performs syntax checking, so all test programs must be syntactically valid. Must throw for the test to correctly fail if it was an invalid test.

We currently perform the following static semantic analyses:

- return value type checking
- assignment type checking
- arg number
- arg types
- board dimension
- @ operator type
- board location operator type
- board subset operator type

At this level we basically only had invalid tests. Clearly the number of possible errors are intractable: for example, simply for an assignment, given that we have [N = however many we have] different types, we would need N^2 number of tests to check all combinations. Our strategy was simply to select a small random subset of permutations and write tests for them.

AValidPrintTest, and **AinvalidPrintTest:** These tests were our "highest" level of testing. This level tested actual functionality through printing. We designed the Interpreter in such a way that we could redirect the output of the program into an internal buffer for later comparison. For each test, we manually hard-coded the expected output into each class. The test would be run with the output redirected, and then compared against this correct output.

These tests were very useful in debugging during development. It was only through such tests that we found and fixed many bugs involving some of the more advanced features such as scoping on control constructs. The utility of this level of testing can not be overstated.

6.2 Example: Chinese Checkers

```
/*
 * ChineseCheckers.bggl
 */

piece G;
piece B;
piece Y;
piece P;
piece O;
piece R;

player p1;
player p2;

rule movement(player thisplayer):R,G
{
      return test 2,[[0,2],[-1,1],[-1,-1],[0,-2],[1,-
1],[1,1]],true,true;

}
```

```
func gameover() returns player {
      player winner;
      if (<_0> == [#,#,#,#,#,#,#,#,#,#,#,#,#,R,#,#,#,#,#,#,#,#,#,#,#,#,#]
       && <_1> == [#,#,#,#,#,#,#,#,#,#,#,#,#,R,#,R,#,#,#,#,#,#,#,#,#,#,#]
       && <_1> == [#,#,#,#,#,#,#,#,#,#,#,#,R,#,R,#,R,#,#,#,#,#,#,#,#,#,#]
       && <_1> == [#,#,#,#,#,#,#,#,#,#,#,R,#,R,#,R,#,R,#,#,#,#,#,#,#,#,#])
      {
            winner = p1;
      }
      else
      {
            if (<_12> ==
[#,#,#,#,#,#,#,#,#,#,G,#,G,#,G,#,G,#,#,#,#,#,#,#,#,#]
             && <_13> ==
[#,#,#,#,#,#,#,#,#,#,#,G,#,G,#,G,#,#,#,#,#,#,#,#,#,#]
             && <_13> ==
[#,#,#,#,#,#,#,#,#,#,#,#,G,#,G,#,#,#,#,#,#,#,#,#,#,#]
             && <_14> ==
[#,#,#,#,#,#,#,#,#,#,#,#,#,G,#,#,#,#,#,#,#,#,#,#,#,#])
            {
                  winner = p2;
            }
      }
      return winner;
}

func getpiece(player p) returns piece
{
      if (p == p1) {
            return R;
      }
      else {
            return G;
      }
}

func nextplayer(player p) returns player
{
      if (p == p1) {
            return p2;
      }
      else {
            return p1;
      }
}
```

```
game {
      board = <
      [#,#,#,#,#,#,#,#,#,#,#,#,G,#,#,#,#,#,#,#,#,#,#,#,#]
      [#,#,#,#,#,#,#,#,#,#,#,#,G,#,G,#,#,#,#,#,#,#,#,#,#]
      [#,#,#,#,#,#,#,#,#,#,#,#,G,#,G,#,G,#,#,#,#,#,#,#,#]
      [#,#,#,#,#,#,#,#,#,#,#,#,G,#,G,#,G,#,G,#,#,#,#,#,#]
      [B,#,B,#,B,#,B,#,_,#,_,#,_,#,_,#,_,#,Y,#,Y,#,Y,#,Y]
      [#,B,#,B,#,B,#,_,#,_,#,_,#,_,#,_,#,_,#,Y,#,Y,#,Y,#]
      [#,#,B,#,B,#,_,#,_,#,_,#,_,#,_,#,_,#,_,#,Y,#,Y,#,#]
      [#,#,#,B,#,_,#,_,#,_,#,_,#,_,#,_,#,_,#,_,#,Y,#,#,#]
      [#,#,#,#,_,#,_,#,_,#,_,#,_,#,_,#,_,#,_,#,_,#,#,#,#]
      [#,#,#,P,#,_,#,_,#,_,#,_,#,_,#,_,#,_,#,_,#,O,#,#,#]
      [#,#,P,#,P,#,_,#,_,#,_,#,_,#,_,#,_,#,_,#,O,#,O,#,#]
      [#,P,#,P,#,P,#,_,#,_,#,_,#,_,#,_,#,_,#,O,#,O,#,O,#]
      [P,#,P,#,P,#,P,#,_,#,_,#,_,#,_,#,_,#,O,#,O,#,O,#,O]
      [#,#,#,#,#,#,#,#,#,#,R,#,R,#,R,#,R,#,#,#,#,#,#,#,#]
      [#,#,#,#,#,#,#,#,#,#,#,R,#,R,#,R,#,#,#,#,#,#,#,#,#]
      [#,#,#,#,#,#,#,#,#,#,#,#,R,#,R,#,#,#,#,#,#,#,#,#,#]
      [#,#,#,#,#,#,#,#,#,#,#,#,#,R,#,#,#,#,#,#,#,#,#,#,#]
      >;

      print "Welcome to Chinese Checkers";
      print "Player " + p1 + " = R, Player " + p2 + " = G";
      print "";

      boolean done = false;
      int row1;
      int col1;
      int row2;
      int col2;
      int start_row;
      int start_col;
      move m;
      player thisplayer = p1;
      piece thispiece;
      string promptgoagain;
      int next_r1;
      int next_c1;
      boolean goagain = false;
      player winner;
      boolean exit = false;

      while (! done && ! exit) {
            print board;
            print "Player " + thisplayer + ": " + getpiece(thisplayer);

            if (goagain) {
                  row1 = next_r1;
```

```
                col1 = next_c1;
          }
          else {
                row1 = input "Enter source row: ", int;
                col1 = input "Enter source col: ", int;

                start_row = row1;
                start_col = col1;
          }
          row2 = input "Enter destination row: ", int;
          col2 = input "Enter destination col: ", int;

          piece thispiece = getpiece(thisplayer);
          m = :^:thispiece:row1:col1:row2:col2;

          if (movement(thisplayer):m) {
                apply m;
                goagain = false;

                promptgoagain = input "Do you want to go again?
(y/n/exit) ", string;
                while (promptgoagain != "y" && promptgoagain == "Y" &&
                      promptgoagain != "n" && promptgoagain == "N" &&
                      promptgoagain != "exit")
                {
                      promptgoagain = input "Do you want to go again?
(y/n/exit) ", string;
                }
                print "";

                if (promptgoagain == "y" || promptgoagain == "Y") {
                      goagain = true;
                      next_r1 = row2;
                      next_c1 = col2;
                }
                else {
                      if (promptgoagain == "n" || promptgoagain == "N")
{
                            if (row2 == start_row && col2 == start_col)
{
                                  print "";
                                  print "===You are back where you
started. Please make a move.===";
                                  print "";
                            }
                            else {
                                  thisplayer = nextplayer(thisplayer);
                            }
```

```
                    }
                    else {
                        print "";
                        print "Exiting. Thanks for playing!";
                        exit = true;
                    }
                }
            }
            else {
                print "";
                print "===Invalid move. You must use your piece to
jump over a piece on an empty square===";
                print "";
            }

            if (! exit) {
                winner = gameover();
                if (winner == p1 || winner == p2) {
                    print "GAME OVER";
                    print "" + winner + " (" + getpiece(winner) + ")
won!";
                    done = true;
                }
            }
        }

        print "That was fun.";
}
```

Chapter 7: Lessons Learned

Despite repeated admonitions from Professor Edwards and our peers about starting early and the importance of planning, the first lesson-learned for our group would be that 'going through the motions' for two months is just not sufficient. While we met every week, the most intense brainstorming sessions occurred at the end of the project.

By this point, most of the foundation for the project had already been laid. As a result, certain new ideas (including thoughts on both syntactic and semantic aspects of the code) were considerably more difficult to implement, and at some points, were difficult to even conceive how to implement. Specifically, we had to rethink the turn { } block's implementation and move to a less flexible model.

Had we done better planning at the beginning we could have really solidified the language in an orderly manner. Some of the solutions to problems we encountered at the end of the project were, for lack of a better word, hacks (hard-wired, nasty code).

Scoping is tough. It should have been a higher priority in our language's evolution.

Wikis are great, e-mails are fantastic, but cell phones are the best. In terms of communication, it was important that during the last month of the project our group realized that the easiest, most reliable means of communication was over cell-phone.

Chapter 8: Complete Code Listing

bggl.g

```
header
{
    package edu.bggl.grammar;
}

class BGGLParser extends Parser;

options
{
    // We want ANTLR to pass on exceptions (for JUnit testing)
    defaultErrorHandler = false;

    exportVocab = BGGL;
    buildAST = true;
    k=2;
}

// IDENTIFIER; only keywords should go here
tokens
{
    GAME = "game";
    TURN  = "turn";
    PRINT = "print";
    IF = "if";
    ELSE = "else";
    WHILE = "while";
    FOR = "for";
      TO = "to";
      FUNC = "func";
      RULE = "rule";
      INPUT = "input";
      APPLY = "apply";
      RETURNS = "returns";
      WIDTH = "width";
      HEIGHT = "height";
      RETURN = "return";
      BREAK = "break";

      TYPE_PIECE = "piece";
      TYPE_PLAYER = "player";
    TYPE_BOOLEAN = "boolean";
    TYPE_ARRAY = "array";
      TYPE_MOVE = "move";
    TYPE_INT = "int";
    TYPE_STRING = "string";
```

```
    TRUE = "true";
    FALSE = "false";

      NOTHING = "nothing";

      TEST = "test";
      ORTHO = "ortho";
      DIAG = "diag";
      ORTHODIAG = "orthodiag";

      // "pseudo" tokens for AST node construction
    BLOCK;
    ARRAY;
    BOARD;
    MOVE;

    PARAMETER_LIST;
    PROGRAM_START;
    FUNCTION_DEF;
    CALL;
    EXPRESSION_LIST;
    BOARD_LOC;
    BOARD_SUBSET;
    PIECES_LIST;
    MOVE_LIST;
    DIRECTION;
    JUMP_EMPTY;
    MOVE_LENGTH;
}

main_program:
      (function_def | rule_def | declaration SEMICOLON!)*
      game_block
            {
                  #main_program =
                        #([PROGRAM_START, "program_start"], main_program);
            }
    ;

function_def:
      FUNC^ IDENTIFIER LPAREN! parameter_list RPAREN!
            RETURNS! (base_datatype | NOTHING)
      block
      ;

rule_def:
      RULE^ IDENTIFIER LPAREN! parameter_list RPAREN!
            pieces_list
      block
      ;
```

```
// You must name at least one Piece
pieces_list:
      COLON! IDENTIFIER (COMMA! IDENTIFIER)*
      {
            #pieces_list = #([PIECES_LIST, "pieces_list"], pieces_list);
      }
      ;

parameter_list:
      (declaration (COMMA! declaration)* )?
      {
            #parameter_list = #([PARAMETER_LIST, "parameter_list"],
parameter_list);
      }
      ;

block:
      LBRACE! (statement)* RBRACE!
      { #block = #([BLOCK, "block"], block); }
      ;

game_block:
      GAME^ LBRACE! (statement)* RBRACE!
      ;

turn_block:
    TURN^ IDENTIFIER LBRACE! (statement)* RBRACE!
    ;

statement:
    (print
    | input
    | declaration
    | assignment
    | return_stmt
    | call
    | BREAK
    | apply
    | // epsilon
    ) SEMICOLON!
    // no semicolon after * blocks
    | controlBlock
    | turn_block
    | block
    ;

test:
      TEST^
            move_length COMMA!
            direction COMMA!
            jump_empty COMMA!
            jump_empty
```

```
	;

move_length:
	(IDENTIFIER | INTEGER)?
	{
		#move_length = #([MOVE_LENGTH, "move_length"], #move_length);
	}
	;

direction:
	(array | IDENTIFIER | ORTHO | DIAG | ORTHODIAG)?
	{
		#direction = #([DIRECTION, "direction"], #direction);
	}
	;

jump_empty:
	(bggl_boolean | IDENTIFIER)?
	{
		#jump_empty = #([JUMP_EMPTY, "jump_empty"], #jump_empty);
	}
	;

return_stmt:
	RETURN^ (expression)?
	;

print:
	PRINT^ expression
	;

input:
	INPUT^ (STRING | IDENTIFIER) COMMA! base_datatype
	;

input_type:
	(COMMA! base_datatype)?
	;

apply:
	APPLY^ ( move | IDENTIFIER)
	;

declaration:
	(TYPE_INT^ | TYPE_BOOLEAN^ | TYPE_STRING^ |
		TYPE_ARRAY^ | TYPE_PIECE^ | TYPE_PLAYER^ | TYPE_MOVE^)
		IDENTIFIER (ASSIGN expression)?
	;

assignment:
	IDENTIFIER ASSIGN^ expression
	;
```

```
controlBlock:
      ifStatement
      | whileStatement
      | forStatement
      ;

ifStatement:
      IF^ LPAREN! expression RPAREN! block
      (ELSE! block)?
      ;

whileStatement:
      WHILE^ LPAREN! expression RPAREN! block
      ;

forStatement:
      FOR^
      LPAREN! IDENTIFIER ASSIGN! expression TO! expression RPAREN!
      block
      ;

piece:
      (IDENTIFIER | INVALID | UNDERSCORE)
      ;

array:
       LBRACKET! (expression (COMMA! expression)*)? RBRACKET!
       {#array = #([ARRAY, "array"], array); }
      ;

board:
       LT! (array)+ GT!
       {#board = #([BOARD, "board"], board); }
      ;

board_dim:
      (WIDTH | HEIGHT)
      ;

// Expression list in increasing precedence order
expression:
      expression_logic
      ;

expression_logic:
      expression_logic_not
            ((AND^ | OR^) expression_logic_not)*
      ;

expression_logic_not:
      (NOT^)? expression_equality
```

```
      ;

expression_equality:
      expression_add_sub
            ((EQ^ | NEQ^ | GE^ | LE^ | GT^ | LT^) expression_add_sub)*
      ;

expression_add_sub:
      expression_mult_div
            ((PLUS^ | MINUS^) expression_mult_div)*
      ;

expression_mult_div:
      expression_unary_negate
            ((MULT^  | DIV^ | MOD^) expression_unary_negate)*
      ;

expression_unary_negate:
      (MINUS^)? expression_index
      ;

//index of an array
expression_index:
      expression_base
            (AT^ expression_base)?;

expression_base:
      IDENTIFIER
      | bggl_boolean
      | INTEGER
      | STRING
      | array
      | board
      | UNDERSCORE
      | INVALID
      | LPAREN! expression RPAREN!
      | call
      | move
      | board_loc
      | board_subset
      | board_dim
      | test
      | input
      ;

// Either a function call or a rule call
call:
      IDENTIFIER LPAREN! expression_list RPAREN! move_list
      {
            #call = #([CALL, "call"], call);
      }
      ;
```

```
expression_list:
      (expression (COMMA! expression)*)?
      {
            #expression_list = #([EXPRESSION_LIST, "expr_list"],
expression_list);
      }
      ;

// It's called a list, but you can only specify one
move_list:
      (COLON! IDENTIFIER)?
      {
            #move_list = #([MOVE_LIST, "move_list"], move_list);
      }
      ;

board_subset:
       LT! (UNDERSCORE | COL | DIV | BACKSLASH) (IDENTIFIER|INTEGER) GT!
       {#board_subset = #([BOARD_SUBSET, "board_subset"], board_subset); }
      ;


board_loc:
      LT! (IDENTIFIER | INTEGER) COMMA! (IDENTIFIER | INTEGER) GT!
      { #board_loc = #([BOARD_LOC, "board_loc"], board_loc); }
      ;

//a move is represented as :+:5:4:3:1;
move:
      COLON! move_type COLON! IDENTIFIER COLON!
      (IDENTIFIER | INTEGER) COLON! (IDENTIFIER | INTEGER)
      (COLON! (IDENTIFIER | INTEGER) COLON! (IDENTIFIER | INTEGER))?
      { #move = #([MOVE, "move"], move); }
      ;

move_type:
      (PLUS|MINUS|EXP)
      ;

base_datatype:
      TYPE_INT
      | TYPE_BOOLEAN
      | TYPE_STRING
      | TYPE_BOARD
      | TYPE_ARRAY
      | TYPE_PIECE
      | TYPE_PLAYER
      ;

bggl_boolean:
      TRUE | FALSE
```

```
      ;

class BGGLLexer extends Lexer;

options
{
    // (?) By default, don't check tokens against keywords
    testLiterals = false;

    // (?) Need to decide when string literals end
    k = 2;

    // Accept all 8-bit ASCII characters
    charVocabulary = '\3'..'\377';

    // We want ANTLR to pass on exceptions (for JUnit testing)
    defaultErrorHandler = false;

    exportVocab = BGGL;
}

// Identifier rules:
IDENTIFIER
    options
    {
        testLiterals = true;
    }:
    LETTER (LETTER | UNDERSCORE | DIGIT)*;

UNDERSCORE: "_";
INVALID: "#";

LBRACKET: "[";
RBRACKET: "]";

LBRACE: "{";
RBRACE: "}";
LPAREN: "(";
RPAREN: ")";

ASSIGN: '=';
EQ: "==";
NEQ: "!=";
GE: ">=";
LE: "<=";
GT: '>';
LT: '<';
OR: "||";
AND: "&&";
NOT: '!';
PLUS: '+';
MINUS: '-';
```

```
DIV: "/";
MULT: '*';
MOD: "%";
EXP: "^";
COLON: ':';
AT: '@';
SEMICOLON: ';';
COMMA: ',';
COL: '|';
BACKSLASH: '\\';

protected LETTER: ('a'..'z' | 'A'..'Z');

protected DIGIT: ('0'..'9');

INTEGER: (DIGIT)+;

STRING:
    "\""!
    ( "\"" "\""! | ~('\"') )*
    "\""!
    ;

COMMENT: (
                  "/*"  (
                                    options { greedy = false; } :
                                    ( '\n' | '\r' ) | ~( '\n' | '\r' )
                              )*
                  "*/"
            |
                  "//"  (
                                    ~( '\n' | '\r' )
                              )*
            )
                  { $setType(Token.SKIP); }
            ;

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

BGGLWalker.g

```
header
{
    package edu.bggl.grammar;

    import edu.bggl.semantic.*;
    import edu.bggl.interpreter.*;
    import edu.bggl.exception.*;
    import java.util.*;
}

class BGGLWalker extends TreeParser;

options
{
      defaultErrorHandler = false;
      importVocab = BGGL;
      buildAST = true;
      k=2;
}

{
      private Interpreter interpreter;
      private SymbolTable symbolTable = SymbolTable.getInstance();

      public BGGLWalker(Interpreter interpreter)
      {
            this();
            this.interpreter = interpreter;
      }

      public boolean terminate = false;
}

main_program
{
      Variable v = null;
}
: #(PROGRAM_START
      (function_def
      |
            {
                  // set global scope mode, so that all declarations go there
                  symbolTable.setGlobalMode(true);
            }
            v = declaration_assignment
            {
                  // clear global mode
                  symbolTable.setGlobalMode(false);
            }
      | rule_def
```

```
      )*
      game_block
      )
      ;

function_def
{
      List<Variable> parameters;
      ADataType returnType = null;
      ADataType e = null;
}
: #(FUNC
            fId:IDENTIFIER
            parameters = parameter_list
            returnType = return_type
            block:.
      )
      {
            symbolTable.add(
                  fId.getText(),
                  new BGGLFunction(
                        fId.getText(),
                        parameters,
                        returnType,
                        block));

            for (Variable v : parameters)
            {
                  symbolTable.remove(v);
            }
      }
      ;
exception
      catch [DuplicateIdentifierException ex]
      {
            ExceptionHandler.err(ex);
      }

rule_def
{
      List<Variable> parameters;
      List<String> pieces;
      ADataType e = null;
}
: #(RULE
            rId:IDENTIFIER
            parameters = parameter_list
            pieces = pieces_list
            block:.
      )
      {
            symbolTable.add(
```

```
                  rId.getText(),
                  new BGGLRule(
                        rId.getText(),
                        parameters,
                        pieces,
                        block));

            for (Variable v : parameters)
            {
                  symbolTable.remove(v);
            }
      }
      ;
exception
      catch [DuplicateIdentifierException ex]
      {
            ExceptionHandler.err(ex);
      }

parameter_list returns [List<Variable> parameters]
{
      parameters = new ArrayList<Variable>();
      Variable v = null;
}
: #(PARAMETER_LIST
      (v = declaration_assignment
            {
                  parameters.add(v);
            }
      )*
      )
      ;

return_type returns [ADataType dt]
: dt = base_datatype
| NOTHING
      { dt = null; }
      ;

pieces_list returns [List<String> pieces]
{
      pieces = new ArrayList<String>();
}
: #(PIECES_LIST
      (id:IDENTIFIER
            {
                  pieces.add(id.getText());
            }
      )*
      )
      ;
```

```
base_datatype returns [ADataType dt]
: TYPE_INT
      { dt = new BGGLInt(); }
| TYPE_BOOLEAN
      { dt = new BGGLBoolean(); }
| TYPE_STRING
      { dt = new BGGLString(); }
| TYPE_ARRAY
      { dt = new BGGLArray(); }
| TYPE_PIECE
      { dt = new BGGLPiece(); }
| TYPE_PLAYER
      { dt = new BGGLPlayer(); }
| TYPE_MOVE
      { dt = new BGGLMove(); }
      ;

move_type returns [int type]
: PLUS
      { type = 1; }
| MINUS
      { type = 2; }
| EXP
      { type = 3; }
;

subset_type returns [int type]
: UNDERSCORE
      { type = 1; }
| COL
      { type = 2; }
| DIV
      { type = 3; }
| BACKSLASH
      { type = 4; }
;

game_block
{
      ADataType dt;
}

: #(GAME (dt = expression)* )
;

expression returns [ADataType dt]
{
      ADataType l = null;
      ADataType r = null;
      ADataType e = null;
      ADataType x1=null, y1=null, x2=null, y2=null;
      int type;
```

```
      dt = null;
      List<ADataType> al;

      // Only for rule calls
      List<BGGLMove> moveList;
      // Only for test
      ADataType a = null;
      ADataType b = null;
      ADataType c = null;
      ADataType d = null;

      if (terminate)
            return null;

      Variable v = null;
}
: #(NOT l = expression)
      {
            dt = l.not();
      }
| #(EQ l = expression r = expression)
      {
            dt = l.eq(r);
      }
| #(NEQ l = expression r = expression)
      {
            dt = l.neq(r);
      }
| #(AND l = expression r = expression)
      {
            dt = l.and(r);
      }
| #(OR l = expression r = expression)
      {
            dt = l.or(r);
      }
| #(GE l = expression r = expression)
      {
            dt = l.ge(r);
      }
| #(LE l = expression r = expression)
      {
            dt = l.le(r);
      }
| #(GT l = expression r = expression)
      {
            dt = l.gt(r);
      }
| #(LT l = expression r = expression)
      {
            dt = l.lt(r);
      }
```

```
| #(PLUS l = expression r = expression)
      {
            dt = l.add(r);
      }
// This also handles unary negation
| #(MINUS l = expression (r = expression)?)
      {
            //dt = l.sub(r);
            if (r == null)
                  dt = l.negate();
            else
                  dt = l.sub(r);
      }
| #(MULT l = expression r = expression)
      {
            dt = l.mult(r);
      }
| #(DIV l = expression r = expression)
      {
            dt = l.div(r);
      }
| #(MOD l = expression r = expression)
      {
            dt = l.mod(r);
      }
| #(AT l = expression r = expression)
      {
            if (! (l instanceof BGGLArray))
                  throw new TypeMismatchException("BGGLArray", l.getType());
            if (! (r instanceof BGGLInt))
                  throw new TypeMismatchException("BGGLInt", r.getType());

            dt = ((BGGLArray)l).getElement(((BGGLInt)r).getValue());
      }
| #(ARRAY
            {
                  al = new ArrayList<ADataType>();
            }
            (
                  l = expression
                  { al.add(l); }
            )*
      )
      {
            dt = new BGGLArray(al);
      }
| #(BOARD
            {
                  ArrayList brd = new ArrayList<BGGLArray>();
            }
            (
                  l = expression
```

```
                {
                        brd.add((BGGLArray)l);
                }
            )*
            {
                dt = new BGGLBoard(brd);
            }
        )
    | #(MOVE
            type = move_type
            l = expression
            x1 = expression
            y1 = expression
            (x2 = expression y2=expression)?
        )
        {
            if (x2 != null && y2 != null) {
                dt = new BGGLMove(
                        type,
                        (BGGLPiece) l,
                        (BGGLInt) x1,
                        (BGGLInt) y1,
                        (BGGLInt) x2,
                        (BGGLInt) y2);
            }
            else {
                dt = new BGGLMove(
                        type,
                        (BGGLPiece) l,
                        (BGGLInt) x1,
                        (BGGLInt) y1);
            }
        }

    | #(BOARD_LOC l = expression r = expression)
        {
            if (! (l instanceof BGGLInt))
                throw new TypeMismatchException("BGGLInt", l.getType());
            if (! (r instanceof BGGLInt))
                throw new TypeMismatchException("BGGLInt", r.getType());

            // Hack. Needs access to global scope for predefined variables.
            // Note that you cannot change the board from within a function.
            int x_coord = ((BGGLInt)l).getValue();
            int y_coord = ((BGGLInt)r).getValue();
            dt =
                (
                    (BGGLBoard) symbolTable.get(BGGLBoard.ID_BOARD, false)
                            .getDataType()
                ).getPiece(x_coord, y_coord);
        }
    | #(BOARD_SUBSET type = subset_type l = expression)
```

```
      {
            if (! (l instanceof BGGLInt))
                  throw new TypeMismatchException("BGGLInt", l.getType());

            BGGLBoard bd =
                  ((BGGLBoard) symbolTable.get("board", false).getDataType());
            int x = ((BGGLInt) l).getValue();

            if (type == 1)
                  dt = bd.getRow(x);
            else if (type == 2)
                  dt = bd.getCol(x);
            else if (type == 3)
                  dt = bd.getForwardDiag();
            else if (type == 4)
                  dt = bd.getBackwardDiag();
            else
                  dt = null;
      }
| WIDTH
      {
            BGGLBoard bd =
                  ((BGGLBoard) symbolTable.get(BGGLBoard.ID_BOARD,
false).getDataType());
            dt = new BGGLInt(bd.getWidth());
      }
| HEIGHT
      {
            BGGLBoard bd =
                  ((BGGLBoard) symbolTable.get(BGGLBoard.ID_BOARD,
false).getDataType());
            dt = new BGGLInt(bd.getLength());
      }
| id:IDENTIFIER
      {
            if (id.getText().equals(BGGLBoard.ID_BOARD))
            {
                  dt = symbolTable.get(id.getText(), false).getDataType();
            }
            else
            {
                  // Guaranteed to exist by static semantic analysis
                  v = symbolTable.get(id.getText());
                  dt = v.getDataType();
            }
      }
//Special Variables, _ and #
| UNDERSCORE
      {
            dt = symbolTable.get("_").getDataType();
      }
| INVALID
```

```
      {
            dt = symbolTable.get("#").getDataType();
      }

| t:TRUE
      {
            dt = new BGGLBoolean(true);
      }
| f:FALSE
      {
            dt = new BGGLBoolean(false);
      }
| i:INTEGER
      {
            dt = new BGGLInt(Integer.parseInt(i.getText()));
      }
| s:STRING
      {
            dt = new BGGLString(s.getText());
      }
| #(BLOCK
      { symbolTable.pushScope(); }
            (l = expression)*
            )
      { symbolTable.popScope(); }
| #(IF l = expression thenp:. (elsep:.)?)
      {
            symbolTable.pushScope();

            if ( ((BGGLBoolean)l).getValue()) {
                  expression(thenp);
            }
            else if (elsep != null) {
                  expression(elsep);
            }
            else {
            }

            symbolTable.popScope();
      }
| #(WHILE l = whileExpr:expression whileBody:.)
      {
            symbolTable.pushScope();

            if (! (l instanceof BGGLBoolean))
                  throw new TypeMismatchException("BGGLBoolean", l.getType());

            interpreter.whileLoop(whileExpr, whileBody);

            symbolTable.popScope();
      }
| BREAK
```

```
      {
            //terminate = true;
      }
| #(FOR forId:IDENTIFIER l=expression r=expression forbody:.)
      {
            symbolTable.pushScope();
            interpreter.forLoop(forId.getText(), l, r, forbody);
            symbolTable.popScope();
      }
| #(ASSIGN xId:IDENTIFIER e = expression)
      {
            symbolTable.set(xId.getText(), e);
      }
| v = declaration_assignment
| #(TURN tId:IDENTIFIER
      {
            symbolTable.pushScope();
      }
      (dt = expression)*
      {
            symbolTable.popScope();
      }
      )
| #(PRINT e = expression)
      {
            interpreter.print(e);
      }
| #(INPUT e = expression (in_int:TYPE_INT | in_str:TYPE_STRING))
      {
            String in_type = "";
            if (in_int != null) in_type = in_int.getText();
            else in_type = in_str.getText();
            dt = interpreter.input(e, in_type);
      }
| #(APPLY e = expression)
      {
            interpreter.applyMove(e);
      }
| #(CALL fId:IDENTIFIER al = expression_list moveList = move_list)
      {
            dt = interpreter.invoke(fId.getText(), al, moveList);
      }
| #(RETURN e = expression)
      {
            interpreter.setReturnValue(e);
      }
| #(TEST
            a = expression
            b = expression
            c = expression
            d = expression)
      {
```

```
            dt = interpreter.test(a, b, c, d);
      }
| #(MOVE_LENGTH (dt = expression)? )
| #(DIRECTION (dt = expression)? )
| #(JUMP_EMPTY (dt = expression)? )
| ORTHO
      {
            List temp = new ArrayList<BGGLString>();
            temp.add(new BGGLString("ortho"));

            dt = new BGGLArray(temp);
      }
| DIAG
      {
            List temp = new ArrayList<BGGLString>();
            temp.add(new BGGLString("diag"));

            dt = new BGGLArray(temp);
      }
| ORTHODIAG
      {
            List temp = new ArrayList<BGGLString>();
            temp.add(new BGGLString("orthodiag"));

            dt = new BGGLArray(temp);
      }
;
exception
      catch [OperationNotSupportedException ex]
      {
            ExceptionHandler.err(ex);
      }
      catch [IrregularBoardDimensionException ex]
      {
            ExceptionHandler.err(ex);
      }
      catch [TypeMismatchException ex]
      {
            ExceptionHandler.err(ex);
      }
      catch [IdentifierNotFoundException ex]
      {
            ExceptionHandler.err(ex);
      }
      catch [InvalidArgumentTypeException ex]
      {
            ExceptionHandler.err(ex);
      }
      catch [InvalidNumberOfArgumentsException ex]
      {
            ExceptionHandler.err(ex);
      }
```

```
declaration_assignment returns [Variable v]
{
      ADataType e = null;
      v = null;
}
: #(TYPE_INT iId:IDENTIFIER (ASSIGN e = expression)?)
      {
            if (e == null) {
                  e = new BGGLInt(0);
            }

            if (! (e instanceof BGGLInt))
                  throw new TypeMismatchException();

            v = new Variable(iId.getText(), e);
            symbolTable.add(v);
      }
| #(TYPE_BOOLEAN bId:IDENTIFIER (ASSIGN e = expression)?)
      {
            if (e == null) {
                  e = new BGGLBoolean(false);
            }

            if (! (e instanceof BGGLBoolean))
                  throw new TypeMismatchException();

            v = new Variable(bId.getText(), e);
            symbolTable.add(v);
      }
| #(TYPE_STRING sId:IDENTIFIER (ASSIGN e = expression)?)
      {
            if (e == null) {
                  e = new BGGLString("");
            }

            if (! (e instanceof BGGLString))
                  throw new TypeMismatchException();

            v = new Variable(sId.getText(), e);
            symbolTable.add(v);
      }
| #(TYPE_PIECE pId:IDENTIFIER (ASSIGN e = expression)?)
      {
            if (e == null) {
                  e = new BGGLPiece(pId.getText());
            }

            if (! (e instanceof BGGLPiece))
                  throw new TypeMismatchException();

            v= new Variable(pId.getText(), e);
```

```
            symbolTable.add(v);
      }
| #(TYPE_ARRAY aId:IDENTIFIER (ASSIGN e = expression)?)
      {
            if (e == null) {
                  e = new BGGLArray(new ArrayList());
            }

            if (! (e instanceof BGGLArray))
                  throw new TypeMismatchException();

            v = new Variable(aId.getText(), e);
            symbolTable.add(v);
      }
| #(TYPE_PLAYER plId:IDENTIFIER (ASSIGN e = expression)?)
      {
            if (e == null) {
                  e = new BGGLPlayer(plId.getText());
            }

            if (! (e instanceof BGGLPlayer))
                  throw new TypeMismatchException();

            v = new Variable(plId.getText(), e);
            symbolTable.add(v);
      }
| #(TYPE_MOVE mId:IDENTIFIER (ASSIGN e = expression)?)
      {
            if (e == null) {
                  e = new BGGLMove();
            }

            if (! (e instanceof BGGLMove))
                  throw new TypeMismatchException();

            v = new Variable(mId.getText(), e);
            symbolTable.add(v);
      }
;
exception
      catch [DuplicateIdentifierException ex]
      {
            ExceptionHandler.err(ex);
      }
      catch [TypeMismatchException ex]
      {
            ExceptionHandler.err(ex);
      }

expression_list returns [List<ADataType> expressions]
{
      expressions = new ArrayList<ADataType>();
```

```
      ADataType e = null;
}
: #(EXPRESSION_LIST
            (e = expression
                  { expressions.add(e); }
            )*
      )
      ;

move_list returns [List<BGGLMove> moves]
{
      moves = new ArrayList<BGGLMove>();
      ADataType move = null;
}
: #(MOVE_LIST
            (move = expression
                  {
                        moves.add((BGGLMove) move);
                  }
            )?
      )
      ;
```

Interpreter.java

```
package edu.bggl.interpreter;

import java.io.FileReader;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

import antlr.RecognitionException;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
import edu.bggl.exception.IdentifierNotFoundException;
import edu.bggl.exception.InvalidArgumentTypeException;
import edu.bggl.exception.InvalidNumberOfArgumentsException;
import edu.bggl.exception.OperationNotSupportedException;
import edu.bggl.exception.TypeMismatchException;
import edu.bggl.grammar.BGGLLexer;
import edu.bggl.grammar.BGGLParser;
import edu.bggl.grammar.BGGLWalker;
import edu.bggl.semantic.ADataType;
import edu.bggl.semantic.BGGLArray;
import edu.bggl.semantic.BGGLBoard;
import edu.bggl.semantic.BGGLBoolean;
import edu.bggl.semantic.BGGLFunction;
import edu.bggl.semantic.BGGLInt;
import edu.bggl.semantic.BGGLMove;
import edu.bggl.semantic.BGGLRule;
```

```
import edu.bggl.semantic.BGGLString;
import edu.bggl.semantic.SymbolTable;
import edu.bggl.semantic.Variable;

public class Interpreter
{
	public static final void main(String[] args)
	{
		//String filename = "tests/lexparse/valid/" + "TicTacToe" +
".bggl";
		//String filename = "tests/semantic/invalid/" + "ReturnMistype" +
".bggl";
		//String filename = "tests/print/valid/" + "BlockScopeVariables"
+ ".bggl";
		//String filename = "programs/" + "primes" + ".bggl";
		String filename = "programs/" + "ChineseCheckers" + ".bggl";

		boolean showTextAST = true;
		boolean showGuiAST = false;
		//boolean showGuiAST = true;

		if (args.length == 1)
		{
			filename = args[0];
			showTextAST = false;
			showGuiAST = false;
		}
		else if (args.length == 2)
		{
			showTextAST = true;

			if (args[0] == "-t")
				showGuiAST = false;
			else
				showGuiAST = true;

			filename = args[1];
		}

		execute(filename, showTextAST, showGuiAST);
	}

	public static void execute(
		String filename,
		boolean showTextAst,
		boolean showGuiAst)
	{
		try
		{
			System.out.println("File: " + filename);

			FileReader fr = new FileReader(filename);
```

```
                BGGLParser parser = new BGGLParser(new BGGLLexer(fr));
                parser.main_program();

                if (showTextAst)
                      System.out.println(
                            "Tree:\n" +
                            parser.getAST().toStringTree() +
                            "\n");

                if (showGuiAst)
                {
                      ASTFrame frame = new ASTFrame("BGGL",
parser.getAST());
                      frame.setVisible(true);
                }

                SymbolTable.getInstance().reset();

                // first do static semantic analysis
                Interpreter interpreter = new Interpreter(
                      System.out,
                      parser.getAST(),
                      false);
                interpreter.run();

                SymbolTable.getInstance().reset();

                // then do actual interpretation
                interpreter = new Interpreter(
                      System.out,
                      parser.getAST(),
                      true);
                interpreter.run();
          }
          catch( Exception e )
          {
                e.printStackTrace();
          }
    }

    private BGGLWalker walker;
    private PrintStream out;
    private List<String> outBuffer;
    private Scanner in;
    private AST ast;

    public static boolean enableInterpretation;

    private ADataType returnValue;
    // For function termination.
    private boolean functionReturned;
```

```
	public Interpreter(AST ast, boolean enableInterpretation)
	{
		this(System.out, ast, enableInterpretation);
	}

	/**
	 * @param out If <code>null</code> then "prints" to an internal buffer.
	 * @param ast
	 */
	public Interpreter(PrintStream out, AST ast, boolean
enableInterpretation)
	{
		// initialize "stdout"
		this.out = out;
		if (this.out == null)
			outBuffer = new ArrayList<String>();

		// initialize "stdin"
		this.in = new Scanner(System.in);

		this.ast = ast;
		this.walker = new BGGLWalker(this);

		this.returnValue = null;
		functionReturned = false;

		this.enableInterpretation = enableInterpretation;
	}

	public void run()
		throws RecognitionException
	{
		walker.main_program(ast);
	}

	public List<String> getOutBuffer()
	{
		return this.outBuffer;
	}

	public void print(ADataType dt)
	{
		print_ln(dt, true);
	}

	public void print_ln(ADataType dt, boolean linebreak)
	{
		if (! enableInterpretation)
			return;

		if (functionReturned) return;
```

```
            if (this.out != null)
            {
                  if (linebreak)
                        this.out.println(dt.toString());
                  else
                        this.out.print(dt.toString());
            }
            else
            {
                  if (linebreak)
                        this.outBuffer.add(dt.toString());
                  else
                  {
                        int lastIndex = outBuffer.size() - 1;
                        String s = outBuffer.get(lastIndex);

                        s += dt.toString();

                        outBuffer.set(lastIndex, s);
                  }
            }
      }

      /**
       * Reads in input from stdin.
       *
       * @param prompt The prompt to print.
       * @param responseId The name of the variable that will store the
response.
       * It will be the correct type.
       */
      public ADataType input(ADataType prompt, String type)
      {
            if (! enableInterpretation)
            {
                  if (type.equals("string")) return new BGGLString();
                  else return new BGGLInt();
            }

            if (functionReturned) return null;

            print_ln(prompt, false);

            String inputStr = in.nextLine();

            if (type.equals("string"))
            {
                  return new BGGLString(inputStr);
            }
            else
            {
                  int i = 0;
```

```
                while (true)
                {
                        try
                        {
                                i = Integer.parseInt(inputStr);
                        }
                        catch (NumberFormatException ex)
                        {
                                print(
                                        new BGGLString("Invalid input. Please try
again."));
                                print_ln(prompt, false);
                                inputStr = in.nextLine();
                                continue;
                        }
                        return new BGGLInt(i);
                }
        }
    }

    // The current functor being executed
    private BGGLFunction currentFunctor = null;

    public ADataType invoke(
            String functorId,
            List<ADataType> arguments,
            List<BGGLMove> moves)
            throws RecognitionException,
                    IdentifierNotFoundException,
                    InvalidArgumentTypeException,
                    InvalidNumberOfArgumentsException
    {
            SymbolTable st = SymbolTable.getInstance();

            BGGLFunction f =
                    (BGGLFunction) st.get(functorId, false)
                            .getDataType();
            this.returnValue = null;

            // check arguments
            List<Variable> parameters = f.getParameters();
            if (parameters.size() != arguments.size())
            {
                    throw new InvalidNumberOfArgumentsException(
                            f.getId(),
                            parameters.size(),
                            arguments.size());
            }
            for (int i = 0; i < parameters.size(); i ++)
            {
                    ADataType param = parameters.get(i).getDataType();
                    ADataType arg = arguments.get(i);
```

```
                if (! param.getClass().equals(arg.getClass()))
                      throw new InvalidArgumentTypeException(
                            f.getId(),
                            param.getType(),
                            arg.getType());
          }

          if (functionReturned) return null;

          // initialize function call environment
          st.setBreakpoint(true);
          st.pushScope();

          // initialize parameters
          f.initializeParameters(arguments);

          // save it; this is only used for rules and "test" predicate
          currentFunctor = f;
          if (f instanceof BGGLRule)
          {
                if (moves.size() != 0) {
                      BGGLMove curr_move = (BGGLMove)moves.get(0);
                      st.addNoCheck(BGGLRule.ID_MOVETYPE, new
BGGLInt(curr_move.getMoveType()));
                      st.addNoCheck(BGGLRule.ID_THISPIECE,
curr_move.getPiece());
                      st.addNoCheck(BGGLRule.ID_R1, new
BGGLInt(curr_move.getX1()));
                      st.addNoCheck(BGGLRule.ID_C1, new
BGGLInt(curr_move.getY1()));
                      st.addNoCheck(BGGLRule.ID_R2, new
BGGLInt(curr_move.getX2()));
                      st.addNoCheck(BGGLRule.ID_C2, new
BGGLInt(curr_move.getY2()));
                }

                ((BGGLRule) f).setMove(moves.get(0));
                // add them to this scope
                st.addNoCheck(BGGLRule.ID_THISMOVE, moves.get(0));
          }

          // Now walk its AST
          walker.expression(f.getRoot());
          // Clear the function returned flag, so that interpretation is
restored.
          functionReturned = false;

          // restore original environment
          st.popScope();
          st.setBreakpoint(false);
```

```
		return this.returnValue;
	}

	public void setReturnValue(ADataType value)
		throws TypeMismatchException
	{
		String expectedType = currentFunctor.getReturnValue().getType();
		String actualType = value.getType();

		if (! expectedType.equals(actualType))
			throw new TypeMismatchException(expectedType, actualType);

		if (functionReturned) return;

		this.returnValue = value;
		functionReturned = true;
	}

	public void forLoop (
		String counterId,
		ADataType start,
		ADataType end,
		AST bodyRoot)
		throws RecognitionException,
			OperationNotSupportedException,
			TypeMismatchException
	{
		SymbolTable st = SymbolTable.getInstance();

		// type checking
		ADataType counter = st.getNoCheck(counterId).getDataType();
		if (! (counter instanceof BGGLInt))
			throw new TypeMismatchException("BGGLInt",
counter.getType());
		if (! (start instanceof BGGLInt))
			throw new TypeMismatchException("BGGLInt", start.getType());
		if (! (end instanceof BGGLInt))
			throw new TypeMismatchException("BGGLInt", end.getType());

		if (! enableInterpretation)
		{
			walker.expression(bodyRoot);
			return;
		}

		int a = ((BGGLInt) start).getValue();
		int z = ((BGGLInt) end).getValue();
		for (int itr = a; itr <= z; itr ++) {
			// update counter
			st.setNoCheck(counterId, new BGGLInt(itr));

			st.pushScope();
```

```
                walker.expression(bodyRoot);
                st.popScope();
        }
    }

    private final int LOOP_LIMIT = 200;

    public void whileLoop(AST whileExpression, AST bodyRoot)
        throws RecognitionException
    {
        SymbolTable st = SymbolTable.getInstance();

        if (! enableInterpretation)
        {
            walker.expression(bodyRoot);
            return;
        }

        BGGLBoolean b = (BGGLBoolean) walker.expression(whileExpression);

        int loopCounter = 0;
        while (b.get() && loopCounter < LOOP_LIMIT) {
            st.pushScope();
            walker.expression(bodyRoot);
            st.popScope();

            b = (BGGLBoolean) walker.expression(whileExpression);
            loopCounter ++;
        }
    }

    public BGGLBoolean test(
        ADataType moveLength,
        ADataType direction,
        ADataType jump,
        ADataType empty)
        throws OperationNotSupportedException,
            TypeMismatchException
    {
        BGGLRule currentRule = (BGGLRule) currentFunctor;

        if (moveLength != null && ! (moveLength instanceof BGGLInt))
            throw new TypeMismatchException("BGGLInt",
moveLength.getType());
        if (direction != null && ! (direction instanceof BGGLArray))
            throw new TypeMismatchException("BGGLArray",
direction.getType());
        if (jump != null && ! (jump instanceof BGGLBoolean))
            throw new TypeMismatchException("BGGLBoolean",
jump.getType());
        if (empty != null && ! (empty instanceof BGGLBoolean))
```

```
                  throw new TypeMismatchException("BGGLBoolean",
empty.getType());

            if (! enableInterpretation) return new BGGLBoolean();

            if (moveLength != null) {
                  currentRule.setMoveLength((BGGLInt) moveLength);
            }
            if (jump != null) {
                  currentRule.setJump((BGGLBoolean)jump);
            }
            if (empty != null) {
                  currentRule.setEmptySquare((BGGLBoolean)empty);
            }
            if (direction != null) {
                  BGGLArray array = (BGGLArray) direction;

                  if (array.get().get(0) instanceof BGGLString)
                  {
                        String option = ((BGGLString)
array.get().get(0)).getValue();
                        if (option.compareTo("ortho") == 0) {
                              currentRule.setOrtho();
                        } else if (option.compareTo("diag") == 0) {
                              currentRule.setDiag();
                        } else if (option.compareTo("orthodiag") == 0) {
                              currentRule.setOrtho();
                              currentRule.setDiag();
                        }
                  } else {
                        // it's a custom direction, which is a BGGLArray
                        // of BGGLArrays of BGGLInts
                        List<List<Integer>> custom = new
ArrayList<List<Integer>>();

                        List<ADataType> l = array.get();
                        // each of these elements is a size 2 list
                        for (int i = 0; i < l.size(); i ++)
                        {
                              List<ADataType> ll = ((BGGLArray)
l.get(i)).get();

                              List<Integer> pair = new ArrayList<Integer>();

                              int row = ((BGGLInt) ll.get(0)).getValue();
                              int col = ((BGGLInt) ll.get(1)).getValue();

                              pair.add(row);
                              pair.add(col);

                              custom.add(pair);
                        }
```

```
                        currentRule.setCustom(custom);
                  }
            }

            // ...
            BGGLBoolean b =
                  new BGGLBoolean(
                        currentRule.checkMoves(
                              (BGGLBoard)
                                    SymbolTable.getInstance()
                                          .getNoCheck(BGGLBoard.ID_BOARD,
false)
                                                .getDataType()
                              )
                        );

            return b;
      }

      public void applyMove(ADataType e)
            throws TypeMismatchException
      {
            if (! (e instanceof BGGLMove))
            {
                  throw new TypeMismatchException("BGGLMove", e.getType());
            }

            if (! enableInterpretation) return;

            ((BGGLBoard)
                  SymbolTable.getInstance()
                        .getNoCheck(BGGLBoard.ID_BOARD, false)
                        .getDataType()
                  )
                  .applyMove(
                        (BGGLMove) e
                        );
      }
}
```

ADataType.java

```
package edu.bggl.semantic;

import edu.bggl.exception.OperationNotSupportedException;

/**
 * The parent class of all BGGL data types. By default it supports no
operations.
 *
 * @author matt
```

```
 *
 * @param <T> The type, either Boolean, Integer, String, or BGGLArray
 */
public abstract class ADataType
{
	public ADataType()
	{
	}

	// Basic operations. By default, nothing is allowed.
	public ADataType negate()
	throws OperationNotSupportedException
	{
		throw new OperationNotSupportedException();
	}

	public ADataType mult(ADataType r)
		throws OperationNotSupportedException
	{
		throw new OperationNotSupportedException();
	}

	public ADataType div(ADataType r)
		throws OperationNotSupportedException
	{
		throw new OperationNotSupportedException();
	}

	public ADataType add(ADataType r)
		throws OperationNotSupportedException
	{
		throw new OperationNotSupportedException();
	}

	public ADataType sub(ADataType r)
	throws OperationNotSupportedException
	{
		throw new OperationNotSupportedException();
	}
	public ADataType mod(ADataType r)
	throws OperationNotSupportedException
	{
		throw new OperationNotSupportedException();
	}

	// Basic comparisons. All subclasses should to define this.
	// Should it be an error if you try to compare two different
	// types, like a boolean and an int?
	public BGGLBoolean and(ADataType r)
	throws OperationNotSupportedException
	{
		throw new OperationNotSupportedException();
```

```
}

public BGGLBoolean or(ADataType r)
throws OperationNotSupportedException
{
      throw new OperationNotSupportedException();
}

public BGGLBoolean not()
throws OperationNotSupportedException
{
      throw new OperationNotSupportedException();
}

public BGGLBoolean eq(ADataType r)
throws OperationNotSupportedException
{
      throw new OperationNotSupportedException();
}

public BGGLBoolean neq(ADataType r)
throws OperationNotSupportedException
{
      throw new OperationNotSupportedException();
}
public BGGLBoolean ge(ADataType r)
throws OperationNotSupportedException
{
      throw new OperationNotSupportedException();
}

public BGGLBoolean le(ADataType r)
throws OperationNotSupportedException
{
      throw new OperationNotSupportedException();
}
public BGGLBoolean gt(ADataType r)
throws OperationNotSupportedException
{
      throw new OperationNotSupportedException();
}

public BGGLBoolean lt(ADataType r)
throws OperationNotSupportedException
{
      throw new OperationNotSupportedException();
}

public String getType()
{
      return this.getClass().getSimpleName();
}}
```

BGGLArray.java

```
package edu.bggl.semantic;

import java.util.ArrayList;
import java.util.List;

import edu.bggl.exception.OperationNotSupportedException;

/**
 * @author Steve Moncada
 */
public class BGGLArray extends ADataType {

	private List<ADataType> pieceSpan;

	public BGGLArray()
	{
		this(new ArrayList<ADataType>());
	}

	public BGGLArray(List<ADataType> list)
	{
		super();
		pieceSpan = list;
	}

	public List<ADataType> get() {
		return pieceSpan;
	}

	public ADataType getElement(int x) {
		return this.pieceSpan.get(x);
	}

	public void setElement(int x, ADataType input_element) {
		this.pieceSpan.set(x, input_element);
	}

	public String toString() {
		String spanString = "[ ";

		for (ADataType obj : pieceSpan)
		{
			spanString += obj.toString() + " ";
		}
		spanString += "]";
		return spanString;
	}

	public int getSize() {
		return pieceSpan.size();
```

```
	}

	@Override
	public BGGLBoolean eq(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLArray))
			throw new OperationNotSupportedException();

		//boolean retVal = false;
		BGGLArray that = (BGGLArray) r;

		if (this.pieceSpan.size() == that.pieceSpan.size())
		{
			for (int i = 0; i < this.pieceSpan.size(); i ++)
			{
				ADataType ll = this.pieceSpan.get(i);
				ADataType rr = that.pieceSpan.get(i);

				if (! ll.eq(rr).getValue())
				{
					return new BGGLBoolean(false);
				}
			}

			return new BGGLBoolean(true);
		}

		return new BGGLBoolean(false);
	}
}
```

BGGLBoard.java

```
package edu.bggl.semantic;

import java.util.ArrayList;
import java.util.List;

import edu.bggl.exception.IrregularBoardDimensionException;
import edu.bggl.exception.OperationNotSupportedException;
import edu.bggl.exception.TypeMismatchException;

/**
 * @author Steve Moncada
 */
public class BGGLBoard extends ADataType
{
	public static final String ID_BOARD = "board";

	private List<BGGLArray> boardMatrix;
	private BGGLPiece invalidPiece;
```

```
	private BGGLPiece emptyPiece;

	public BGGLBoard()
		throws IrregularBoardDimensionException,
			TypeMismatchException
	{
		this(new ArrayList<BGGLArray>());
	}

	public BGGLBoard(List<BGGLArray> input_board)
		throws IrregularBoardDimensionException,
			TypeMismatchException
	{
		// check all arrays same size and that all elements are of type
piece
		for (int i = 0; i < input_board.size(); i ++)
		{
			if (i != 0 &&
				input_board.get(i - 1).getSize() !=
input_board.get(i).getSize())
				throw new IrregularBoardDimensionException();
			List<ADataType> a = input_board.get(i).get();
			for (ADataType b : a)
			{
				if (! (b instanceof BGGLPiece))
					throw new TypeMismatchException("BGGLPiece",
b.getType());
			}
		}

		initBoard();

		boardMatrix = input_board;
		for (int i=0; i < this.getLength(); i++) {
			this.setRow(i, input_board.get(i));
		}
	}

	private void initBoard()
	{
		SymbolTable st = SymbolTable.getInstance();
		invalidPiece = (BGGLPiece) st.getNoCheck("#").getDataType();
		emptyPiece = (BGGLPiece) st.getNoCheck("_").getDataType();
	}

	public void applyMove(BGGLMove input_move) {
		int movetype = input_move.getMoveType();
		switch (movetype) {
		case BGGLMove.ADD:
			this.setPiece(input_move.getX1(), input_move.getY1(),
input_move.getPiece());
			break;
```

```
            case BGGLMove.REMOVE:
                  this.setPiece(input_move.getX1(), input_move.getY1(),
this.emptyPiece);
                  break;
            case BGGLMove.MOVE:
                  this.setPiece(input_move.getX1(), input_move.getY1(),
this.emptyPiece);
                  this.setPiece(input_move.getX2(), input_move.getY2(),
input_move.getPiece());
                  break;
            }
      }

      public void setPiece(int x, int y, BGGLPiece input_piece) {
            this.boardMatrix.get(x).setElement(y, input_piece);
      }

      public BGGLPiece getPiece(int x, int y) {
            return (BGGLPiece) this.boardMatrix.get(x).getElement(y);
      }

      public void setRow(int x, BGGLArray input_row) {
            this.boardMatrix.set(x, input_row);
      }

      public BGGLArray getRow(int x) {
            return this.boardMatrix.get(x);
      }

      public void setCol(int y, BGGLArray input_col) {
            for (int i=0; i<input_col.getSize(); i++) {
                  this.boardMatrix.get(i).setElement(y, (BGGLPiece)
input_col.get().get(y));
            }
      }

      public BGGLArray getCol(int y) {
            ArrayList<ADataType> tempCol = new ArrayList<ADataType>();
            for (int i=0; i<this.getLength(); i++) {
                  tempCol.add(i, this.boardMatrix.get(i).getElement(y));
            }
            return new BGGLArray(tempCol);
      }

      public BGGLArray getForwardDiag() {
            ArrayList<ADataType> tempDiag = new ArrayList<ADataType>();
            for (int i=0; i<this.getLength(); i++) {
                  tempDiag.add(i, this.boardMatrix.get((this.getLength()-1)-
i).getElement(i));
            }
            return new BGGLArray(tempDiag);
      }
```

```
	public BGGLArray getBackwardDiag() {
		ArrayList<ADataType> tempDiag = new ArrayList<ADataType>();
		for (int i=0; i<this.getLength(); i++) {
			tempDiag.add(i, this.boardMatrix.get(i).getElement(i));
		}
		return new BGGLArray(tempDiag);
	}

	public List<BGGLArray> get() {
		return this.boardMatrix;
	}

	public void set(List<BGGLArray> t) {
		this.boardMatrix = t;
	}

	public int getWidth() { return this.boardMatrix.get(0).getSize(); }
	public int getLength() { return this.boardMatrix.size(); }

	public int countPiece(BGGLPiece input_piece) {
		int piececount = 0;
		for (int l = 0; l < this.getLength(); l++) {
			for (int w = 0; w < this.getWidth(); w++) {
				if
(input_piece.equals(this.boardMatrix.get(l).getElement(w))) {
					piececount++;
				}
			}
		}
		return piececount;
	}

	public String toString() {
		String boardString = new String("   ");
		if (this.getWidth() > 10) {
			boardString = boardString.concat("                    ");
			for (int w=10; w< this.getWidth(); w++) {
				boardString += (w/10) + " ";
			}
			boardString += "\n   ";
		}
		for (int w = 0; w < this.getWidth(); w++) {
			boardString = boardString.concat((w%10) + " ");
		}
		boardString = boardString.concat("\n");
		for (int l = 0; l < this.getLength(); l++) {
			if (l<10) { boardString = boardString.concat(" "); }
			boardString = boardString.concat((l) + " ");
			for (int w = 0; w < this.getWidth(); w++) {
				boardString +=
```

```
      this.boardMatrix.get(l).getElement(w).toString() +
                              " ";
                  }
                  if (l<10) { boardString = boardString.concat(" "); }
                  boardString = boardString.concat((l) + " \n");

            }
            boardString = boardString.concat("   ");

            if (this.getWidth() > 10) {
                  boardString = boardString.concat("                    ");
                  for (int w=10; w< this.getWidth(); w++) {
                        boardString += (w/10) + " ";
                  }
                  boardString += "\n   ";
            }
            for (int w = 0; w < this.getWidth(); w++) {
                  boardString = boardString.concat((w%10) + " ");
            }
            boardString += "\n";

            return boardString;
      }

      //methods to compare if a given square on a board is invalid or blank
      public boolean isBlank(int x, int y) throws
OperationNotSupportedException
      {
            SymbolTable st = SymbolTable.getInstance();
            BGGLPiece blankPiece = (BGGLPiece) st.getNoCheck("_",
false).getDataType();
            return this.getPiece(x, y).eq(blankPiece).getValue();
      }

      public boolean isInvalid(int x, int y) throws
OperationNotSupportedException
      {
            SymbolTable st = SymbolTable.getInstance();
            BGGLPiece invalidPiece = (BGGLPiece) st.getNoCheck("#",
false).getDataType();
            return this.getPiece(x, y).eq(invalidPiece).getValue();
      }
}
```

BGGLBoolean.java

```
package edu.bggl.semantic;

import edu.bggl.exception.OperationNotSupportedException;

public class BGGLBoolean extends ADataType
```

```
{
	private boolean b;

	public BGGLBoolean()
	{
		this(false);
	}

	public BGGLBoolean(boolean b)
	{
		super();
		this.b = b;
	}

	public String toString()
	{
		return b + "";
	}

	public void set(Boolean b)
	{
		this.b = b;
	}

	public Boolean get()
	{
		return this.b;
	}
	public boolean getValue() {
		return this.b;
	}

	@Override
	public BGGLBoolean and(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLBoolean))
			throw new OperationNotSupportedException();

		return new BGGLBoolean(this.b && ((BGGLBoolean) r).get());
	}

	@Override
	public BGGLBoolean or(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLBoolean))
			throw new OperationNotSupportedException();

		return new BGGLBoolean(this.b || ((BGGLBoolean) r).get());
	}
```

```
	@Override
	public BGGLBoolean not()
		throws OperationNotSupportedException
	{
		return new BGGLBoolean(!(this.b));
	}

	@Override
	public BGGLBoolean eq(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLBoolean))
			throw new OperationNotSupportedException();

		return new BGGLBoolean(this.b == ((BGGLBoolean) r).get());
	}

	@Override
	public BGGLBoolean neq(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLBoolean))
			throw new OperationNotSupportedException();

		return new BGGLBoolean(this.b != ((BGGLBoolean) r).get());
	}

}
```

BGGLFunction.java

```
package edu.bggl.semantic;

import java.util.List;

import antlr.collections.AST;

public class BGGLFunction extends ADataType
{
	private String name;
	private List<Variable> parameters;
	private ADataType returnValue;

	private AST root;

	public static final String RETURN_VARIABLE_ID = "_RETURN_";

	public BGGLFunction(
		String name,
		List<Variable> parameters,
		ADataType returnValue,
		AST root)
```

```
	{
		super();

		this.name = name;
		this.parameters = parameters;
		this.returnValue = returnValue;

		this.root = root;
	}

	@Override
	public String toString()
	{
		return name;
	}

	public void initializeParameters(List<ADataType> arguments)
	{
		for (int i = 0; i < parameters.size(); i ++)
		{
			SymbolTable.getInstance().addNoCheck(
				parameters.get(i).getIdentifier(),
				arguments.get(i)
				);
		}
	}

	public AST getRoot()
	{
		return this.root;
	}

	public ADataType getReturnValue()
	{
		return returnValue;
	}

	public List<Variable> getParameters()
	{
		return this.parameters;
	}

	public String getId()
	{
		return this.name;
	}
}
```

BGGLInt.java

```
package edu.bggl.semantic;
```

```
import edu.bggl.exception.OperationNotSupportedException;
import edu.bggl.interpreter.Interpreter;

public class BGGLInt extends ADataType
{
	private int i;

	public BGGLInt()
	{
		this(0);
	}

	public BGGLInt(int i)
	{
		super();
		this.i = i;
	}

	public String toString()
	{
		return i + "";
	}

	public void set(Integer i)
	{
		this.i = i;
	}

	public Integer get()
	{
		return this.i;
	}

	public int getValue() {
		return this.i;
	}

	@Override
	public ADataType add(ADataType r)
		throws OperationNotSupportedException
	{
		if (r instanceof BGGLInt) {
			return new BGGLInt(this.i + ((BGGLInt) r).getValue());
		}
		else if (r instanceof BGGLString) {
			return new BGGLString("" + this.i +
((BGGLInt)r).getValue());
		}
		else {
			throw new OperationNotSupportedException();
		}
	}
```

```
@Override
public ADataType div(ADataType r)
      throws OperationNotSupportedException
{
      if (! (r instanceof BGGLInt))
            throw new OperationNotSupportedException();

      // yet another hack.
      if (! Interpreter.enableInterpretation)
            return new BGGLInt();

      return new BGGLInt(this.i / ((BGGLInt) r).getValue());
}

@Override
public ADataType mod(ADataType r)
      throws OperationNotSupportedException
{
      if (! (r instanceof BGGLInt))
            throw new OperationNotSupportedException();

      // yet another hack.
      if (! Interpreter.enableInterpretation)
            return new BGGLInt();

      return new BGGLInt(this.i % ((BGGLInt) r).getValue());
}

@Override
public BGGLBoolean eq(ADataType r)
      throws OperationNotSupportedException
{
      if (! (r instanceof BGGLInt))
            throw new OperationNotSupportedException();

      return new BGGLBoolean(this.i == ((BGGLInt) r).getValue());
}

@Override
public ADataType mult(ADataType r)
      throws OperationNotSupportedException
{
      if (! (r instanceof BGGLInt))
            throw new OperationNotSupportedException();

      return new BGGLInt(this.i * ((BGGLInt) r).getValue());
}

@Override
public ADataType negate()
      throws OperationNotSupportedException
```

```
{
	return new BGGLInt(-(this.i));
}

@Override
public BGGLBoolean neq(ADataType r)
	throws OperationNotSupportedException
{
	if (! (r instanceof BGGLInt))
		throw new OperationNotSupportedException();

	return new BGGLBoolean(this.i != ((BGGLInt) r).getValue());
}

@Override
public ADataType sub(ADataType r)
	throws OperationNotSupportedException
{
	if (! (r instanceof BGGLInt))
		throw new OperationNotSupportedException();

	return new BGGLInt(this.i - ((BGGLInt) r).getValue());
}

@Override
public BGGLBoolean ge(ADataType r)
	throws OperationNotSupportedException
{
	if (! (r instanceof BGGLInt))
		throw new OperationNotSupportedException();

	return new BGGLBoolean(this.i >= ((BGGLInt) r).getValue());
}

@Override
public BGGLBoolean le(ADataType r)
	throws OperationNotSupportedException
{
	if (! (r instanceof BGGLInt))
		throw new OperationNotSupportedException();

	return new BGGLBoolean(this.i <= ((BGGLInt) r).getValue());
}
@Override
public BGGLBoolean gt(ADataType r)
	throws OperationNotSupportedException
{
	if (! (r instanceof BGGLInt))
		throw new OperationNotSupportedException();

	return new BGGLBoolean(this.i > ((BGGLInt) r).getValue());
}
```

```
	@Override
	public BGGLBoolean lt(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLInt))
			throw new OperationNotSupportedException();

		return new BGGLBoolean(this.i < ((BGGLInt) r).getValue());
	}
}
```

BGGLMove.java

```
package edu.bggl.semantic;

/*
 * BGGLMoves to be passed through the rules and eventually to the board.
 *
 * @author vitaliy, steve
 */

public class BGGLMove extends ADataType
{
	private BGGLPiece Piece;
	private int x1, x2, y1, y2;
	private int move_type;

	public static final int ADD = 1;
	public static final int REMOVE = 2;
	public static final int MOVE = 3;

	public int getX1() { return this.x1; }
	public int getX2() { return this.x2; }
	public int getY1() { return this.y1; }
	public int getY2() { return this.y2; }
	public BGGLPiece getPiece() { return this.Piece; }
	public int getMoveType() { return this.move_type; }

	//type = add or remove
	public BGGLMove(int move_type, BGGLPiece Piece, BGGLInt x1, BGGLInt y1)
	{
		this.move_type = move_type;
		this.Piece = Piece;
		this.x1 = x1.getValue();
		this.y1 = y1.getValue();
	}

	//needed for walker!!
	public BGGLMove() {

	}
```

```
	//type = move
	public BGGLMove(int move_type, BGGLPiece Piece, BGGLInt x1, BGGLInt y1,
BGGLInt x2, BGGLInt y2)
	{
		this.move_type = move_type;
		this.Piece = Piece;
		this.x1 = x1.getValue();
		this.y1 = y1.getValue();
		this.x2 = x2.getValue();
		this.y2 = y2.getValue();
	}

	public String toString()
	{
		String output = new String("");
		switch (this.move_type) {
		case ADD:
			output = output.concat("Adding " + Piece + " to " + x1 + ",
" + y1);
			break;
		case REMOVE:
			output = output.concat("Removing " + Piece + " from " + x1
+ ", " + y1);
			break;
		case MOVE:
			output = output.concat("Moving " + Piece + " from " + x2 +
", " + y2 +
                                                " to " + x1 + ", " + y1);
			break;
		}
		return output;
	}

}
```

BGGLPiece.java

```
package edu.bggl.semantic;

import edu.bggl.exception.OperationNotSupportedException;

public class BGGLPiece extends ADataType
{
	private String Piece;

	public BGGLPiece()
	{
		this("_PIECE_");
	}

	public BGGLPiece(String Piece)
```

```
	{
		super();
		this.Piece = Piece;
	}

	public String toString()
	{
		return Piece + "";
	}

	public void set(String Piece)
	{
		this.Piece = Piece;
	}

	public String getValue()
	{
		return this.Piece;
	}

	@Override
	public BGGLBoolean eq(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLPiece))
			throw new OperationNotSupportedException();

		BGGLPiece that = (BGGLPiece) r;

		return new BGGLBoolean(this.Piece.equals(that.Piece));
	}

	@Override
	public BGGLBoolean neq(ADataType r)
		throws OperationNotSupportedException
	{
		return new BGGLBoolean(! this.eq(r).getValue());
	}

	/**
	 * You can concat with a string.
	 */
	@Override
	public ADataType add(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLString))
			throw new OperationNotSupportedException();

		return new BGGLString(this.Piece + r);
	}
}
```

BGGLPlayer.java

```
package edu.bggl.semantic;

import edu.bggl.exception.OperationNotSupportedException;

public class BGGLPlayer extends ADataType
{
	private String Player;

	public BGGLPlayer() {
		this("_PLAYER_");
	}

	public BGGLPlayer(String Player)
	{
		this.Player= Player;
	}

	public String toString()
	{
		return Player + "";
	}

	public void set(String Player)
	{
		this.Player = Player;
	}

	public String get()
	{
		return this.Player;
	}

	public String toStriong() {
			return this.Player;
	}

	@Override
	public BGGLBoolean eq(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLPlayer))
			throw new OperationNotSupportedException();

		BGGLPlayer that = (BGGLPlayer) r;
		return new BGGLBoolean(this.Player.equals(that.Player));
	}

	@Override
	public BGGLBoolean neq(ADataType r)
		throws OperationNotSupportedException
```

```
	{
		return new BGGLBoolean(! this.eq(r).getValue());
	}

	/**
	 * You can concat with a string.
	 */
	@Override
	public ADataType add(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLString))
			throw new OperationNotSupportedException();

		return new BGGLString(this.Player + r);
	}
}
```

BGGLRule.java

```
package edu.bggl.semantic;

import java.util.List;

import edu.bggl.exception.OperationNotSupportedException;

import antlr.collections.AST;

/**
 * @author matt
 */
public class BGGLRule extends BGGLFunction
{

	private List<String> pieces;

	public static final String RETURN_VARIABLE_ID = "_RETURN_";
	public static final String ID_THISMOVE = "thismove";
	public static final String ID_MOVETYPE= "movetype";
	public static final String ID_THISPIECE = "thispiece";
	public static final String ID_R1 = "r1";
	public static final String ID_C1 = "c1";
	public static final String ID_R2 = "r2";
	public static final String ID_C2 = "c2";

	//Fields for automatic movement checks
	private enum setting { UNSET, TRUE, FALSE; }
	private setting emptysquare, jump;
	private int moveLength;
	private int direction;
	private final int direction_orthoMask = 0x1;
	private final int direction_diagMask = 0x2;
```

```
private final int direction_customMask = 0x4;
private List<List<Integer>> customDirections;
private int customDirectionTaken;

private BGGLMove m;


public BGGLRule(
      String name,
      List<Variable> parameters,
      List<String> pieces,
      AST root)
{
      super(name, parameters, new BGGLBoolean(), root);

      this.pieces = pieces;
      moveLength = 0;
      direction = 0;
      emptysquare = setting.UNSET;
      jump = setting.UNSET;
      customDirectionTaken = -1;
      m = null;
}

public void setMoveLength(BGGLInt length)
{
      moveLength = length.getValue();
}

public void setOrtho()
{
      direction |= direction_orthoMask;
}

public void setDiag()
{
      direction |= direction_diagMask;
}

public void setMove(BGGLMove m)
{
      this.m = m;
}

public void setJump(BGGLBoolean jump)
{
      if (jump.getValue() == true) {
            this.jump = setting.TRUE;
      } else {
            this.jump = setting.FALSE;
      }
}
```

```
	public void setCustom(List<List<Integer>> customDirections)
	{
		direction |= direction_customMask;
		this.customDirections = customDirections;
	}

	public void setEmptySquare(BGGLBoolean emptysquare)
	{
		if (emptysquare.getValue() == true) {
			this.emptysquare = setting.TRUE;
		} else {
			this.emptysquare = setting.FALSE;
		}
	}

	public boolean checkMoves(BGGLBoard b) throws
OperationNotSupportedException
	{
		BGGLPiece piece = m.getPiece();
		if (!pieces.contains(piece.getValue())) {
			return true;
		}

		//get the coordinates
		int x1, x2, y1, y2;
		int move_type = m.getMoveType();
		if (move_type == BGGLMove.ADD) {
			x1 = -1;
			y1 = -1;
			x2 = m.getX1();
			y2 = m.getY1();
		} else if (move_type == BGGLMove.REMOVE) {
			x1 = m.getX1();
			y1 = m.getY1();
			x2 = -1;
			y2 = -1;
		} else {
			//moving element
			x1 = m.getX1();
			y1 = m.getY1();
			x2 = m.getX2();
			y2 = m.getY2();
		}

		if (x1 == x2 && y1 == y2) {
			return false;
		}

		if (x1 >= b.getLength() || x2 >= b.getLength() || y1 >=
b.getWidth() || y2 >= b.getWidth()) {
			return false;
```

```
            }

            if ((x1 >= 0 && y1 >= 0) && b.isInvalid(x1, y1) || (x2 >= 0 && y2
>= 0) && b.isInvalid(x2, y2)) {
                  return false;
            }

            //check if the piece on the source square specified in the move
is actually the one present on the board
            if (x1 != -1) {
                  if (b.getPiece(x1, y1).neq(piece).getValue()) {
                        return false;
                  }
            }

            //the order of function calls in this if statement is important
            if (this.checkEmptySquare(b, x1, y1, x2, y2) &&
                        this.checkDirection(x1, y1, x2, y2) &&
                        this.checkLengthJump(b, x1, y1, x2, y2)) {
                  return true;
            } else {
                  return false;
            }
      }

      private boolean checkEmptySquare(BGGLBoard b, int x1, int y1, int x2,
int y2) throws OperationNotSupportedException
      {
            if (x2 != -1) {
                  //piece has either been added or moved
                  if (emptysquare == setting.TRUE && b.isBlank(x2, y2)) {
                        return true;
                  } else if (emptysquare == setting.FALSE && !b.isBlank(x2,
y2)) {
                        return true;
                  } else if (emptysquare == setting.UNSET) {
                        return true;
                  } else {
                        return false;
                  }
            } else {
                  return true;
            }
      }

      private boolean checkDirection(int x1, int y1, int x2, int y2)
      {
            if (x1 == -1 || x2 == -1) {
                  //piece added or removed; return true
                  return true;
            }
```

```
            //first check for ortho and diag
            // if both are set, only one of them needs to be satisfied
            int orthodiagSuccess = 0;
            if ((direction & direction_diagMask) != 0) {
                  orthodiagSuccess |= direction_diagMask;
                  if (Math.abs(x1 - x2) != Math.abs(y1 - y2)) {
                        orthodiagSuccess &= ~direction_diagMask;
                  }
            }
            if ((direction & direction_orthoMask) != 0) {
                  orthodiagSuccess |= direction_orthoMask;
                  if (x1 != x2 && y1 != y2) {
                        orthodiagSuccess &= ~direction_orthoMask;
                  }
            }
            if ((direction & (direction_diagMask | direction_orthoMask)) ==
0) {
                  //neither ortho nor diag set; mark success as true
                  orthodiagSuccess = 1;
            }

            boolean customSuccess = true;
            //check for custom directions
            if ((direction & direction_customMask) != 0) {
                  customSuccess = false;
                  int count = 0;
                  for (List<Integer> l: customDirections) {
                        int x_incr = l.get(0);
                        int y_incr = l.get(1);

                        if (x_incr == 0) {
                              if (x1 == x2) {
                                    customSuccess = true;
                                    this.customDirectionTaken = count;
                                    break;
                              }
                        } else if (y_incr == 0) {
                              if (y1 == y2) {
                                    customSuccess = true;
                                    this.customDirectionTaken = count;
                                    break;
                              }
                        } else if (((x2-x1)/x_incr == (y2-y1)/y_incr) &&
((x2-x1)/x_incr) > 0) {
                              customSuccess = true;
                              this.customDirectionTaken = count;
                              break;
                        }
                        count++;
                  }
            }
```

```
            if ((orthodiagSuccess != 0) && customSuccess) {
                  return true;
            } else {
                  return false;
            }
      }

      private boolean checkLengthJump(BGGLBoard b, int x1, int y1, int x2,
int y2) throws OperationNotSupportedException
      {
            //when we get here, we know that the direction test has passed

            if (x1 == -1 || x2 == -1) {
                  //piece added or removed; return true
                  return true;
            }

            int x_incr = 0;
            int y_incr = 0;
            //first check for ortho and diag
            if (((direction & direction_orthoMask) != 0) || ((direction &
direction_diagMask) != 0)) {
                  if (x1 == x2) {
                        y_incr = (y2 - y1)/Math.abs(y2-y1);
                  } else if (y1 == y2) {
                        x_incr = (x2 - x1)/Math.abs(x2-x1);
                  } else {
                        //move is diagonal
                        x_incr = (x2 - x1)/Math.abs(x2-x1);
                        y_incr = (y2 - y1)/Math.abs(y2-y1);
                  }
            } else if ((direction & direction_customMask) != 0) {
                  List<Integer> dir =
customDirections.get(this.customDirectionTaken);
                  x_incr = dir.get(0);
                  y_incr = dir.get(1);
            } else {
                  //direction is not set
                  return true;
            }

            int moveLength = 1;
            boolean hasJumped = false;
            int x_cur = x1+x_incr;
            int y_cur = y1+y_incr;
            for (; x_cur != x2 || y_cur != y2; x_cur += x_incr, y_cur +=
y_incr) {
                  if (!b.isInvalid(x_cur, y_cur)) {
                        moveLength++;
                        if (!b.isBlank(x_cur, y_cur)) {
                              hasJumped = true;
                        }
```

```
                  }
            }
            if (this.moveLength != moveLength) {
                  return false;
            }
            if (jump == setting.TRUE && hasJumped != true) {
                  return false;
            }
            if (jump == setting.FALSE && hasJumped != false) {
                  return false;
            }

            return true;
      }
}
```

BGGLString.java

```
package edu.bggl.semantic;

import edu.bggl.exception.OperationNotSupportedException;

public class BGGLString extends ADataType
{
      private String s;

      public BGGLString()
      {
            this("");
      }

      public BGGLString(String s)
      {
            super();
            this.s = s;
      }

      public String toString()
      {
            return s;
      }

      public void set(String s)
      {
            this.s = s;
      }

      public String getValue()
      {
            return this.s;
      }
```

```
	@Override
	public ADataType add(ADataType r)
		throws OperationNotSupportedException
	{
		return new BGGLString(this.s + r);
	}

	@Override
	public BGGLBoolean eq(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLString))
			throw new OperationNotSupportedException();

		return new BGGLBoolean(this.s.equals(((BGGLString)
r).getValue()));
	}

	@Override
	public BGGLBoolean neq(ADataType r)
		throws OperationNotSupportedException
	{
		if (! (r instanceof BGGLString))
			throw new OperationNotSupportedException();

		return new BGGLBoolean(! eq(r).getValue());
	}
}
```

Scope.java

```
package edu.bggl.semantic;

import java.util.ArrayList;
import java.util.List;

public class Scope
{
	private List<Variable> variables;
	private boolean breakpoint;

	public Scope()
	{
		variables = new ArrayList<Variable>();
		breakpoint = false;
	}

	public void add(Variable v)
	{
		this.variables.add(v);
	}
```

```
	public void remove(Variable v)
	{
		this.variables.remove(
			this.variables.indexOf(v));
	}

	public Variable get(String identifier)
	{
		return this.variables.get(this.indexOf(identifier));
	}

	public boolean contains(String identifier)
	{
		return this.indexOf(identifier) >= 0;
	}

	public int indexOf(String identifier)
	{
		for (int i = 0; i < variables.size(); i ++)
		{
			if (variables.get(i).getIdentifier().equals(identifier))
				return i;
		}
		return -1;
	}

	public void setBreakpoint(boolean b)
	{
		this.breakpoint = b;
	}

	public boolean isBreakpoint()
	{
		return this.breakpoint;
	}

	public void clear()
	{
		this.variables.clear();
	}
}
```

SymbolTable.java

```
package edu.bggl.semantic;

import java.util.ArrayList;
import java.util.List;

import edu.bggl.exception.DuplicateIdentifierException;
import edu.bggl.exception.IdentifierNotFoundException;
import edu.bggl.exception.IrregularBoardDimensionException;
```

```
import edu.bggl.exception.TypeMismatchException;

/**
 * This class is nothing more than a primitive queue of <code>Scope</code>s,
 * with convenience methods to add/get variables from the current (i.e.
front)
 * scope.
 * <p/>
 * Note that this uses (essentially) the Singleton pattern.
 *
 * @author matt
 */
public class SymbolTable
{
      private static SymbolTable me = new SymbolTable();

      public static final SymbolTable getInstance()
      {
            return me;
      }

      private List<Scope> scopes;
      private Scope currentScope;

      private Scope globalScope;

      /**
       * This is supposed to be private; this is part of the Singleton
pattern.
       */
      private SymbolTable()
      {
            scopes = new ArrayList<Scope>();
            pushScope();
            globalScope = new Scope();
      }

      public void pushScope()
      {
            currentScope = new Scope();
            scopes.add(currentScope);
      }

      public void popScope()
      {
            scopes.remove(scopes.size() - 1);
            if (scopes.size() > 0)
                  currentScope = scopes.get(scopes.size() - 1);
      }

      /**
       * Convenience method.
```

```
	 *
	 * @see #add(Variable)
	 */
	public void add(String identifier, ADataType dt)
		throws DuplicateIdentifierException
	{
		this.add(new Variable(identifier, dt));
	}

	/**
	 * Add a new variable to the current scope. Note that this does no
	 * error-checking on whether or not the variable is already defined.
	 */
	public void add(Variable v)
		throws DuplicateIdentifierException
	{
		if (globalMode)
		{
			if (globalScope.contains(v.getIdentifier()))
				throw new
DuplicateIdentifierException(v.getIdentifier());
		}
		else
		{
			if (currentScope.contains(v.getIdentifier()))
				throw new
DuplicateIdentifierException(v.getIdentifier());
		}

		addNoCheck(v);
	}

	public void addNoCheck(Variable v)
	{
		if (globalMode)
			globalScope.add(v);
		else
			currentScope.add(v);
	}

	public void addNoCheck(String identifier, ADataType dt)
	{
		this.addNoCheck(new Variable(identifier, dt));
	}

	/**
	 * Gets the first variable it finds matching the given identifier.
	 * Note that this will look through all scopes until it finds it, or
	 * will return null if it was not found.
	 * <p/>
	 * Also checks for breakpoints.
	 * <p/>
```

```
	 * Does NOT check for missing identifiers.
	 */
	public Variable getNoCheck(String identifier)
	{
		return getNoCheck(identifier, true);
	}

	public Variable getNoCheck(String identifier, boolean checkBreakpoint)
	{
		try
		{
			return get(identifier, checkBreakpoint, false);
		}
		catch (IdentifierNotFoundException ex)
		{
			return null;
		}
	}

	public Variable get(String identifier)
		throws IdentifierNotFoundException
	{
		return this.get(identifier, true);
	}

	public Variable get(String identifier, boolean checkBreakpoint)
		throws IdentifierNotFoundException
	{
		return this.get(identifier, checkBreakpoint, true);
	}

	public void set(String identifier, ADataType t)
		throws TypeMismatchException,
			IdentifierNotFoundException
	{
		if (get(identifier).getDataType().getClass().equals(t.getClass()))
		{
			get(identifier).setDataType(t);
		}
		else {
			throw new
TypeMismatchException(get(identifier).getDataType().getClass().toString(),
t.getClass().toString());
		}
	}

	public void setNoCheck(String identifier, ADataType t)
	{
		getNoCheck(identifier).setDataType(t);
	}

	private Variable get(
```

```
        String identifier,
        boolean checkBreakpoint,
        boolean throwsException)
        throws IdentifierNotFoundException
{
        for (int i = scopes.size() - 1; i >= 0; i --)
        {
                if (checkBreakpoint && scopes.get(i).isBreakpoint())
                        break;

                if (scopes.get(i).contains(identifier))
                        return scopes.get(i).get(identifier);
        }

        if (globalScope.contains(identifier))
                return globalScope.get(identifier);

        if (throwsException)
                throw new IdentifierNotFoundException(identifier);
        else
                return null;
}

public boolean contains(String identifier)
{
        for (int i = scopes.size() - 1; i >= 0; i --)
        {
                if (scopes.get(i).contains(identifier))
                        return true;
        }

        return false;
}

public void reset()
{
        scopes.clear();

        pushScope();

        globalScope.clear();

        // Default variables, ignoring exception
        try
        {
                this.globalMode = true;
                this.addNoCheck("_", new BGGLPiece("_"));
                this.addNoCheck("#", new BGGLPiece("_"));
                this.addNoCheck("board", new BGGLBoard());
                this.globalMode = false;
        }
        catch (IrregularBoardDimensionException ex)
```

```
            {
                  // can't happen.
            }
            catch (TypeMismatchException ex)
            {
                  // can't happen.
            }
      }

      /**
       * For functions.
       *
       * @author matt
       */
      public void setBreakpoint(boolean b)
      {
            currentScope.setBreakpoint(b);
      }

      /**
       * For functions.
       *
       * @param identifier
       */
      public void remove(Variable v)
      {
            currentScope.remove(v);
      }

      private boolean globalMode = false;

      public boolean isGlobalMode() {
            return globalMode;
      }

      public void setGlobalMode(boolean globalMode) {
            this.globalMode = globalMode;
      }
}
```

Variable.java

```
package edu.bggl.semantic;

/**
 * The internal representation of a variable. This is used by
 * <code>SymbolTable</code>.
 *
 * @author matt
 */
public class Variable
{
```

```
	private String identifier;
	private ADataType dataType;

	public Variable(String identifier, ADataType dataType)
	{
		this.identifier = identifier;
		this.dataType = dataType;
	}

	public ADataType getDataType()
	{
		return dataType;
	}

	public void setDataType(ADataType dataType)
	{
		this.dataType = dataType;
	}

	public String getIdentifier()
	{
		return identifier;
	}
	public void setIdentifier(String identifier)
	{
		this.identifier = identifier;
	}
	/**
	 * 2 variables are equal iff their identifiers match; note that the
	 * actual value of the variable is irrelevant, as is its scope. This
	 * is to make it easier to use by <code>SymbolTable</code>.
	 */
	public boolean equals(Object o)
	{
		String thatId = null;
		if (o instanceof Variable)
			thatId = ((Variable) o).getIdentifier();

		if (o instanceof String)
			thatId = (String) o;

		if (thatId == null)
			return false;

		return this.identifier.equals(thatId);
	}

	public int hashCode()
	{
		return this.identifier.hashCode();
	}
}
```

AllTests.java

```
package edu.bggl.test;

import junit.framework.JUnit4TestAdapter;
import junit.framework.Test;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

import edu.bggl.test.lexparse.invalid.InvalidArithmeticAdd;
import edu.bggl.test.lexparse.invalid.InvalidArithmeticDiv;
import edu.bggl.test.lexparse.invalid.InvalidArithmeticMod;
import edu.bggl.test.lexparse.invalid.InvalidArithmeticMult;
import edu.bggl.test.lexparse.invalid.InvalidArithmeticSub;
import edu.bggl.test.lexparse.invalid.InvalidDeclarationsBeginDigit;
import edu.bggl.test.lexparse.invalid.InvalidDeclarationsBeginUnderscore;
import edu.bggl.test.lexparse.invalid.InvalidDeclarationsUnderscoreOnly;
import edu.bggl.test.lexparse.invalid.InvalidExpressions;
import edu.bggl.test.lexparse.invalid.InvalidForStatement;
import edu.bggl.test.lexparse.invalid.MissingGameBraces;
import edu.bggl.test.lexparse.invalid.SmallestLPInvalid;
import edu.bggl.test.lexparse.valid.Assignment;
import edu.bggl.test.lexparse.valid.BoardInitialize;
import edu.bggl.test.lexparse.valid.Comments;
import edu.bggl.test.lexparse.valid.EmptyProgram;
import edu.bggl.test.lexparse.valid.EmptyStatements;
import edu.bggl.test.lexparse.valid.Expressions;
import edu.bggl.test.lexparse.valid.ForStatement;
import edu.bggl.test.lexparse.valid.FuncCall;
import edu.bggl.test.lexparse.valid.FuncDecl;
import edu.bggl.test.lexparse.valid.IfStatement;
import edu.bggl.test.lexparse.valid.InitializedDeclarations;
import edu.bggl.test.lexparse.valid.MissingTurn;
import edu.bggl.test.lexparse.valid.MoveAssignment;
import edu.bggl.test.lexparse.valid.PieceAssignment;
import edu.bggl.test.lexparse.valid.PlayerAssignment;
import edu.bggl.test.lexparse.valid.ReadInput;
import edu.bggl.test.lexparse.valid.RuleCall;
import edu.bggl.test.lexparse.valid.RuleDecl;
import edu.bggl.test.lexparse.valid.SixEmptyStatements;
import edu.bggl.test.lexparse.valid.SmallestLPValid;
import edu.bggl.test.lexparse.valid.ValidDeclarations;
import edu.bggl.test.lexparse.valid.ValidStrings;
import edu.bggl.test.lexparse.valid.WhileStatement;
import edu.bggl.test.print.valid.ArrayAssignment;
import edu.bggl.test.print.valid.ArrayIndex;
import edu.bggl.test.print.valid.BlockScopeVariables;
import edu.bggl.test.print.valid.BoardLocationGet;
import edu.bggl.test.print.valid.BoardProperty;
```

```
import edu.bggl.test.print.valid.ComputeGCD;
import edu.bggl.test.print.valid.Expressions2;
import edu.bggl.test.print.valid.ForLoop;
import edu.bggl.test.print.valid.ForScopeVariables;
import edu.bggl.test.print.valid.FunctionCallByValue;
import edu.bggl.test.print.valid.FunctionReturn;
import edu.bggl.test.print.valid.FunctionReturn2;
import edu.bggl.test.print.valid.FunkyForInts;
import edu.bggl.test.print.valid.GetRowColDiag;
import edu.bggl.test.print.valid.GlobalBoard;
import edu.bggl.test.print.valid.GlobalBoardOperations;
import edu.bggl.test.print.valid.GlobalScope;
import edu.bggl.test.print.valid.IfScoping;
import edu.bggl.test.print.valid.InvalidScoping3;
import edu.bggl.test.print.valid.JumpMoveTest;
import edu.bggl.test.print.valid.NegativeNumbers;
import edu.bggl.test.print.valid.NestedForWhile;
import edu.bggl.test.print.valid.NestedFuncCall;
import edu.bggl.test.print.valid.PlayerEq;
import edu.bggl.test.print.valid.Print;
import edu.bggl.test.print.valid.PrintAssignments;
import edu.bggl.test.print.valid.Procedure;
import edu.bggl.test.print.valid.ReturnTermination;
import edu.bggl.test.print.valid.RuleGetsMove;
import edu.bggl.test.print.valid.TestPredBadRegions;
import edu.bggl.test.print.valid.TestPredJumpEmpty;
import edu.bggl.test.print.valid.TestPredOrthoDiag;
import edu.bggl.test.print.valid.TestPredicate;
import edu.bggl.test.print.valid.TestPredicateAdv;
import edu.bggl.test.print.valid.TypeInitializations;
import edu.bggl.test.print.valid.While;
import edu.bggl.test.print.valid.WhileBreak;
import edu.bggl.test.print.valid.WhileScoping;
import edu.bggl.test.semantic.invalid.Assign1;
import edu.bggl.test.semantic.invalid.Assign2;
import edu.bggl.test.semantic.invalid.Assign3;
import edu.bggl.test.semantic.invalid.Assign4;
import edu.bggl.test.semantic.invalid.Assign5;
import edu.bggl.test.semantic.invalid.At1;
import edu.bggl.test.semantic.invalid.At2;
import edu.bggl.test.semantic.invalid.At3;
import edu.bggl.test.semantic.invalid.At4;
import edu.bggl.test.semantic.invalid.At5;
import edu.bggl.test.semantic.invalid.BoardLocArgs1;
import edu.bggl.test.semantic.invalid.BoardLocArgs2;
import edu.bggl.test.semantic.invalid.BoardLocArgs3;
import edu.bggl.test.semantic.invalid.BoardLocArgs4;
import edu.bggl.test.semantic.invalid.BoardLocArgs5;
import edu.bggl.test.semantic.invalid.BoardSubsetArgs1;
import edu.bggl.test.semantic.invalid.BoardSubsetArgs2;
import edu.bggl.test.semantic.invalid.BoardSubsetArgs3;
import edu.bggl.test.semantic.invalid.BoardSubsetArgs4;
```

```
import edu.bggl.test.semantic.invalid.BoardSubsetArgs5;
import edu.bggl.test.semantic.invalid.DeclInit1;
import edu.bggl.test.semantic.invalid.DeclInit2;
import edu.bggl.test.semantic.invalid.DeclInit3;
import edu.bggl.test.semantic.invalid.DeclInit4;
import edu.bggl.test.semantic.invalid.DeclInit5;
import edu.bggl.test.semantic.invalid.DupIdInScope;
import edu.bggl.test.semantic.invalid.DupIdInScope2;
import edu.bggl.test.semantic.invalid.ForType;
import edu.bggl.test.semantic.invalid.ForType2;
import edu.bggl.test.semantic.invalid.ForType3;
import edu.bggl.test.semantic.invalid.FuncArgNumber;
import edu.bggl.test.semantic.invalid.FuncArgType;
import edu.bggl.test.semantic.invalid.FuncNotDefined;
import edu.bggl.test.semantic.invalid.IdNotDefined;
import edu.bggl.test.semantic.invalid.IfType;
import edu.bggl.test.semantic.invalid.InitBoard1;
import edu.bggl.test.semantic.invalid.InitBoard2;
import edu.bggl.test.semantic.invalid.IrregularBoardSize1;
import edu.bggl.test.semantic.invalid.IrregularBoardSize2;
import edu.bggl.test.semantic.invalid.MoveApply1;
import edu.bggl.test.semantic.invalid.MoveApply2;
import edu.bggl.test.semantic.invalid.ReturnMistype;
import edu.bggl.test.semantic.invalid.TestArgs;
import edu.bggl.test.semantic.invalid.WhileType;
import edu.bggl.test.semantic.valid.Scoping;

/**
 * Add new test classes by adding to the "@SuiteClasses" list.
 *
 * @author matt
 */
@RunWith(Suite.class)
@SuiteClasses({

      // lexer/parser tests
**************************************************

      // valid
      Assignment.class,
      BoardInitialize.class,
      BoardProperty.class,
      Comments.class,
    EmptyProgram.class,

    EmptyStatements.class,
    Expressions.class,
    ForStatement.class,
      FuncCall.class,
      FuncDecl.class,

    IfStatement.class,
```

```
    InitializedDeclarations.class,
    MissingTurn.class,
    MoveAssignment.class,
    PieceAssignment.class,

    PlayerAssignment.class,
      ReadInput.class,
    RuleCall.class,
    RuleDecl.class,
      SixEmptyStatements.class,

      SmallestLPValid.class,
      ValidDeclarations.class,
      ValidStrings.class,
      WhileStatement.class,

      // invalid
      InvalidArithmeticAdd.class,
      InvalidArithmeticMult.class,
      InvalidArithmeticSub.class,
      InvalidArithmeticMod.class,
      InvalidArithmeticDiv.class,

      InvalidExpressions.class,
      InvalidForStatement.class,
      InvalidDeclarationsBeginDigit.class,
      InvalidDeclarationsBeginUnderscore.class,
      InvalidDeclarationsUnderscoreOnly.class,

      MissingGameBraces.class,
      SmallestLPInvalid.class,

      // semantic tests
*********************************************************

      // valid
      Scoping.class,

      // invalid
      Assign1.class,
      Assign2.class,
      Assign3.class,
      Assign4.class,
      Assign5.class,

      At1.class,
      At2.class,
      At3.class,
      At4.class,
      At5.class,

      BoardLocArgs1.class,
```

```
      BoardLocArgs2.class,
      BoardLocArgs3.class,
      BoardLocArgs4.class,
      BoardLocArgs5.class,

      BoardSubsetArgs1.class,
      BoardSubsetArgs2.class,
      BoardSubsetArgs3.class,
      BoardSubsetArgs4.class,
      BoardSubsetArgs5.class,

      DeclInit1.class,
      DeclInit2.class,
      DeclInit3.class,
      DeclInit4.class,
      DeclInit5.class,

      DupIdInScope.class,
      DupIdInScope2.class,
      ForType.class,
      ForType2.class,
      ForType3.class,

      FuncArgNumber.class,
      FuncArgType.class,
      FuncNotDefined.class,
      IdNotDefined.class,
      IfType.class,

      InitBoard1.class,
      InitBoard2.class,
      IrregularBoardSize1.class,
      IrregularBoardSize2.class,
      MoveApply1.class,

      MoveApply2.class,
      WhileType.class,

      // print tests
*************************************************************

      // valid
      ArrayAssignment.class,
      ArrayIndex.class,
      BlockScopeVariables.class,
      BoardLocationGet.class,
      BoardProperty.class,

      ComputeGCD.class,
      Expressions2.class,
      ForLoop.class,
      ForScopeVariables.class,
```

```
      FunctionCallByValue.class,

      FunctionReturn.class,
      FunctionReturn2.class,
      FunkyForInts.class,
      GetRowColDiag.class,
      GlobalBoard.class,

      GlobalBoardOperations.class,
      GlobalScope.class,
      IfScoping.class,
      InvalidScoping3.class,
      JumpMoveTest.class,

      NegativeNumbers.class,
      NestedForWhile.class,
      NestedFuncCall.class,
      PlayerEq.class,
      Print.class,

      PrintAssignments.class,
      Procedure.class,
      ReturnTermination.class,
      RuleGetsMove.class,
      ReturnMistype.class,

      TestArgs.class,
      TestPredBadRegions.class,
      TestPredicate.class,
      TestPredicateAdv.class,
      TestPredJumpEmpty.class,

      TestPredOrthoDiag.class,
      TypeInitializations.class,
      While.class,
//    WhileBreak.class,
      WhileScoping.class,

      // invalid

      // end
**********************************************************************
})

public class AllTests
{
      public static Test suite()
      {
            return new JUnit4TestAdapter(AllTests.class);
      }
}
```

ALexParseTestInvalid.java

```
package edu.bggl.test;

import static org.junit.Assert.assertTrue;

import java.io.FileNotFoundException;
import java.io.FileReader;

import org.junit.Test;

import antlr.RecognitionException;
import antlr.TokenStreamException;
import edu.bggl.grammar.BGGLLexer;
import edu.bggl.grammar.BGGLParser;

/**
 * A fake test. Mainly used as a test in testing the framework (scripts, etc).
 *
 * @author matt
 */
public class ALexParseTestInvalid extends ATestInvalid
{
      /**
       * This is where the testing code would go.
       * <p/>
       * Note that the method name can be anything.
```

```
 * <p/>
 * Note that to call <code>assertTrue</code>, you need a STATIC import.
 * <p/>
 * If the test passes, you can either do nothing, or you can assertTrue.
 * If the test fails, you MUST assertFalse.
 */
@Test public void isValid()
      throws FileNotFoundException
{
      try
      {
            FileReader f =
                  new FileReader(
                        "lexparse/invalid/" +
                        this.getClass().getSimpleName() + ".bggl");
            BGGLParser parser = new BGGLParser(new BGGLLexer(f));
            parser.main_program();
      }
      catch (RecognitionException e)
      {
            assertTrue(true);
            return;
      }
      catch (TokenStreamException e)
      {
            assertTrue(true);
```

```
                return;
            }

            assertTrue(false);
      }
}
```

ALexParseTestValid.java

```
package edu.bggl.test;

import static org.junit.Assert.assertTrue;

import java.io.FileReader;

import org.junit.Test;

import edu.bggl.grammar.BGGLLexer;
import edu.bggl.grammar.BGGLParser;

/**
 * A fake test. Mainly used as a test in testing the framework (scripts, etc).
 *
 * @author matt
 */
public class ALexParseTestValid extends ATestValid
{
      /**
```

```
	 * This is where the testing code would go.
	 * <p/>
	 * Note that the method name can be anything.
	 * <p/>
	 * Note that to call <code>assertTrue</code>, you need a STATIC import.
	 * <p/>
	 * If the test passes, you can either do nothing, or you can assertTrue.
	 * If the test fails, you MUST assertFalse.
	 */
	@Test public void isValid()
		throws Exception
	{
		FileReader f =
			new FileReader(
				"lexparse/valid/" +
				this.getClass().getSimpleName() + ".bggl");
		BGGLParser parser = new BGGLParser(new BGGLLexer(f));
		parser.main_program();

		assertTrue(true);
	}
}
```

APrintTestValid.java

```
package edu.bggl.test;

import static org.junit.Assert.assertTrue;

import java.io.FileReader;
import java.util.ArrayList;
```

```
import java.util.List;

import org.junit.Test;

import edu.bggl.grammar.BGGLLexer;
import edu.bggl.grammar.BGGLParser;
import edu.bggl.interpreter.Interpreter;

/**
 * The base class for a valid test that prints output to the console.
Override
 * <code>getCorrectOutput</code> to specify the correct output.
 *
 * @author matt
 */
public abstract class APrintTestValid extends ATestValid
{
	/**
	 * Loads the sample program, and runs it through the lexer and the
parser.
	 * <p/>
	 * Note that the method name can be anything.
	 *
	 * @throws Exception If anything goes wrong. Any exception constitutes a
	 * failure.
	 */
	@Test public void isValid()
		throws Exception
	{
		FileReader f =
			new FileReader(
				"print/valid/" +
				this.getClass().getSimpleName() + ".bggl");
		BGGLParser parser = new BGGLParser(new BGGLLexer(f));
		parser.main_program();

		Interpreter interpreter = new Interpreter(
			null,
			parser.getAST(),
			true);
		interpreter.run();

		if (isEqual(interpreter.getOutBuffer(), this.getCorrectOutput()))
		{
			assertTrue(true);
		}
		else
		{
			assertTrue(false);
		}
	}
```

```
	private boolean isEqual(List<String> a, List<String> b)
	{
		if (a.size() != b.size())
		{
			return false;
		}


		for (int i = 0; i < a.size(); i ++)
		{
			if (! a.get(i).equals(b.get(i)))
				return false;
		}

		return true;
	}

	/**
	 * Override this if necessary. By default this tests against no output.
	 *
	 * @return The output in a <code>List</code>, where each line
represents one line of the
	 * output. So if your program was supposed to output "1\n2", the result
from this method
	 * should be a 2 element list.
	 */
	protected List<String> getCorrectOutput()
	{
		List<String> correct = new ArrayList<String>();

		return correct;
	}
}
```

ASemanticTestInvalid.java

```
package edu.bggl.test;

import static org.junit.Assert.assertTrue;

import java.io.FileNotFoundException;
import java.io.FileReader;

import org.junit.Test;

import antlr.RecognitionException;
import antlr.TokenStreamException;
import edu.bggl.grammar.BGGLLexer;
import edu.bggl.grammar.BGGLParser;
import edu.bggl.grammar.BGGLWalker;
import edu.bggl.interpreter.Interpreter;
```

```
/**
 * A fake test. Mainly used as a test in testing the framework (scripts, etc).
 *
 * @author matt
 */
public class ASemanticTestInvalid extends ATestInvalid
{
	/**
	 * This is where the testing code would go.
	 * <p/>
	 * Note that the method name can be anything.
	 * <p/>
	 * Note that to call <code>assertTrue</code>, you need a STATIC import.
	 * <p/>
	 * If the test passes, you can either do nothing, or you can assertTrue.
	 * If the test fails, you MUST assertFalse.
	 */
	@Test public void isValid()
		throws FileNotFoundException
	{
		try
		{
			FileReader f =
				new FileReader(
					"semantic/invalid/" +
					this.getClass().getSimpleName() + ".bggl");
			BGGLParser parser = new BGGLParser(new BGGLLexer(f));
			parser.main_program();

			Interpreter interpreter = new Interpreter(
				null,
				parser.getAST(),
				false);
			interpreter.run();
		}
		catch (RecognitionException e)
		{
			assertTrue(true);
			return;
		}
		catch (TokenStreamException e)
		{
			assertTrue(false);
			return;
		}

		assertTrue(false);
	}
}
```

ASemanticTestValid.java

```
package edu.bggl.test;

import static org.junit.Assert.assertTrue;

import java.io.FileReader;

import org.junit.Test;

import edu.bggl.grammar.BGGLLexer;
import edu.bggl.grammar.BGGLParser;
import edu.bggl.grammar.BGGLWalker;
import edu.bggl.interpreter.Interpreter;

/**
 * The base class for a valid static semantic analysis test.
 *
 * @author matt
 */
public abstract class ASemanticTestValid extends ATestValid
{
	/**
	 * Loads the sample program, and runs it through the lexer and the
parser.
	 * <p/>
	 * Note that the method name can be anything.
	 *
	 * @throws Exception If anything goes wrong. Any exception constitutes a
	 * failure.
	 */
	@Test public void isValid()
		throws Exception
	{
		FileReader f =
			new FileReader(
				"semantic/valid/" +
				this.getClass().getSimpleName() + ".bggl");
		BGGLParser parser = new BGGLParser(new BGGLLexer(f));
		parser.main_program();

		Interpreter interpreter = new Interpreter(
			null,
			parser.getAST(),
			true);
		interpreter.run();

		assertTrue(true);
	}
}
```

ATestInvalid.java

```
package edu.bggl.test;

import static org.junit.Assert.assertTrue;

import java.io.FileNotFoundException;
import java.io.FileReader;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import antlr.RecognitionException;
import antlr.TokenStreamException;
import edu.bggl.grammar.BGGLLexer;
import edu.bggl.grammar.BGGLParser;
import edu.bggl.semantic.SymbolTable;

/**
 * Base class of all INVALID tests.
 *
 * @author matt
 */
public abstract class ATestInvalid
{
	/**
	 * This is where "build up" code would go.
	 * <p/>
	 * Note that the method name can be anything.
	 */
	@Before public void preTestCode()
	{
		SymbolTable.getInstance().reset();
	}

	/**
	 * Loads the sample program, and runs it through the lexer and the
parser.
	 * Note that the test is SUPPOSED to throw an exception.
	 *
	 * @throws Exception If anything goes wrong. Note that NO exception
	 * constitutes a failure (unless it is
<code>FileNotFoundException</code>).
	 */
	@Test public void isValid()
		throws FileNotFoundException
	{
		try
		{
			FileReader f =
				new FileReader(
```

```
                              this.getClass().getSimpleName() + ".bggl");
                  BGGLParser parser = new BGGLParser(new BGGLLexer(f));
                  parser.main_program();
            }
            catch (RecognitionException e)
            {
                  assertTrue(true);
                  return;
            }
            catch (TokenStreamException e)
            {
                  assertTrue(true);
                  return;
            }

            assertTrue(false);
      }

      /**
       * This is where "tear down" code would go.
       * <p/>
       * Note that the method name can be anything.
       */
      @After public void postTestCode()
      {
      }
}
```

ATestValid.java

```
package edu.bggl.test;

import static org.junit.Assert.assertTrue;

import java.io.FileReader;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import edu.bggl.grammar.BGGLLexer;
import edu.bggl.grammar.BGGLParser;
import edu.bggl.semantic.SymbolTable;

/**
 * Base case of all VALID tests.
 *
 * @author matt
 */
public abstract class ATestValid
{
      /**
```

```
	 * This is where "build up" code would go.
	 * <p/>
	 * Note that the method name can be anything.
	 */
	@Before public void preTestCode()
	{
		SymbolTable.getInstance().reset();
	}

	/**
	 * Loads the sample program, and runs it through the lexer and the
parser.
	 * <p/>
	 * Note that the method name can be anything.
	 *
	 * @throws Exception If anything goes wrong. Any exception constitutes a
	 * failure.
	 */
	@Test public void isValid()
		throws Exception
	{
		System.out.println(this.getClass().getSimpleName());

		FileReader f =
			new FileReader(
				this.getClass().getSimpleName() + ".bggl");
		BGGLParser parser = new BGGLParser(new BGGLLexer(f));
		parser.main_program();

		assertTrue(true);
	}

	/**
	 * This is where "tear down" code would go.
	 * <p/>
	 * Note that the method name can be anything.
	 */
	@After public void postTestCode()
	{
	}
}
```

build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project
    name="BGGL"
    basedir="../"
    default="gen-all-run-tests"
>

    <description></description>
```

```
<property name="dir.grammar" location="grammar"/>
<property name="grammar.lexer_parser" value="BGGL.g"/>
<property name="grammar.semantic" value="SemanticAnalyzer.g"/>
<property name="grammar.walker" value="BGGLWalker.g"/>
<fileset
    id="files.grammar.antlr"
    dir="${dir.grammar}"
>
    <include name="BGGL.g"/>
    <include name="SemanticAnalyzer.g"/>
    <include name="BGGLWalker.g"/>
</fileset>
<fileset
    id="files.grammar.java"
    dir="${dir.grammar}"
>
    <include name="BGGLLexer.java"/>
    <include name="BGGLParser.java"/>
    <include name="BGGLTokenTypes.java"/>
    <include name="SemanticAnalyzer.java"/>
    <include name="SemanitcAnalyzerTokenTypes.java"/>
    <include name="BGGLWalker.java"/>
    <include name="BGGLWalkerTokenTypes.java"/>
</fileset>

<property name="BIN_DIR" location="bin"/>
<property name="SRC_DIR" location="src"/>
<property name="dir.classes" location="classes"/>

<property name="TESTS_DIR" location="tests"/>

<property name="PKG_GRAMMAR_DIR" value="edu/bggl/grammar"/>

<property name="pkg.tests" value="edu.bggl.test"/>
<property name="pkg.tests.lp" value="edu.bggl.test.lexparse"/>

<property name="TOOLS_HOME" location="tools"/>

<property name="ANTLR_HOME" location="${TOOLS_HOME}/antlr-2.75"/>
<property name="ANTLR_BIN" location="${ANTLR_HOME}/bin"/>
<property name="ANTLR_BAT" location="${ANTLR_BIN}/antlr.bat"/>
<property name="ANTLR_SBIN" location="${ANTLR_HOME}/sbin"/>
<property name="ANTLR_PY" location="${ANTLR_SBIN}/pyantlr.py"/>
<property name="ANTLR_SH" location="${BIN_DIR}/compile-grammar.sh"/>
<property name="ANTLR_LIB" location="${ANTLR_HOME}/lib"/>
<property name="ANTLR_JAR" location="${ANTLR_LIB}/antlr.jar"/>

<property name="JUNIT_HOME" location="${TOOLS_HOME}/junit-4.1"/>
<property name="JUNIT_JAR" value="${JUNIT_HOME}/junit-4.1.jar"/>

<target
```

```
    name="compile-compiler"
>
    <javac
        srcdir="${SRC_DIR}"
        destdir="${dir.classes}"
        source="1.5"
        target="1.5"
    >
        <!-- "Generics" warnings
        <compilerarg
            value="-Xlint:unchecked"
        />
        -->
        <classpath>
            <pathelement location="${TESTS_DIR}"/>
            <pathelement location="${JUNIT_JAR}"/>
            <pathelement location="${ANTLR_JAR}"/>
            <pathelement location="${dir.classes}"/>
        </classpath>
    </javac>
</target>

<target
    name="run-tests"
    depends="compile-compiler"
>
    <!-- This is stupid, but I can't get it working with classpaths. -->
    <copy
        todir="${dir.classes}"
    >
        <!--
        <fileset
            dir="${TESTS_DIR}"
            includes="**/*.bggl"
        />
              includes="lexparse,semantic,print"
        -->
        <fileset
            dir="${TESTS_DIR}"
              includes="**/*.bggl"
        />
    </copy>

    <java
        classname="org.junit.runner.JUnitCore"
        fork="true"
        dir="${dir.classes}"
    >
        <classpath>
            <pathelement location="${TESTS_DIR}"/>
            <pathelement location="${JUNIT_JAR}"/>
            <pathelement location="${ANTLR_JAR}"/>
```

```
            <pathelement location="${dir.classes}"/>
        </classpath>
        <!--
        <arg value="${pkg.tests.lp}.AllTests"/>
        -->
        <arg value="${pkg.tests}.AllTests"/>
    </java>

    <!-- Now we have to clean up our mess. -->
    <delete
    >
        <fileset
            dir="${dir.classes}"
            includes="**/*.bggl"
        />
        <!--
        <fileset
            dir="${dir.classes}"
            includes="lexparse,semantic,print"
        />
        -->
    </delete>
</target>

<target
    name="compile-grammar"
>
    <exec
        executable="${ANTLR_BAT}"
        dir="${dir.grammar}"
        os="Windows 2000, Windows XP"
    >
        <arg
            value="${grammar.lexer_parser}"
        />
    </exec>
   <exec
        executable="${ANTLR_SH}"
        dir="${dir.grammar}"
        os="Linux, unix, Mac OS X"
    >
        <arg
            value="${dir.grammar}/${grammar.lexer_parser}"
        />
    </exec>
    <move
        todir="${SRC_DIR}/${PKG_GRAMMAR_DIR}"
    >
        <fileset
            dir="${dir.grammar}"
        >
            <include name="BGGLLexer.java"/>
```

```
                <include name="BGGLParser.java"/>
                <include name="BGGLTokenTypes.java"/>
            </fileset>
        </move>
      <!-- Can't delete this!
        <delete
            file="${dir.grammar}/BGGLTokenTypes.txt"
        />
        -->
        <delete
            file="${dir.grammar}/BGGLParserTokenTypes.txt"
        />
    </target>
<!--
    <target
        name="compile-semantic-analyzer"
        description="Generates semantic analyzer."
    >
        <exec
            executable="${ANTLR_BAT}"
            dir="${dir.grammar}"
            os="Windows 2000, Windows XP"
        >
            <arg
                value="${grammar.semantic}"
            />
        </exec>
        <exec
            executable="${ANTLR_SH}"
            dir="${dir.grammar}"
            os="Linux, unix, Mac OS X"
        >
            <arg
                value="${grammar.semantic}"
            />
        </exec>
        <move
            todir="${SRC_DIR}/${PKG_GRAMMAR_DIR}"
        >
            <fileset
                dir="${dir.grammar}"
            >
                <include name="SemanticAnalyzer.java"/>
                <include name="SemanticAnalyzerTokenTypes.java"/>
            </fileset>
        </move>
        <delete
            file="${dir.grammar}/SemanticAnalyzerTokenTypes.txt"
        />
    </target>
-->
    <target
```

```
    name="compile-walker"
        description="Generates walker for interpretation."
>
    <exec
        executable="${ANTLR_BAT}"
        dir="${dir.grammar}"
        os="Windows 2000, Windows XP"
    >
        <arg
            value="${grammar.walker}"
        />
    </exec>
    <exec
        executable="${ANTLR_SH}"
        dir="${dir.grammar}"
        os="Linux, unix, Mac OS X"
    >
        <arg
            value="${grammar.walker}"
        />
    </exec>
    <move
        todir="${SRC_DIR}/${PKG_GRAMMAR_DIR}"
    >
        <fileset
            dir="${dir.grammar}"
        >
            <include name="BGGLWalker.java"/>
            <include name="BGGLWalkerTokenTypes.java"/>
        </fileset>
    </move>
    <delete
        file="${dir.grammar}/BGGLWalkerTokenTypes.txt"
    />
</target>

<target
    name="gen-all"
    description="Generate all Java classes from ANTLR files."
>
    <antcall
        target="compile-grammar"
    />

    <antcall
        target="compile-walker"
    />
    <antcall
        target="compile-compiler"
    />
</target>
```

```
    <target
        name="gen-all-run-tests"
        description="Generate all Java classes from ANTLR files and run unit
tests."
    >
        <antcall
            target="gen-all"
        />
        <antcall
            target="run-tests"
        />
    </target>

    <target
        name="demo"
        description="Demo."
    >
        <java
            classname="edu.bggl.interpreter.Demo"
            fork="true"
            dir="${dir.classes}"
        >
            <classpath>
                <pathelement location="${TESTS_DIR}"/>
                <pathelement location="${JUNIT_JAR}"/>
                <pathelement location="${ANTLR_JAR}"/>
                <pathelement location="${dir.classes}"/>
            </classpath>
        </java>
    </target>

</project>
```

TESTS/LEXPARSE/INVALID/SmallestLPInvalid.bggl

```
game
```

TESTS/LEXPARSE/VALID/ForStatement.bggl

```
game
{
      int x;
      for (x = 1 to 5) {
            print x;
      }
}
```

TESTS/PRINT/VALID/ForLoop.bggl

```
game {

      int x = 4;
```

```
	for (x = x to 5) {
		print x;
	}

	boolean b1 = true;
	boolean b2 = (x <= 5);
	print b2;

	while (b2) {

		if (x>10) {
			b2 = false;
			print x;
			int u = 8;
		} else {
			x = x+1;
		}

	}

}
```

TESTS/SEMANTIC/INVALID/BoardSubsetArgs3.bggl

```
game
{
	piece a;
	piece b;
	piece c;
	piece d;

	board = <[a,b][c,d]>;

	string s;

	print <_s>;
}
```

TESTS/SEMANTIC/VALID/Scoping.bggl

```
game
{
	int i = 1;

	turn main
	{
		int j = 2;
	}

	int j = 3;
}
```

8.1 Project Log

An exhaustive project log can be retrieved at the following site....
http://www.thmttch.dreamhosters.com/cs4115/trunk
username: steven | matt | vitaliy | hrishikesh
password: plt2006

165		12/19/06 6:52 PM	steven	[Vitaliy]: removed all commented out code and all the "hack" comments
164		12/19/06 2:19 PM	hrishikesh	Added source square checking and modified ChineseCheckers to get rid of that extra explicit check[...]
163		12/19/06 9:21 AM	matt	Added a Demo class.[...]
162		12/19/06 9:05 AM	steven	FINAL SLIDES...check them over before I send to Edwards.
161		12/19/06 8:44 AM	matt	Finally passed BlockScopeVariables test.[...]
160		12/19/06 8:07 AM	matt	Cleaned up chinese checkers a little.[...]
159		12/19/06 6:02 AM	vitaliy	go play chinese fucking checkers!!!![...]
156		12/19/06 3:34 AM	matt	Added type checking for return types and the test predicate.[...]
155		12/19/06 3:29 AM	hrishikesh	Fixed custom direction bug about identifying the correct direction[...]
154		12/19/06 3:10 AM	matt	Added type checking for board location, board subset, and array at.[...]
153		12/19/06 2:54 AM	vitaliy	updated chinese checkers
152		12/19/06 2:52 AM	vitaliy	removed extra comment terminator
151		12/19/06 2:48 AM	matt	Fixed board dimension checking.[...]
150		12/19/06 2:29 AM	matt	Negative numbers and negation. Yay.[...]
149		12/19/06 2:27 AM	vitaliy	updated tic-tac-toe and chinese checkers
148		12/19/06 2:02 AM	matt	More static semantic analysis:[...]
147		12/19/06 1:03 AM	hrishikesh	Fixed and simplified custom direction checking[...]
146		12/19/06 12:56 AM	matt	Added type checking for if statements and for and while loops.[...]
145		12/19/06 12:55 AM	vitaliy	fixed tic-tac-toe
144		12/19/06 12:21 AM	hrishikesh	Fixed TestPredicateAdv
143		12/18/06 11:54 PM	matt	We now do static semantic analysis on function arguments.[...]
142		12/18/06 11:13 PM	matt	We now have basic static semantic analysis for type mismatches on assignments.[...]

141		12/18/06 10:28 PM	matt	Fixed bug with input and "COMMA expected".
140		12/18/06 8:01 PM	hrishikesh	Fixed an orthodiag bug[...]
139		12/18/06 7:36 PM	vitaliy	fixed globalboard test[...]
138		12/18/06 7:02 PM	matt	Added another test for jumps and empty.[...]
137		12/18/06 6:26 PM	matt	Added more test predicate tests.[...]
136		12/18/06 5:44 PM	vitaliy	modified displaying of board coordinates[...]
135		12/18/06 5:39 PM	hrishikesh	Added more checking for invalid regions[...]
134		12/18/06 5:27 PM	hrishikesh	Added methods on BGGLBoard to test for blank and invalid regions[...]
133		12/18/06 5:04 PM	matt	Added another test of test predicate.[...]
132		12/18/06 4:52 PM	hrishikesh	Most of Rule stuff should work... working on a few more bugs.[...]
131		12/18/06 3:33 PM	matt	The "test" predicate now works (but you must specify arguments).[...]
130		12/18/06 1:37 PM	hrishikesh	Added Rule handling stuff in Interpreter[...]
129		12/18/06 5:54 AM	matt	Added new implicit variable "thismove" for rule blocks.[...]
128		12/18/06 5:22 AM	matt	Added 2 more tests, one of which fails (and so needs to be fixed).[...]
126		12/18/06 4:25 AM	matt	Ladies and gentlemen, come on up! Would you like to play some tic-tac-toe... written in motherfuckin' BGGL![...]
125		12/18/06 3:20 AM	matt	Huge amount of new tests and bugfixes. Almost got tic-tac-toe working.[...]
124		12/17/06 11:46 PM	matt	Board location test.
123		12/17/06 11:02 PM	matt	Began integration with Hrishikesh's code.[...]
122		12/17/06 10:55 PM	hrishikesh	Removed MoveNotSetException for rule - guaranteed never to happen due to grammar[...]
121		12/17/06 10:52 PM	hrishikesh	fixed crappy compile time error[...]
120		12/17/06 10:49 PM	hrishikesh	Added setMove method to Rule and checks for custom directions[...]
119		12/17/06 9:31 PM	vitaliy	- changed structure of symbolTable[...]

118		12/17/06 8:47 PM	matt	Fixed another scoping bug.[...]
117		12/17/06 8:42 PM	hrishikesh	Fixed the obscenely large number of compile errors[...]
116		12/17/06 8:05 PM	hrishikesh	Updated Rule to check for movements. Some stuff still remaining.[...]
115		12/17/06 8:00 PM	vitaliy	- unfinished tic tac toe test[...]
114		12/17/06 7:38 PM	steven	FunkyForInts does assignment in the wrong sequence...
113		12/17/06 7:08 PM	matt	Fixed scoping issue in while loops.[...]
112		12/17/06 7:02 PM	steven	more tests
111		12/17/06 6:59 PM	vitaliy	added parsing + test for player initialization
110		12/17/06 6:48 PM	steven	more tests, mostly lex/pars
109		12/17/06 6:44 PM	vitaliy	added syntax, parsing and test for getting rows, cols and diagonals[...]
108		12/17/06 6:00 PM	steven	Added applyMove(BGGLMove input_move) method to BGGLBoard[...]
107		12/17/06 5:59 PM	matt	Moved loops into interpreter.[...]
106		12/17/06 5:17 PM	steven	
105		12/17/06 4:11 PM	steven	
104		12/17/06 4:00 PM	vitaliy	- stricter syntax for loops (id or integer only)[...]
103		12/17/06 3:34 PM	steven	
102		12/17/06 1:21 PM	matt	Fixed yet another bug with function returns and termination.[...]
101		12/17/06 2:38 AM	matt	Added programs directory, with a primes program.[...]
100		12/17/06 12:34 AM	matt	Removed more crap we don't care about, and shouldn't be under source control.[...]
99		12/16/06 9:58 PM	matt	Cleaned up grammar; no changes.[...]
98		12/16/06 7:56 PM	steven	final doc in progress.
97		12/16/06 6:16 PM	steven	I tried adding the negation unary operator, you know, -<number>.[...]

96		12/16/06 3:32 PM	vitaliy	added walking and initialization of BGGLMove
95		12/16/06 2:46 PM	vitaliy	- changed all variables to lowercase[...]
94		12/16/06 2:43 PM	matt	Added nested function calls test, which currently fails.
93		12/16/06 2:11 PM	matt	Added all rule syntax.
92		12/16/06 10:53 AM	matt	Added syntax for rules.
91		12/15/06 9:10 PM	matt	Cleaned up build.xml.[...]
90		12/15/06 8:09 PM	matt	Completed functions that actually return something.[...]
89		12/15/06 5:21 PM	matt	You can now call "procedural" functions; that is, functions that don't return anything.[...]
88		12/15/06 2:58 PM	steven	board's all set.
87		12/15/06 2:14 PM	steven	Commented out exception handling in the constructor of BGGLBoard so we can try to catch it in static semantic analysis.
86		12/15/06 1:22 PM	steven	Fixed BGGLBoard to have an internal ArrayList<BGGLArray> type called boardmatrix. [...]
85		12/15/06 11:45 AM	vitaliy	fixed printing bug in the interpreter[...]
84		12/15/06 4:02 AM	vitaliy	fixed while loop, removed debugging statements[...]
83		12/14/06 12:01 PM	matt	Grammar now correctly builds the AST for function defs, and these nodes are recognized by the walker. Starting on function calls.
82		12/14/06 11:46 AM	matt	Fixed bug where Interpreter was NOT going through semantic analysis but Print tests were.[...]
81		12/13/06 11:53 PM	steven	Steve: Filled out the BGGLBoard class, will comment soon.
80		12/13/06 9:54 PM	matt	Removed all ANTLR-generated Java files from SVN control.
79		12/13/06 9:13 PM	matt	Formatting change only; no code changes.
78		12/13/06 9:07 PM	matt	Added two new targets to build.xml for convenience. Finally.
77		12/13/06 8:34 PM	vitaliy	
76		12/13/06 4:05 AM	vitaliy	added for and foreach statements to parser and walker[...]
75		12/13/06 2:10 AM	vitaliy	- fixed bug in BGGLInt for comparison operators[...]