The Amaze Language

Created by: The Amazing Group

Project Manager:	Rouault Francoeur, rff2111@columbia.edu
System Integrator:	Orlando Pineda, omp2114@columbia.edu
Language Guru:	Daniel Mercado, dm2497@columbia.edu
System Architect:	Jonathan Bourdett,jab2279@columbia.edu
Tester and Validator:	Jose Contreras, jdc2168@columbia.edu

Contents

Chapter 1: The Amaze *Language*: An Overview

Introduction

The Amaze language is designed to easily create mazes outputted on a graphical user interface. This is in contrast to using common programming languages, (such as Java or C) to create a graphical user interface, which requires a lot of overhead and research into various libraries. We wanted to create a language that would make it easy to create a simple maze, but also intuitive enough to build more complicated mazes.

Simple

Building a simple maze in Java is not as simple as one might think. GUI's in Java have many options that one needs to keep track of in order to display shapes or images. This is because Java is meant to be robust, so in order to do something very specific such as displaying graphics; one has to learn how to use the appropriate library. Other languages face similar dilemmas.

We want our language to be optimized for designing mazes. The main commands of the language will consist of actions directly associated with manipulating paths. Our language will allow the user to easily create a maze without the need to initialize the appropriate objects for Java graphical user interfaces. The user will be able to simply specify the type and location of a maze path using a coordinate system.

Joe wants to program a maze game for his friend, Ada, to play, but is spending too much time trying to figure out JFrame methods and what they do instead of actually working on the design of the game.

Detailed

Out language allows programmers to be detailed and specific when designing mazes. Our goal is to give them the most power in designing their maze using the Amaze language. Although the language is simple and intuitive, we don't want to restrict the programmers from being as detailed as possible in the design of their mazes. If users want to recreate some of their favorite mazes line by line we should allow programmers to do so.

The commands we provide will be simple and intuitive allowing for simpler mazes to be easily created and for bigger more complicated mazes to be broken down into small well-defined commands. Our language implements all the structural components that any maze will need in order to be created by the programmer.

Joe wants to program a really detailed maze. He wants his maze to have x number of walls, y number of dead ends. Programming these various details in Java, for example, requires manipulating parameters of Java graphics functions in many places. Because Amaze is specific to creating mazes, Amaze allows Joe to make these all of these detailed design in a simple way. Thus, it will be easier to define these constructs in Amaze than in another more general programming language, such as Java.

Architecture Neutral/Interpreted/Portable
Amaze will allow the programmer to compile and run their maze on machines that may not share the same CPU or operating system with the machine in which the maze was developed. This would be accomplished in a process similar to Java in which the Amaze compiler will produce a Java executable which would not be specific to a particular CPU or operating system. Adding this functionality to our language would allow programmers to have interactions with other systems or programmers, without having to use a specific CPU or operating system.

Some of these interactions would include the following:

Programmers are allowed to share their compiled programs so that others may be able to view the mazes on their own systems, without having to share their source code. This would give programmers the ability to distribute and share their maze with others without giving the user of the program access to alter the maze, effectively adding a level of security.

Programmers are allowed to collaborate and develop different parts of their maze separately without having to be restricted to a specific system type.

Joe wants to work with his friend Ada on developing an intricate maze. However, Ada uses a computer running on MacOS and Joe uses a computer running on Windows and is worried about being able to split the work between the two, due to the differences in OS. However, due to the portability of our Amaze Joe and Ada are able to effectively design a program that generates a maze.

Intuitive
Many people know how to solve mazes, but not necessarily design them. Amaze has commands and constructs that the average programmer already has an intuition for, so that the learning curve of our language is not that steep.

Most people can already think of the basic parameters of actually designing a maze; these parameters can include the positioning of paths, the start/end point of a player, and perhaps the size of the maze as a whole, just to name a few. Amaze provides functions with obvious names that match up with these types of parameter changes.

Joe wants to design a simple maze to show off to his friends what he is learning in school. He figures that writing a program that displays some cool graphics will probably do the trick, and so he tells them he will design a simple maze for them. Joe knew Java generally well, so he thought he would write this game in Java, even though he hadn't really used the graphics libraries.

But he realizes that it is not intuitive to call upon the swing libraries and create containers and panels and such. It really will take him a long time to figure out how to design this maze in Java or any other similar language, for that matter. By the time Joe figured out how to design his maze in Java, he took too long, and his friends assume that Joe's programming skills need some practice, or that programming mazes into GUIs is too hard.

Educational

Our language is meant to be simple, keeping the learning curve low so that anyone could use and develop simple or intricate mazes. Because of this, our language could be used to introduce and attract people into the field of computer science by allowing them to learn the basics of programming and have fun while doing so.

Jane is a high school student who has developed an interest in computer science. However she is intimidated by most programming languages and is considering taking up something else instead. Due to the entertaining aspects of our language and the fact she enjoys mazes, Jane decides to learn the Amaze language. While learning Amaze, Jane receives a good introduction to programming, allowing her to easily transition into other programming languages like C or Java.

Creative

When writing code in our language we want users to be able to take creative steps with their programs. Amaze tries not to be overly restrictive about the types of mazes one can design.

Our language allows for creative maze design. Perhaps a user would like to design a maze with their name in it or a maze modeled after an image . Our language encourages creative steps to maze development.

Joe wants to recreate the maze layout from his favorite video games. Amaze will allow him to do so and also exercise his creativeness to design his own version.

Summary

The Amaze Language provides an easy way to design both simple and complicated mazes because it is a programming language allowing the user to take advantage of control flow, functions, and types. The language is ideal for any person with or without programming experience interested in a simple yet deep method of designing mazes.

Sample Program in Amaze

```
struct a {
      point b: 25, 25;
      path c: b, right, 45;
}
board d {
      size: 100, 100;
      start: 0, 0;
      end: 50, 50;

      point e: 0, 0;
      point f: 50, 0;

      path g: e, right, 50;
      path h: f, down, 50;
      set(a);
}
board i {
      size: 100, 100;
      start: 50, 0;
      end: 50, 50;

      point j: 0, 0;

      path k: j, down, 50;
      set(a);
}
main {
      draw(i);
      draw(d);
}
```

Chapter 2: Amaze Language Tutorial

Getting Started

Amaze is a language used for creating custom maze designs to be displayed in a GUI. The simplest possible Amaze program requires you to define a board with size coordinates, a start point, and an end point. A main block would then draw that board.

Board

```
board board1{
      size : 11, 11;
      start : 0,0;
      end : 10, 10;
}
```

Here, we have created an 11 by 11 board with the start and end points at the corners. It is important to note that the Amaze language uses a tile based coordinate system for laying out paths for a maze. A 4x4 board, for example, would have its coordinates laid out in the following manner:

0,0	**1,0**	**2,0**	**3,0**
0,1	**1,1**	**2,1**	**3,1**
0,2	**1,2**	**2,2**	**3,2**
0,3	**1,3**	**2,3**	**3,3**

*Note: Maximum size of board is defined by the designer.

The first number of the coordinate point represents horizontal position. The number increases rightward. The second number of the coordinate point represents vertical position. The number increases downward.

We use the convention of ":" to define size, start, end, point, and path. Semicolons end the statements. "board1" is the id of the board that will be needed later to actually draw the board. The whole board is encapsulated within brackets.

Paths

We now need to place paths in this board. To initialize a path, we need to also initialize a starting point for that path. You must also specify a direction for that path to be drawn, as well as the number of tiles that extend from that starting point to form the path.

```
Point p : 0,0;
Path myPath : p, down, 10;
```

This snippet of code, if located within a board structure, would know to draw a path starting at (0,0), and extending to the bottom right point (0,10). Current program:

```
board board1{
      size : 11, 11;
      start : 0,0;
      end : 10, 10;

      Point p : 0,0;
      Path myPath : p, down, 10;
}
```

Main

To complete this basic program, we need only to include a main block that will actually draw the maze:

```
main{
      draw(board1);
}
```

Sample program

Including one more path along with the above code, we get the following Amaze program:

```
board board1{
      size : 11, 11;
      start : 0,0;
      end : 10, 10;

      Point p : 0,0;
      Path myPath : p, down, 10;
}

main{
      draw(board1);
}
```

Compiling and Running the Program

To run and compile an Amaze program, we would need to have our code in a file appended with ".amz". Then, from the directory containing the Amaze file, you would type the following to compile:

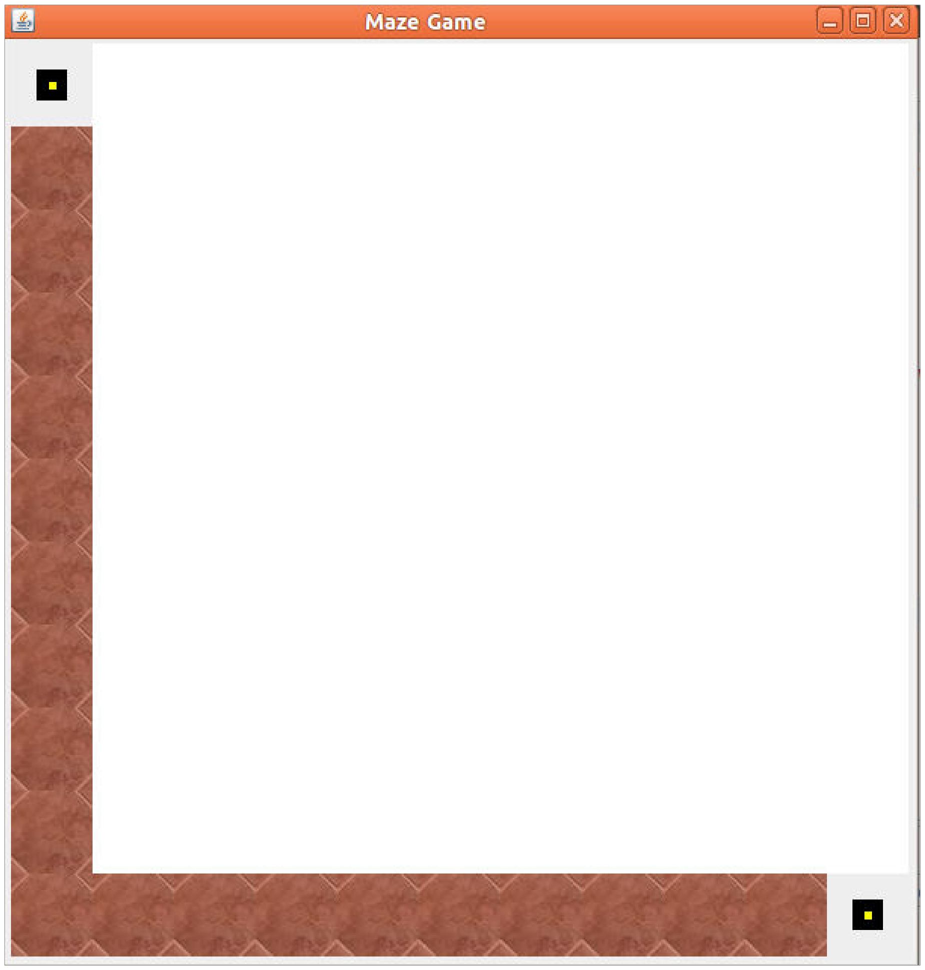
```
./amazec.sh Test.amz
```

The would produce a .class java file in the src directory of our compiler project folder, which you could run with the Java Virtual Machine(JVM):

```
java Out
```

Output

Running the target executable our Amaze compiler, you would see a GUI displaying the maze:



General Structure of Amaze File

Above we defined our simple Test.amz file with just one board declaration and a main declaration to draw that board. However, an amaze file generally has this structure:

Structures/Functions/Global Variables
Boards
Main

We could have declared various structures, functions, and/or global variables above the board declaration. Defining these globally can be useful if a specific path or path pattern is reused in multiple boards. We'll look at each individually, but we would like to identify what types we have in our Amaze language first.

Types

Primitives

The Amaze language only has a couple of primitive data types. They are:

int	a type representing an integer value.
boolean	a type representing a true or false value.

General Constructs

Amaze also has more general types:

point	a type representing a point on the board.
path	a type representing a collection of points intended for a player to traverse.
	board a type representing a collection of points on a grid layout. will be displayed in a gui
structure	a type representing a collection of paths that can be inserted into board
function_type	a type that can represent a point, and int, or a boolean

Variables

Variable Declaration

To declare a variable, you write the keyword representing the type, then the variable name:

```
int aNumber;
```

The keywords are reserved, and not allowed as variable names. The first character of the variable name must be a letter. The remaining characters can be any combination of letters and numbers.

Valid:

```
      int aNumber;
      boolean name34;
```

Invalid:

```
      int 32string;
      boolean 34name;
```

Variable Initialization

To initialize a variable, you would use "=" followed by the constant or expression. For example:

```
int number = 5;
int number2 = 5 + 5 ;
boolean foobar = false;
```

Again, we initialize point, path, and board in the following fashion:

```
point k= 3,3;
path bob = k,up,9;


board maze2 {
      size = 5, 5;
      start = 1, 1;
      end = 2, 2;
}
```

Path needs a point reference as its first parameter. Board need not actually have any paths declared inside, but it does need to declare size, start, and end.

Reserved Keywords

When initializing variables, you cannot use any names already reserved for tokens. These include:

```
MAIN : 'main';
DRAW: 'draw';
STRUCTURE : 'structure';
FUNC : 'func';
INT : 'int';
WHILE: 'while';
RETURN : 'return';
BOOLEAN: 'boolean';
IF      : 'if';
ELSE   :' else';
SET    : 'set';
SIZE   : 'size';
END    : 'end';
START : 'start';
FALSE : 'false';
TRUE  : 'true';
BOARD : 'board';
LEFT   : 'left';
UPP    : 'up';
DWN : 'down';
RIGHT : 'right';
PATH  : 'path';
POINT : 'point';
PRINT : 'print';
```

Global Variables

Declaring a variable before and outside of the board blocks allows that variable to be accessible to all of the boards.

Example program:

```
int x = 2;
int y = 2;

func int addTwo (int f , int s)
{
      return f + s;
}

board board1{
      size : 10, 10;
      start: 2 , 3;
      end: 3, 3;

      int z = 2;
      int a = 2;
      int m = addTwo (x,y);
      int n = addTwo (a,z);

}

main{
      draw(board1);
}
```

As shown in the code above x and y are global variables that can be accessed anywhere within our program while a and z can only be accessed within the main block.

Structure

For Amaze, structures are useful in holding various paths. You can then call upon that structure by its name within a board to draw those many paths. For example, if a programmer wants to create a sort of cross structure or T shape they can define it as follows:

Declaring and Initializing Structures

```
structure cross {
      point px : 10,10;
      path upx : px, up, 20;
      path downx : px, down, 20;
      path leftx : px, left, 20;
      path rightx : px, right 20;
}
```

Set Structure in Board

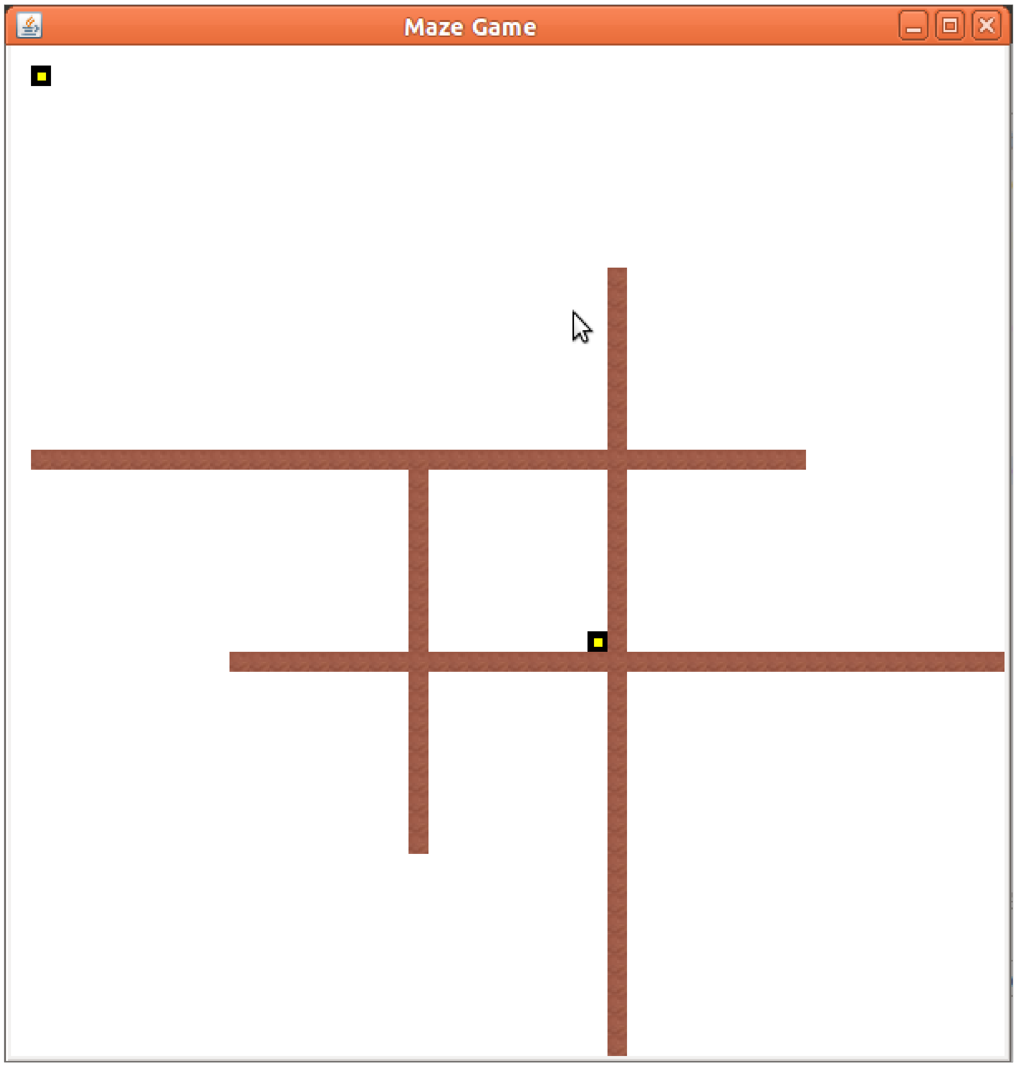
You could call upon these structures with set(nameOfStructure):

Example Program:

```
structure cross {
      point px : 30,30;
      path upx : px, up, 19;
      path downx : px, down, 19;
      path leftx : px, left, 19;
      path rightx : px, right, 19;
}

structure downtshape {
      point p : 20, 20;
      path downy : p, down, 19;
      path lefty : p, left, 19;
      path righty : p, right, 19;
}

board board1{
      size: 50, 50;
      start: 1, 1;
      end: 29, 29;

      set (cross);
      set(downtshape);
}
```

```
board board2{
      size: 50, 50;
      start: 1, 1;
      end: 29, 29;

      set(cross);
      set(downtshape);
}

main{
      draw(board1);
}
```

Output:



Function

When using and creating functions those familiar with Java should not have any issues with the format of our functions since it is so similar to that of Java methods.

Form for functions

```
func return-type function-identifier(set-of-parameters)
{
      body of function and return statement
}
```

As we can see in the form of the function there must be a valid return statement for every function. By valid return statement we mean that the function should return the type that is specified in the header of the function declaration.

Calling on Functions

Example:

```
func int factorial(int x)
{
      if(x==0)
      {
            return 1;
      }
      else
      {
            return x*factorial(x-1);
      }
}
```

In the example code above you can see how we return values, and their types are the same as the one identified in the header. Trying to return any value other than one of the type in the header will result in an error. Any variable that is created within the function and the parameters are all local variables within the function that can only be accessed within the function.

Functions provide the programmer with the tools to create complex functions similar to that of methods in Java. Functions all must be declared in their own blocks and cannot be created within boards, structures, and the main block.

Expressions

Amaze has support for expressions among ints and booleans. Other than size, start, and end, any other general variables can use expressions in place of the usual constant value. Expressions require operators and a definition of precedence.

Arithmetic Operators

There are unary and binary arithmetic operators.

Unary Operators

The unary operators include - and !. These unary operators have the same precedence, and the highest precedence. They act immediately on the term to its right. For example:

```
- 0
-(0)
-(9+5)
- 9 + 5
```

The unary operator '-' negates a number/boolean, while '!' flips booleans and makes integers from zero to non-zero(and vice-versa).

Binary Operators

In the above expressions, as in usual order of operations 0 is negated in the first two expressions, (9+5) is negated in the third expression, and only 9 is negated in the last expression.

The binary operators include +, -, *, and /. These have a lower precedence than the unary operators. They are left-to-right associative.

Relational, Equality, and Logical Operators

Relational Operators

The relational operators are

```
>     >=    <     <=
```

These relational operators are binary, of course, and are left-to-right associative.

Equality Operators

Lower in precedence to these relational operators are the equality operators:

==   !=

As an example in this precedence:

a <= c != b>=d is equivalent to (a<=c) != (b>=d). This is rather than a <= (c != b) >= d.

Logical Operators

Finally, we have the logical operators && and ||. These have the lowest precedence of the operators mentioned so far. || has a lower precedence than &&.

Example:

a <= c == b >= d || e < f && r>=s != c<d

With parentheses for clarity on the order, the expression looks like this:

((a<=c) == (b>=d)) || ((e<f) && ((r>=s) != (c<d)))

Conditional Expressions

Conditional Expressions refer to either if and else, or a while loop.

if/else

And if/else statement will check the expression within the if parentheses for truth. If true, it will execute the code immediately following if. If false, it will execute the code immediately following else.

Example:

```
int temp = 0;
int n=8;
if (n/2 <6){
   temp = 1;
}
else{
   temp = -1;
}
```

while

A while loop will continue to run until the statement within the while is no longer true.

Example:

```
int i=0;
while (i<6){
      print(“Hello”);
      i++;
}
```

This code will print Hello 6 times.

Precedence and Order and Evaluation

The following table summarizes the precedence and order of operations for all of our operators. Each line in the table has symbols that have the same precedence.

Operators	**Associativity**
()	**left to right**
- !	**right to left**
*** -**	**left to right**
+ -	**left to right**
> >= < <=	**left to right**
== !=	**left to right**
&&	**left to right**
||	**left to right**
=	**right to left**
,	**left to right**

Other Notes

Print

You can print out messages to appear in console. For example,

```
board1{
    …

    print (“Hello”);

}
```

Scope

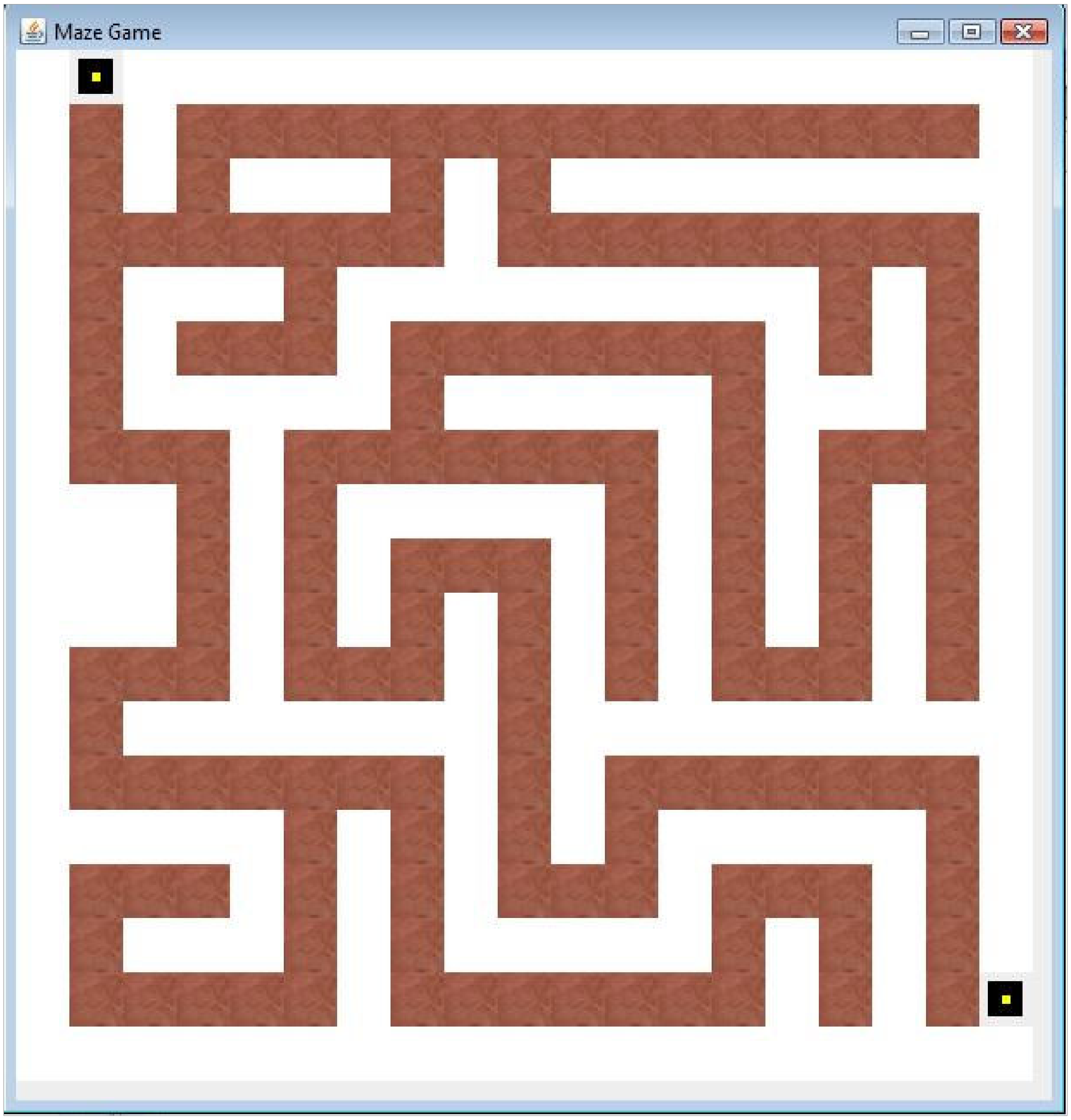
In the Amaze language the scope of variables is very similar to those in Java in the sense that every variable is limited to the block that it is in. Any variables that are initialized within a structure can only be accessed within the structure in which they are initialized. Any variables that are declared in the board struct or in the main block act the same as in structures since any variables are declared within those sections can only be accessed within those sections.

Comments

You can comment a piece of code from being processed by the compiler by placing /* */ around the code:

```
main{
        /*comments go here*/
}
```

Sample Programs



```
board board1
{
        size: 19, 19;
        start: 1, 0;
        end: 18, 17;

        point a: 1, 0;
        path a1: a, down, 7;
        point b: 3,7;
        path b1:b, left, 2;
        path b2t3:b, down, 4;
        point c: 1, 11;
        path c1: c, right, 2;
        path c2: c, down, 2;
        point d: 3,3;
        path d1: d, up, 2;
        path d2: d, left,2;
        point e: 1,9;
        path e1: d, right,1;
        point f: 5,13;
        path f1: f,left,4;
        path f2: f,down,4;
        path f3: f,right,2;
        point g: 1, 17;
        path g1: g,up,2;
        path g2: g,right,3;
        point h: 3,15;
        path h1: h, left,1;
        point i: 7,17;
        path i1: i, up, 3;
        path i2: i, right,6;
        point j: 13, 15;
        path j1: j, down, 1;
        path j2: j, right, 2;
        point k: 15,17;
        path k1: k, up, 1;
        point l: 17,17;
        path l1: l, right 1;
        path l2: l, up, 4;
        point m: 11,13;
        path m1: m, right , 5;
        path m2: m, down, 2;
        point n: 9,15;
        path n1: n, right, 2;
        path n2: n, up, 6;
        point o: 7,9;
        path o1: o , right,2;
        path o2: o, down,2;
        point p: 5,11;
        path p1:  p, right, 2;
        path p2: p, up, 4;
        point q: 7,7;
        path q1: q, left, 2;
        path q2: q, right, 4;
        path q3: q, up, 2;
        point r: 11,11;
        path r1: r, up, 4;
        point s: 13, 5;
```

```
        path s1: s, left, 5;
        path s2: s, down, 6;
        point t: 15, 11;
        path t1: t , left, 1;
        path t2: t , up, 4;
        point u: 17,7;
        path u1: u, left, 1;
        path u2: u, down, 4;
        path u3: u, up, 4;
        point v: 15,3;
        path v1: v, right, 1;
        path v2: v, left, 6;
        path v3: v, down, 2;
        point w: 9, 1;
        path w1: w, down, 1;
        path w2: w, right, 8;
        path w3: w, left, 5;
        point x: 7,3;
        path x1: x, up,2;
        path x2: x, left,2;
        point y: 5,5;
        path y1: y, up,2;
        path y2: y, left,2;
}
main
{
        draw (board1);
}
------------------------------------------------------------------------------------------------------------------------
board graphic
{
        size: 250, 250;
        start: 175, 175;
        end: 10, 10;
        int y = 0;
        int x = 0;
        while(x<10)
        {
                y = x + f(x+y);
                point p : y, x;
                path pat : p, right, 3;
                x = x + 1;
        }
}
main
{
        draw(graphic);
}
------------------------------------------------------------------------------------------------------------------------
```

Chapter3: Reference Manual

1. Introduction/Language Description

Amaze is a programming language designed for the purpose of creating mazes for an interactive maze game. Amaze is not architecture specific. The language compiles into Java code. Because of the simplicity of Amaze, the user does not have to be familiar with Java's interface classes and libraries such as JComponent, JPanel, or JFrame. The languages still allows for the use of more advanced programming features such as loops, conditional statements, and functions in order to allow maze designers an intuitive method of constructing mazes.

2. Lexical Conventions

2.1 Comments

Similar to C and Java, Amaze allows for the user to type comments throughout the code using the characters /* to introduce a comment and the characters */ to end a comment block.
Example:

```
/* This is a comment
in Amaze. */
```

2.2 Identifiers

Identifiers in Amaze, are similar to other well known programming languages like C. An identifier is made up of a sequence of alphanumeric characters. The first character in an identifier must be a letter.
Example:

```
int x
```

x is an identifier.

```
board stage1
```

stage1 is an identifier.

2.3 Keywords/Data Types

Keywords cannot be used as identifiers. Some keywords are used as data types that describe the property of an identifier. Other keywords represent statements that provide a specific type of functionality. These keywords are:

```
board
int
point
path
structure
main
size
start
end
draw
set
print
while
if
left
right
up
down
func
boolean
true
false
return
```

2.4 Integer Constants

An integer constant is a sequence of digits.
Example:

```
int x = 978;
```

978 is an integer.

2.5 Print Statements

Printing Statements can print any sequence of characters. The text constant within the print statement is surrounded by double quotes. “Amaze” for example is considered a text constant that will be used in the print statement print() similar to C.
Example:

```
print(“Print this statement”);
```

“Print this statement” is a text constant

print statements can print any value, not just alphabetical characters.

For example:

```
print(“Print this: 1 + 2. @ #3”);
```

This will print “Print this: 1 + 2. @ #3”;

2.6 Booleans

Booleans are a type which have values either true or false.

Example:

```
boolean b = true;
if (b){
 print (“This prints if true.”);
}
```

This code snippet will print out only if b is true. If b was false, nothing would print, because the if statement would not have been satisfied, so the code inside would not execute.

3. Meaning of Identifiers

Identifiers, like names, allow the programmer to reference different types and structures by a name specified during its declaration. Some of these include structures, types, and functions. In amaze, structures are predefined and consist of predefined types. These types are given preset identifiers which must be used when interacting with these types. Like structures, all types in amaze are predefined. These predefined types are used to specify what kind of information is held in that type, and if compatible, these types can be converted to other types after declaration. This is discussed later in the conversion section. These types and structures all have the same scopes which is similar to java, in which the declared type/struct's longevity is determined by the block in which it is declared. More about scope will be discussed in the scope section.

3.1 Basic Types

As stated before, all types in Amaze are predefined. The range of these variables is dependent on the language in which we are compiling to(Java). These types are as follows:

Objects declared as int have the same range as those in Java, meaning they have a range of 4 bytes signed. In amaze integer values are the main form of numerical representation as well as the main form arithmetic operations. We decided to this in order to keep the language simple and avoid ambiguity when constructing maze points and combining them with arithmetic statements. The truncation is the same as in Java, which means that numbers are rounded down.

Example:

```
int temp = 0;
```

3.2 Derived Types

Aside from the basic types, there are several derived types that are constructed from basic types. There are two types of derived types in amaze, structures and functions:

There are several types of structures in amaze. The first is of type size. The size structure is composed of two basic integer types which are named length and width.

Example:

size: 0,0;

Another structure type in amaze is the type point. Points whole two int types which are used to reference specific x,y coordinates on the board.

Example:

point temp: 0,0;

Another structure is the type path. This type is used to specify the traversable points on a specific map. It is made up of a point to specify the start of a path, a direction which indicates the direction the path will run from the point and another integer value to specify the length of the path.

Example:

point temp : 0,0;
path mainpath : temp, left, 100;

Another derived type is the type board. The board type is similar to that of a function except it has a specific purpose of drawing a maze map and specifying the paths and points. It consists of a size type, two point types to specify the starting position and the goal position and combinations of iteration and conditional statements used to create the paths and points on the map.

Example:

```
board stage1 {
           size: 500,500;
           start: 0,0;
           end: 76,6;
set (piece);
           int i = 0;
           while(i < 80) {
                if (i != 56) {
                     point j: 1,i;
      path l : j,left,7;
      }
           i = i + 1;
}
print("This is a message");
/* This is a comment. */
}
```

Finally the last derived type is of the type structure whose body contains a combination of conditionals and iteration statements which may define several basic types and may return one of those types.

Example:

```
structure piece {
           point p : 1,1;
int x : 99;
path e : p,left,x;
}
```

4. Expressions and Operators

Expressions are composed of operators acting on the following tokens, which we will also call primary expressions:

- constants
- booleans
- identifiers (aka variables)
- parenthesized expressions
- primary-expression (expression-listopt)

4.1 Primary Expressions

Constants

Constants refer to whole decimal numbers. In our language, constants are of type int.

Booleans

Booleans refer to the type boolean, which is either true or false.

Identifiers

An identifier is a primary expression, provided its type has been declared properly.

(expression)

A parenthesized expression is a primary expression whose type is of the return type of the expression within the parentheses.

primary-expression (expression-listopt)
Functions have a return value, which is the return value of the primary-expression.
This function can also potentially have a list of parameters, which are expressions themselves. They are separated by commas. There need not be any parameters though.

Amaze does not allow you to change the actual parameters used for a method. Rather, copies of the parameters are created for you to edit. Thus, Amaze passes parameters by value.
Parentheses will always proceed the name(primary-expression) of a function, whether there are parameters or not. This differentiates them from usual primary-expressions.

Recursive calls to any function are permissible.

4.2 Unary operators

-expression

The result is the negative of the expression. The type of the expression must be of type int.

!expression

The result is the logical negation of the expression. The type of the expression must be of type int or boolean.

The logical negation of a boolean is obvious; !true = false and !false = true. For ints, !0 = 1 and !a = 0, where a is any int not equal to 0.

4.3 Multiplicative Operators

expression * expression

The binary * operator indicates the usual arithmetic multiplication operation. The type of each expression must be int, and it returns a value of type int.

expression / expression
The binary / operator indicators arithmetic division. The type of each expression must be int, and it returns a value of type int.

4.4 Additive Operators

expression + expression

The + operator indicates arithmetic addition. The type of each expression must be int, and it returns a value of type int.

expression - expression

The - operator indicates arithmetic subtraction. The type of each expression must be int, and it returns a value of type int.

4.5 Relational and Equality Operators

expression < expression
expression > expression
expression <= expression
expression >= expression

These operators refer to the logical operations between numbers, less than, greater than, less than or equal to, and greater than or equal to. The result of these expressions will be either true or false, depending on whether the expression satisfies that said inequality.

expression == expression
expression != expression

These binary operators refer to the logical operations checking for equality and non-equality. They return a boolean of true or false, depending on whether the check was satisfied or not. The type of each expression must be int or boolean, and both sides of the equality must have the same type.

4.6 Logical AND Operator

expression && expression

This operator represents the logical and operator. If both expressions do not equal 0(and not null), then the result will be true. Otherwise, the result will be false. The expressions must both return a type of boolean.

4.7 Logical OR Operator

expression || expression

This operator represents the logical or operator. If either expression is not equal to 0,(and not null) then the result will be 1. Both expressions must have a type boolean.

All of these expressions are group left-to-right. Precedence is defined as in the order which we have presented it, which is in agreement with the usual order of operations in mathematics.

4.8 Assignment Operators

In amaze there are two different assignment operators that are used depending on the type of object being assigned a value. If the object is one of the basic types then an '=' sign followed by the value(or expression) being assigned to that type.

Type-specifier identifier **=** *expression* **;**

Where type-specifier can be int or boolean.

In addition, assignments can be declared to already defined identifiers using the following form.

Identifier = expression;

However, the type of the identifier and the type of the expression must match

If the object is a struct, then the ":" sign is used to pass all the values that are to be placed in the types that are held within the struct.

Struct-specifier identifier : value1, value2 ;

The possible structs are point and path.

5. Declarations

Declarations specify the type associated with a specific identifier. They consist of a type followed by an identifier. The format is seen in the following grammar:

declaration → structure-declaration | board-declaration | var-declaration | function-declaration

structure-declaration → structure ID{statement-list}

function-declaration → func ID type-specifier (parameters) {statement-list}

var-declaration → type-specifier ID;

parameters → parameters, var-declaration | var-declaration

5.1 Type Specifiers and Type Declaration

The type specifiers of Amaze are used to declare objects that belong to the type specified. The data that is being inserted into the object must match that of the type-specifier or an error will be returned. The grammar is as follows:

Type-specifier identifier = expression ;

Type-specifier:
- int
- boolean

Similar to the type-specifiers are the derived type-specifier which indicate the type of derived object being declared. The main difference is that these declarations take in the values that a struct will have and instead of an equal sign, a semicolon is used. The grammar would look something like:

Struct-specifier identifier : value1,value2,...;

Struct-specifier:
- point
- path

5.2 Structure and Board Declaration

Board and structure declarations are very similar. Both consist of the derived type-specifier followed by and identifier and finally the body, which consists of the iteration and condition statements used to construct the board and execute the structure:

structure-specifier identifier { *body;* }
board-specifier identifier { *body;* }

5.3 Initialization

The initialization for the basic types are similar to that of java. If they are declared without an assignment expression then the initial value is set to 0 or null. If declared using an assignment operator then the initial value of the object will be the value assigned to it via the assignment operator. For example:

Type-specifier x;

x is zero or null.

Type-specifier x = expression;

x is assigned the value of the expression.

When initializing derived struct types, the values contained in the struct will be passed to it using the colon assignment operator. If the programmer does not specify a value for the types held in the struct, then these values will be set to null or 0. For example:

Struct-specifier p: value1, value2;

Creates a point struct with the values 1 and 2.

6. Statements

Statements are like sentences in natural human language. They represent the execution of a complete idea. In our language as in C and Java, statements are executed in order found or sequentially, except where described below. A statement is executed to perform a desired effect, and do not hold an inherent derived value. Below are the categories for the statements in our language.

statement: assgn_expression
iteration_stmt
selection_stmt
point_declaration
path_declaration
var_declaration

6.1 Expression Statements

The majority of the statements in the language will be expression statements. The form they will take is as follows.

expression-statement:
expression$_{opt}$;

Examples:

```
int i = 0;
i = i + 1;
point p : 1,1;
print("Hello World");
```

All expression statements end in a semicolon. The effects of the expression are done before the execution of the next statement. Expression statements for the most part will consist only of assignments or function calls.

6.2 Compound Statements

Compound statements are a collection of several statements group together to act as one statement. They are traditionally known as "blocks" of code. The body of structures,board, conditionals and functions are all examples of a compound statement. The multiple statements in our language, as in C and Java, are surrounded by brackets.

declaration-list:
declaration
declaration-list

statement-list:
statement
statement-list statement

Examples:

Selection-statement example:

```
if (foo == 1) {
foo = 10;
foo = foo * 5;
}
```

Iteration-statement example:

```
while (foo > 0)
{
      int bar = foo;
      bar = bar - 1;
}
```

Function-declaration example:

```
func string say (string a) {
if (a != "") {
a = a + "!";
      a = "You said: " + a;
            return a;
      }
      a = "You didn't say anything.";
      return a;
}
```

If a variable is declared outside and within a block, the variable in the block takes precedence. Variables within a block are only allowed to be declared once. Within function declarations if return is executed, control is given back to the calling statement. The rest of the statements in the block are ignored.

6.3 Selection Statements

Selection statement are statements that are executed or ignored based on the value of the evaluation of an expression. The two selection statements included in our language are as follows.

selection-statements:
if *(expression)* statement
if (expression) statement else_statement
else_statement:
else
empty

Examples:

```
int i = 0;
if ( i == 0)
      print ("Hello World");

if ( i == 1)
      print("One.");
else
      print ("Zero");
```

If the expression in the if statement evaluates to anything other than 0 or false then the following statement is executed. If the expression in the if-else statement evaluates to 0 or false then control of the program skips the if statement and executes the statement following the else keyword.

6.4 Iteration Statements

Looping in our language is achieved through iteration statements. In our Amaze language, we use only one iteration statement.

iteration-statement:
while *(expression)* statement

A the start of each statement, an evaluation of the expression in the while statement controls the flow the program. While the expression evaluates to anything other than 0 or false, then the following statement is repeated. At the end of the execution of the statement control is returned to the test case. The repetition only terminates when the expression evaluates to 0 or false.

Example:

```
int i = 3;
while (i > 3) {
      i = i - 1;
}
```

6.5 Jump Statements

Jump statements immediately transfer control of the program. The Amaze Language only uses the jump statement return in function definitions.

jump-statement:
return *expression$_{opt}$*;

Example:

```
func string sayHello () {
      string a = "Hello World!";
      return a;
}
```

The return statement is used to return control from the function to the statement calling the function. If return is followed by an expression, its evaluation is returned to the calling statement.

7. Variable Declaration

Declaring a variable in Amaze is as simple and intuitive as in Java for most variables. The basic form for variable declaration:
Type-specifier *identifier*;
All of our variables are simply declared by including the type of object of being created and the identifier which can be anything that follows suit with the restrictions that we set for identifiers.

```
int num;
```

The declarations above are valid forms of variable declarations.

```
int string;
```

The two declarations above on the other hand are not valid forms of variable declaration since both string and int are keywords that we predetermined that they were not available to the user.
There are a few key words that hold certain variable information that pertains to the creation of mazes but they do not need to be initialized because we defined them as always holding the same type of values. This can be seen in the structure that we have to the board:

```
board stage1
{
size: 50,50;
start: 0,0;
end: 25,50;
}
```

.

When defining size, start, and end which we define as structures and when these are initialized the data is saved under the keyword's allocated space in memory and we retrieve that information using these predefined keywords so these variables are initialized essentially before the program is created.

8. Functions

Functions are essential with any language because they allow the user to create their own algorithms which might return any information that the user wants from their algorithms. In the Amaze language our functions work similarly to java methods and have similar structure to a java method.
Form for function declarations:

```
func return-type function-identifier(set-of-parameters)
{
      body of function and return statement
}
```

The key word func must be used whenever declaring a function. Every function has return type which follows func in order to allow the user to expect a certain type of value to be returned by the function. As in Java following the return type we have the identifier which will be the name of the function. Followed in parentheses there will be a set of parameters which will be defined by the user to determine which parameters will be need in the function and their appropriate types.

Following the header described above the body of the function will be written within the area between these two outer braces. Within the body of the function you can declare any variable but scope rules will apply to these variables. Any type of function can be created within the body, it is up to the user to determine the complexity of algorithm(s) that will be created within the body it can be a simple counter that returns a boolean saying that the counter has counted.

Example:

```
int count=0;
func boolean counter()
{
      count = count + 1;
      return true;
}
```

The user might also want to create a slightly more complex function that uses recursion
Example:

```
func int gcd(int m,int n)
{
if(n==0)
{
return m;
}
if(n>m)
{
return gcd(n,m);
}
else
{
            int x = m/ n;
            int y = m*x;
return gcd(n,m-y);
}
}
```

Functions add a complexity to our language that allows us to provide the user with tools that one might use in creating a maze. Functions allow us to reuse algorithms because they take parameters and work as a result of these parameters.

8.1 Built-in Functions

The built in functions that are in our amaze program are `draw`, `set`, and `print`.

`draw`

The `draw` function takes a board as a parameter and creates the maze according to the specifications that have been set within the board it took as a parameter.

`set`

The set function takes a path as its parameter and sets it within the board. The print function will display text below the graphic whenever the is called.

`print`

The print function is called by using the keyword `print` and followed by set of parentheses containing either an expression returning an `int` or a `string`.

9. Language Grammar

ANTLR Grammar:

```
grammar Amaze;

options {

language = Java;
output = AST;
ASTLabelType = CommonTree;
}
tokens{
        NEGATIONS ;
        REASSIGN;

}
@header{
package compiler;
}

@lexer::header{
package compiler;
}

maze : declaration_list main_declaration EOF!
;

declaration_list : declaration declaration_list
|/*empty*/
;

declaration : structure_declaration
| board_declaration
| var_declaration
| function_declaration
;

board_declaration: BOARD ID '{'! board_declaration_list '}'!
;


board_declaration_list: size_declaration start_declaration end_declaration board_statement_list2
/*board_statement_list*/
;
```

```
size_declaration : SIZE ':'! expression ','! expression';'!
;
start_declaration : START ':'! expression','! expression';'!
;

end_declaration : END ':'! expression','! expression';'!
;

var_declaration : int_declaration
| bool_declaration
;

int_declaration : INT ID ';'!
                  | INT ID ASSIGN^ expression ';'!
                  ;

bool_declaration : BOOLEAN ID ';'!
                   | BOOLEAN ID ASSIGN^ expression ';'!
                   ;
set_structure : SET '('! ID ')'! ';'!
;


point_declaration : POINT ID ':'! expression ','! expression ';'!
| POINT ID ';'!
;

path_declaration : PATH ID ':'! (ID | function_call) ','! direction','! expression ';'!
| PATH ID';'!
;

board_statement :
board_iteration_stmt
| function_call
| board_selection_stmt
| print_stmt
| assgn_expression
;
board_iteration_stmt : WHILE^ '('! expression ')'! '{'! board_statement_list2 '}'!
;

board_selection_stmt : IF^ '('! expression ')'! '{'! board_statement_list2 '}'! board_selection_else_stmt
;
board_selection_else_stmt : ELSE^ '{'! board_statement_list2 '}'!
|/*empty*/
;
function_call     : ID'(' expression_list ')'
                                                      ;
direction : LEFT
| RIGHT
| UPP
| DWN
;

main_declaration : MAIN '{'! main_declaration_list '}'!
                    ;

main_declaration_list : DRAW '('! ID ')'! ';'! main_declaration_list
                    |/*empty*/
                    ;


structure_declaration : STRUCTURE ID '{'! structure_body '}'!
;

structure_body :
structure_statement structure_body
| /*empty*/
;

structure_statement : assgn_expression
| point_declaration
| structure_path_declaration
| var_declaration
| structure_board_statement
;
structure_board_statement :
structure_iteration_stmt
| structure_selection_stmt
|print_stmt
| function_call
;
structure_path_declaration : PATH ID ':'! (ID | function_call) ','! direction','! expression ';'!
| PATH ID';'!
;
structure_iteration_stmt : WHILE^ '('! expression ')'! '{'! structure_body '}'!
;

structure_selection_stmt : IF^ '('! expression ')'! '{'! structure_body '}'! structure_selection_else_stmt
;
structure_selection_else_stmt : ELSE^ '{'! structure_body '}'!
|/*empty*/
;

function_declaration : FUNC INT ID'('!parameters')'! '{'!board_statement_list2 jump_stmt'}'!
                    | FUNC BOOLEAN ID'('!parameters')'! '{'!board_statement_list2 jump_stmt'}'!
                    | FUNC POINT ID '('!parameters')'! '{'!board_statement_list2 jump_stmt'}'!
                    | FUNC ID '('!parameters')'! '{'! board_statement_list2 '}'!
;
parameters : (/* empty */ | var_declaration_unassigned) (','! var_declaration_unassigned)*
;
var_declaration_unassigned : INT ID
```

```
| BOOLEAN ID
;


statement_list : (statement)*
;

board_statement_list2: board_statement_2 board_statement_list2
| /*empty*/
;

board_statement_2: set_structure
| var_declaration
| board_statement
| point_declaration
| path_declaration
;

statement : assgn_expression
| iteration_stmt
| selection_stmt
| point_declaration
| path_declaration
| var_declaration
;


iteration_stmt : WHILE^ '('! expression ')'! '{'! statement_list '}'!
;

selection_stmt   : IF^ '('! expression ')'! '{'! statement_list '}'! selection_else_stmt
                 ;

selection_else_stmt       : ELSE^ '{'! statement_list '}'!
                 |/*empty*/
                 ;

jump_stmt : RETURN expression';'!
;

print_stmt : PRINT '('! STRING ')'! ';'!
;

term     :          TRUE
                           |FALSE
                           | ID
                           | function_call
                           | CONST
                           | '('! expression ')'!
                           ;
```

```
negation
                        : NOT^* term
                        ;
unary
                        :       (unary_minus^)* negation
                        ;

unary_minus
                        : MINUS -> NEGATIONS
                        ;
mult
                        : unary ((TIMES^ | DIVIDE^) unary)*
                        ;

add
        : mult ((PLUS^ | MINUS^) mult)*
        ;
relation
        :       add ((GRTR^ | LESS^ | GRTR_EQL^ | LESS_EQL^ | DBL_EQLS^ | NOT_EQLS^)
add)*
        ;
expression
        : relation ((AND^ |OR^) relation)*
        ;

assgn_expression : ID ASSIGN^ expression ';'!
;

expression_list : expression ( ','! expression)*
                                                        |/*empty*/
;

WS: (' ' | '\t' | '\n' | '\r' | '\f')+ {$channel = HIDDEN;};

COMMENT : '/*' .* '*/' {$channel = HIDDEN;};

MAIN  :'main';

DRAW : 'draw';

STRUCTURE
        :       'structure';

FUNC  :'func';

INT     :'int';

AND    :'&&';

OR      :'||';
```

```
NOT_EQLS:    '!=';
DBL_EQLS:    '==';
LESS_EQL:    '<=';
GRTR_EQL:    '>=';
LESS   :'<';
NOT    : '!';
GRTR   :'>';
WHILE:'while';
ASSIGN       :'=';
MINUS:'-';
PLUS   :'+';
TIMES :'*';
DIVIDE       :'/';
RETURN       :'return';
BOOLEAN      :'boolean';
IF     :'if';
ELSE   :'else';
SET    :'set';
SIZE   :'size';
END    :'end';
START:'start';
FALSE :'false';
TRUE   :'true';
BOARD        :'board';
LEFT   :     'left';
UPP    : 'up';
```

```
DWN : 'down';

RIGHT :'right';

PATH  :'path';

POINT :'point';

PRINT : 'print';

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

ID        : ('a'..'z' | 'A'..'Z')('a'..'z'|'A'..'Z'|'0'..'9')*
                ;

STRING          : '"'('a'..'z'| 'A'..'Z'| '0'..'9' |' ' | '\t' | '\n' | '\r' | '\f')* '"'
            ;
```

10. Possible Further Developments

Some functionalities that we would like to include in our language if we have time are the following:

We would like to implement methods that are related with specific structures and boards to give the user the ability to create more complex structures. An example would be a function like getPoint() to see the availability of the point as well as its type.

We would also like to implement a user interface so that the programmer could actually play the game once they have finished programming.

We would also like to somehow make it possible to create circular mazes.

Chapter 4: Project Plan

Language Development Process

Developing the language:
The team met every Saturday afternoon during semester. We had to first discuss what the purpose of our programming language should be. The initial purpose of the Amaze language was to allow programmers to design interactive maze and puzzle games. However, due to time constraints, we needed to set a realistic goal for our language that we can meet by the time the project deliverables were due. The best choice was to simplify the features and ultimate goal of the language and to add functionality if needed as we continued to work on the project. Amaze became a language whose purpose was for the design and development of mazes.

The next step was deciding on how the programmer will reach this goal and what will make Amaze a programming language and not an application. Realizing that a programming language required certain properties such as control flow, types, functions, and looping constructs, we were able to design an appropriate specification of the Amaze programming language along with the syntax and grammar of the language.

Developing the translator:
During the week of April 29, 2012, the team met everyday.
What the team needed to do next was design a compiler for the Amaze programming language in order have programmers be able to use the language to actually design mazes. The team started by implementing the grammar in ANTLR, a Java parser generator. Implementing the grammar in ANTLR created the lexer and parser for the language.
As the project manager, I started assigning roles to the members, to implement a different area of our programming language. Sample programs that created mazes were written in Java. This was to help the team with the translation by allowing us to discover what connections can be made between Amaze and Java. The compiler was written in Java which included classes from the sample Java programs used for semantic analysis.

Team Member Roles and Responsibilities

Although team member were assign specific roles, it was necessary to step outside of these roles and switch responsibilities in order to accomplish our goals efficiently. Some tasks needed multiple team members for the purpose of error checking and debugging. If one member was busy with a task taking more time than expected, another member would take on some of the responsibility.

All team members worked on the early versions of some documents.

Team Member	***Role***	***Software Code***	***Documentation***
Rouault Francoeur	*Project Manager-* *set up meetings* *keep track of everyone elses progress in addition to own.* *assign responsibilities*	*Grammar* *Java interface code* *Code debugging* *Iteration statements* *Selection statements* *Testing* *makefile*	*White Paper revision* *Project Plan* *LRM revision* *Appendix*
Jonathan Bourdett	*System Architect-* *Implement modules*	*Grammar* *Point and Path types*	*white paper revision* *tutorial revision* *system architect*

		Structures *Board*	
Orlando Pineda	*System Integrator-* *implement interfaces*	*Grammar (Semantics)* *Expressions* *functions* *Code debugging*	*LRM revision* *development and run-time environment*
Daniel Mercado	*Language Guru-* *keep track of changes to language. Keep other members up to date with changes.*	*Grammar* *Code debugging* *Main* *makefiles* *Structures*	*White Paper revision* *language evolution*
Jose Contreras	*Tester/Validator- create test cases and test code*	*Testing* *Iteration statements* *Selection statements*	*Test Plan* *Test cases* *function test*

Implementation style sheet

Committing changes

Make sure code works before committing

Be sure to resolve merge conflicts

Notify others when pushing and pulling from the repository

Comments

Comment out code if it needs to be temporarily taken out only when it is finalized that the coded is no being used.

Comment as often as needed using any method

Names

Keep name understandable and related to the purpose of whatever the name represents

Keep naming style consistent

Start the names of everything other than classes with a lower case letter

Start the names of new classes with a upper case letter

Loops and Conditional statements

Use brackets all the time even if the conditional is one line.

Indentations and Spaces

Make indentation in grammar files for each non-terminal

For each non-terminal, start a new line and indent for every new terminal

Use appropriate amount of spaces to make code readable by other member of the team

Project Timeline

Task	***Start Date***	***End Date***
Complete language white paper	*02/14/12*	*02/21/12*
Complete Language Reference Manual Individual Parts	*03/11/2012*	*03/13/2012*
Complete grammar for language	*03/13/2012*	*03/14/2012*
Complete Language Tutorial Individual Parts	*03/14/2012*	*03/15/2012*
Complete compilation of an revision of tutorial and LRM	*03/26/12*	*03/29/2012*
Complete ANTLR lessons and tutorial	*03/31/12*	*04/07/12*
Complete Scanner and Parser	*04/07/12*	*04/21/2012*
Complete Java code interface and code generation	*04/22/12*	*05/01/12*
Complete Semantic Analysis	*05/03/12*	*05/06/12*
Complete Compiler along with testing	*05/04/12*	*05/06/12*
Complete revision of old documentation	*05/05/12*	*05/05/12*

Complete final documentation	*05/04/12*	*05/06/012*

Project Log

Date	***Task***
02/21/12	*Complete language white paper*
03/13/2012	*Complete Language Reference Manual Individual Parts*
03/14/2012	*Complete grammar for language*
03/15/2012	*Complete Language Tutorial Individual Parts*
03/20/12	*Complete compilation of an revision of tutorial and LRM*
04/28/12	*Complete ANTLR lessons and tutorial*
04/30/12	*Complete Scanner and Parser*
05/01/12	*Complete Java code interface and code generation*
05/03/12	*Complete Semantic Analysis*
05/04/12	*Complete Compiler along with testing*
05/05/12	*Complete revision of documentation*

05/06/12	*Complete final documentation and presentation*

Chapter 5 - Language Evolution :

During the early stages of the Amaze language proposal our language was meant to output a game a user could create and traverse. This proved to be a difficult goal to implement as none of our group members had previous experience programming games. To learn how to program a game in Java would take too long and leave us with little time to work on the compiler. On the other hand our group members were fairly familiar with generating GUIs using the Swing package in Java. As a result we decided not to make the maze a traversable game and to instead just draw the structure of a maze using GUIs in Java. Although our language no longer created a game, we still maintained the creative attributes of our proposal. The language could no longer define rules for customized games, however it could still create interesting and creative maze structures. All other attributes simple, detailed, architecture neutral, intuitive, and educational were all maintained by the language during implementation. As a result our language did deviate far from our proposal.

To create the compiler components of our language we relied heavily on several technologies. After researching different lex and parse generators such as *javacc* or *jflex*, we chose to use ANTLR (ANother Tool for Language Recognition) to handle both of these tasks. Given a grammar ANTLR will generate a lexer and parser for your language and even create an abstract syntax tree for recognizers. ANTLR also has the added functionality of letting users generate a tree parser that can be used to evaluate nodes given by the abstract syntax tree. The tree parse could then generate code based on the node evaluation. Rather than use multiple technologies to generate a compiled program and create an interface between them, we were able to use only one external library to generate all parts of our compiler.

Although documentation was sparse on ANTLR development, we were able to find a great resource created by Scott Stanchfield at http://www.javadude.com/articles/antlr3xtut/ that walked us through the process of using ANTLR to create our compiler. With Scott's help were able to use an Eclipse plug-in known as "ANTLR IDE" to iteratively test the implementation our language within the familiar development environment of Eclipse.

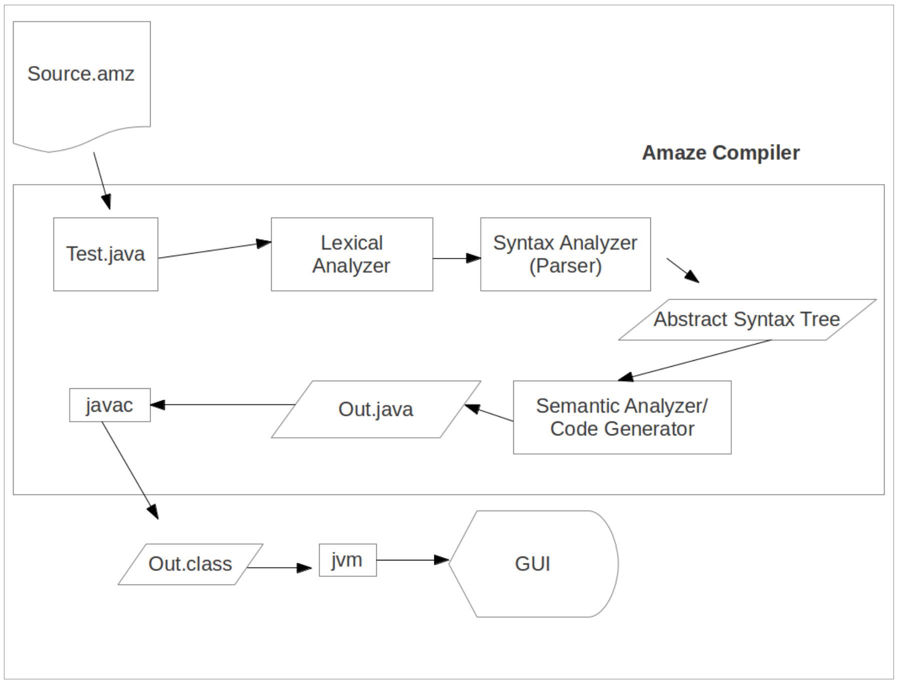
In order to to keep our LRM and compiler consistent, we ensured that the grammar we created would recognize the tokens of our language and generate all productions specified in the LRM. This greatly helped to ensure that our language did not deviate far from the LRM.

During the implementation of code generation within our compiler, many aspects of our language proved frustratingly difficult to translate from our source code to a functional java file. Implementing functions, and creating our derived global data type "structure" were the most pressing concerns about our langauge. Despite our hardships during development we did not remove aspects of our language just to make our lives easier. We endured and were able to remain true to the LRM in our compiler.

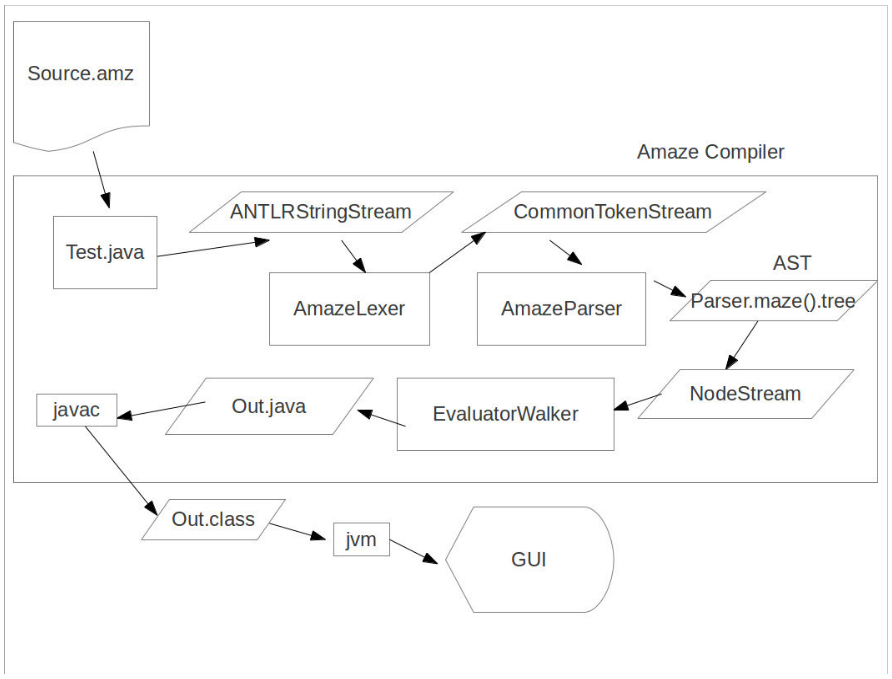
After completing our compiler, we iteratively tested hypothetical Amaze files and ensured they were properly translated to a functional java program. This ensured that the language defined in our LRM remained consistent with our working compiler. Source files that did not generate functional java programs helped us to identify problem areas in the compiler that were later debugged.

Chapter 6 : Translator Architecture

Architectural Block Diagram



Architectural Block Diagram(including intermediate inputs and outputs)



The different modules of the compiler were all contained within Test.java. Test took in the source.amz file and made it in an ANTLRStringStream. This ANTLRStringStream was passed to an instance of AmazeLexer, which output a CommonTokenStream. The CommonTokenStream was passed to the AmazeParser, which output an abstract syntax tree. An instance of EvaluatorWalker then took in a NodeStream created from the AST. This output is a string concatenation of all of the strings from along the way as the EvaluatorWalker walked the grammar.

Lexer/Parser

In terms of the modules, everyone helped to develop the lexer, because that mostly meant hashing out the grammar. Orlando helped to convert the grammar to LL* for use in ANTLR. The parser was also developed from Amaze.g, so again, most of our team members worked to edit the grammar. Syntax analysis was really combined within code generation in EvaluatorWalker.g. The whole group generally developed our individual parts for made checks for validity, but Orlando handled all exceptions being thrown.

Semantic Analysis

All of us generally tried to make checks for semantic analysis, but Orlando made sure of every production one by one and handled exceptions.

Code Generation

The majority of work was done in code Generation, where all of us played a major part, and where work was split up more clearly. The work was mostly in properly bubbling up the minimum information to the parent production. Usually that involved pushing up concatenated strings of code, but occasionally we needed to pass an Identifier to make proper checks.

Rouault worked on iteration and selection statements. In addition, Rouault helped develop and organize many of the classes that interfaced with the Java code generation. These classes include board, path, point, etc. These were the classes that took the final snippets of code and performed the commands we actually wanted in Java.

Dan worked on main declaration, fixed board_statement_2, organized board declaration for proper code generation, and completed structure.

Orlando created many of the node classes that we implemented to deal with bubbling up Strings and Identifiers up the abstract syntax tree. He also worked with int declarations, function declarations, and all of the productions having to do with expressions.

Jonathan worked on path declarations, point declarations, board list and statements(specifically the simple ones, size, start, and end), and just generally bubbling up strings up the tree throughout the more abstract levels in tree.

Jose worked on while loops and if statements in code generation.

Chapter 7: Development and Run-Time Environment

Development Environment Timeline:

We used several different development environments in the construction of our compiler. Below are the different development tools and environments we used in the order in which we used them.

Git/Bitbucket

Perhaps the most useful software development tool and environment we used from start to finish was git. We chose to use bitbucket, a free online repository which has unlimited amounts of private repositories for students. This tool was very useful for not only developing code separately, but for recovering from unfixable errors and other issues such as Eclipse crashes. Having the repository enabled us to keep a working version online at all times and the ability to revert to previous versions of the code. It was also useful since it allowed us to see our commit time line throughout the semester.

Lex and Yacc

When we first began, we thought we would be using C to compile our source code down to Java byte code, and therefore began using compiler development tools targeted for the C language. These tools included lex and yacc. As for the environment at this stage we used normal text editors such as gedit, vim, etc. Soon after, we realized that most of us had little to no experience in developing GUIs with C, and since most of us were comfortable with Java, we decided to switch gears and find a suitable compiler development tool for Java compiler development.

ANTLR Works

The tool which we decided to use for our compiler development was ANTLR. Before starting we wanted to find a suitable development environment. The environment which we began to build our compiler on was ANTLR works, a useful IDE for checking for left recursion, ambiguity, viewing the AST and other forms of debugging. Switching to ANTLR, however, required us to convert our currently working YACC grammar into the special LL(*) EBNF form. This required us to perform the left factoring and left recursion elimination transformations to our grammar. Fortunately, this step was not so difficult since ANTRL works has several tools for eliminating left recursion and points out productions in the grammar where left factoring had to be done.

Eclipse

ANTRLworks was working fine, but we soon stumbled upon a tutorial on how add ANTLR support for Eclipse. Since everyone was comfortable with Eclipse, and the development of supplemental classes and testers could be done in the same IDE, we figured we would switch to Eclipse and use it as our compiler development IDE from then on. In addition, the Eclipse antlr support allowed us to see the tree that was being parsed with its built in interpreter function. This was extremely useful in working out the final bugs in our grammar, to ensure that the source code was being parsed in the correct manner. Finally, during the testing and semantic analysis phases, Eclipse allowed us to run the several tests and view the immediate output java file without having to open another environment.

Makefile

The final development tool we used when developing our Amaze programs, was the following makefile. The makefile used to build and run our compiler is shown below. None of us had any real experience with makefile so we used online resources and examples to learn how makefile works.

src/amazec.sh

```
#!/bin/bash

if [ -f  ./Test.java ]
        then
                javac Test.java
                if [ -f ./Test.class ]
                        then
                                java Test $(readlink -f $1)
                                if [ -f ./Out.java ]
                                        then
                                                javac Out.java
                                                echo $(readlink -f $1)
                                        else
                                                echo "Compiler error"
                                fi
                fi
fi

#javac Test.java
#java Test $1
#javac Out.java
#java Out
```

Outputs Out.class to be ran by java

Compiler Run Time Environment

In order for a user to compile and run our program, the user's machine must contain the Java Development Kit. Once our compiler converts our program source code, the java byte code is interpreted by the java virtual machine. The final java byte code references required .class files which need to be reachable from the java byte code for the output file, which will be generated along with the main .class file. Run time errors are handled by the java virtual machine.

Chapter 8: Test Plan

In order to have progress in the development of any project the testing has to occur at every step of the development. We implemented white-box testing, black-box, visual testing and regression testing in the development our compiler. Each of the different steps that we took as a group in ensuring working code was crucial because of the potential for issues that might arise in compiling our language into Java and in the merger of our project. We used git and gitbucket for version control allowing us to revert to previous builds if changes we commited broke previous working source code.

White-box Testing:

We implemented white-box testing from the initial development of our language. When first creating the grammar of our language we began testing our language so that it would be able to recognize multiple Amaze files that we created for testing purposes. Having a grasp of the structure of our language through testing, trial, and error we were able to develop a function compiler. In order to test that all the components of our language were working properly, we wrote multiple amaze files with varying paths and that used functions in order to ensure that everything was working properly. We created amaze files that were purposely created to create bugs in our code in order to see if our compiler would recognize the errors and process them accordingly. We purposely inserted incorrect data types in loops, if-statements, and in the parameters of functions and observed as our language reported the errors.

Sample Code: Inserting bug
```
board board1
{
       size: 19, 19;
       start: 1, 0;
       end: 18, 17;
       int x = 0;
       while(x)
       {
         x=10;
       }

}
main
{
       draw(board1);
}
```

```
Expected output:
“Invalid type in while loop. Expected a boolean but received a(n) int”
```

Black-box Testing:

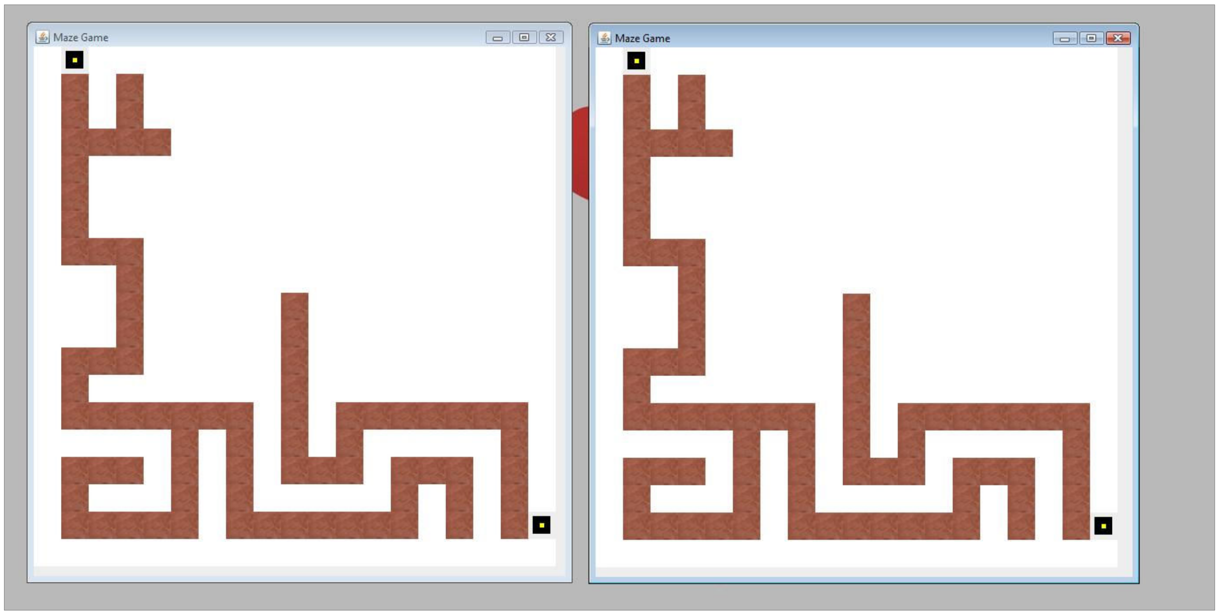
For our group we had a few ways to make sure things were working properly as expected by continuous testing any changes we made and making sure that they did not break anything that we had previously developed. I wrote a java program that generate randomly generated mazes in order to test if our language would react properly if essentially random paths were being generated on the maze If these paths were incorrect amaze is able to inform us of the error and thus we were able to make sure that path declarations were functioning as expected.

Example Code: Testing if Structures creation:

```
structure structs1
{
	point a: 1, 0;
	path a1: a, down, 7;
	point b: 3,7;
	path b1:b, left, 2;
	path b2t3:b, down, 4;
	point c: 1, 11;
	path c1: c, right, 2;
	path c2: c, down, 2;
	point d: 3,3;
	path d1: d, up, 2;
	path d2: d, left,2;
	point e: 1,9;
	path e1: d, right,1;
	point f: 5,13;
	path f1: f,left,4;
	path f2: f,down,4;
	path f3: f,right,2;
	point g: 1, 17;
	path g1: g,up,2;
	path g2: g,right,3;
	point h: 3,15;
	path h1: h, left,1;
	point i: 7,17;
	path i1: i, up, 3;
	path i2: i, right,6;
	point j: 13, 15;
	path j1: j, down, 1;
	path j2: j, right, 2;
	point k: 15,17;
	path k1: k, up, 1;
	point l: 17,17;
	path l1: l, right, 1;
	path l2: l, up, 4;
	point m: 11,13;
	path m1: m, right , 5;
	path m2: m, down, 2;
```

```
        point n: 9,15;
        path n1: n, right, 2;
        path n2: n, up, 6;
}
board board1
{
        size: 19, 19;
        start: 1, 0;
        end: 18, 17;

        set (structs1);
}
board board2
{
        size: 19, 19;
        start: 1, 0;
        end: 18, 17;

        set (structs1);
}
main{
        draw (board1);
        draw (board2);
}
```

```
Expected Output:
Two boards with the same structure drawn
```



Unit Testing and Integration:

It was essential for our group to evoke unit testing we broke off the walker into pieces in which each of us was in charge of a partition of the walker slowly developing our translation of the language. Upon completion of module we would combine the units of the code that were working while testing them. Continuously testing the modules and making sure that each part of our source code was functioning was essential in being able to deliver a final product.

Regression Testing:

In cases where there was an overlap in the source code that we developed we had to use regression testing. When source code that we created was not compatible with each we solved the issues by manipulating the language and add rules that we had applied. In the tree walker as I mentioned before we split up parts of the source and this led to some issues in the development that required regression testing. The merge conflicts that were created when extensive changes were being made to the code sometimes caused errors that busted fixes that had been made to the most update version of our compiler.

Chapter 9: Conclusions

Lessons Learned

Team

As a team, we concluded that we needed to take the time early in the semester more seriously. While we did meet regularly, we did not regularly set deliverables to keep ourselves on pace. At the same time, we didn't really plan ahead in the semester to say that on a certain day, we should have this done.

Lessons Learned By Team Member Rouault Francoeur

As the project manager of the team, I believe that there were many lessons to be learned. This is mostly because it was my first experience as a project manager on a team. Throughout my academic career, I have had roles as a tester, verifier, and found myself doing all sorts of implementation while taking orders from someone else for my projects. Now, for the final project of my academic career before I graduate, I have finally taken the role of project manager.

I have learned that the role of a project manager is not as easy as one may think. I have learned that proper communication is the key to managing a project. As the project progressed, I found myself having to send more emails to make sure that everyone was on track and keeping up with their parts and also let the other members aware of this information.

I have also learned that just because I am the project manager, it does not mean I have the final say in all the decisions. As a team, we should view each other as equals and I should be able to ask for advice and discuss final decisions with other team members.

A problem that I discovered that I had was that I was too soft at times. I learned that, although I want everyone in the team working in a comfortable environment, there are times when I must be strict. During the beginning of the project when one of the members was not showing up to meetings or communicating well with the group, I addressed the issue with the individual, but did not discuss the seriousness and was not strict which led to the individual once again failing to show his commitment as a team member. I finally learned that I must be take more action, so I sent an email explaining the problem and how this problem will be brought to the professor's attention if not solved. As a result, the individual's participation improved and he was able to produce satisfying results on his part.

I find that the most important lesson is that I must learn to adapt and make changes because not everything will run smoothly throughout the lifetime of the project. Deadlines may have to be changed and team member responsibilities may have to be changed.

If I ever become a project manager sometime in the future, I believe that these lessons learned will help me be more prepared to take on that responsibility.

Lessons Learned By Team Member Orlando Pineda

Perhaps one of the most important lessons I learned in doing this project is that when working as a team, is that it takes planning to ensure that at all stages of the program, all group members are active and know exactly what they are doing. Our biggest issue was splitting the work as we progressed through the stages of the compiler. At certain points (especially the beginning) there were instances where the work was only being done by a few of the group members.

Another really important lesson I learned is that effective communication is key amongst the group members. This applies to methods and decisions that individual group members worked on before combining the different parts that are being worked on simultaneously. It allows each member to get a good grasp of what the other team members are doing so that they can attempt their portion of the program in a way that correlates with what the other team members are doing. We had a few issues with this when different members were using different representations during translation. This also means communicating about issues other than the program.

Finally the most important lesson was probably to learn your team's capabilities and take this into consideration to ensure that ideas are not too ambitious or too simple, and when choosing between the two it's probably better to go with the latter. Towards the beginning we were having trouble coming up with a suitable project because we did not know each others capabilities, and were coming out with ideas that varied in difficulty (some would probably not have been possible).

Lessons Learned By Team Member Jonathan Bourdett

If you are working on your own part of a project while in a team meeting, then you may need to distance yourself from the team mentally for a short time to concentrate on that part. I remember specifically attempting to really focus on doing one part, but I ended up getting interrupted with my teammates questions about other things. You may need to make yourself unavailable for a short time.

Always underestimate how much you think you can get done and go with that. I didn’t think we would need as much time as we needed to actually complete the compiler.

Ask friends in the class how they are doing. We got really good insight from other teams, and we felt reassured when we were both using the same technologies or trying the same things.

Don’t be afraid to use new tools or reject old tools. Sometimes there is a better technology out there.

Take goals for the compiler in the early part of the semester more seriously. I felt like we did start early, but we didn’t necessarily push ourselves to have a certain deliverable by a certain date.

Lessons Learned by Team Member Daniel Mercado:
Learning how to properly use a version control program during your project development will help you to avoid headaches during the implementation of your compiler.

Grouping up with friends or people you know you'll work well with helps to create a pleasant work environment and helps to avoid confrontations with other group members.

Although it is often repeated, start early on the actual coding of your project. This will help avoid back to back sleepless nights of programming. In addition you will lighten your workload at the end of the semester and make studying for finals easier.

If no goals are set for the next meeting, speak up and suggest some. Try to move the project forward if at all possible. This will help to avoid procrastination by your fellow group members and reveal problems early.

Lessons Learned by Team Member Jose Contreras
The team has to meet regularly in order to be able to keep communication. At each meeting the project progressed substantially thus any meeting missed was a set back for the team and the member who missed the lab. Communication during development is always necessary because during the development of our walker some of us expected certain implemented things to be necessary. The groups must start early otherwise you will have a lot of work that piles ups and if not started early the groups will suffer at the end of the semester.

Advice for Future Teams

We would advise teams in PLT in future semesters to not only start as early as possible, but also to set and reach milestones/goals as fast as possible. Iteration development seems to be the best way to go about this complex long-term project.

Continually visiting the Professor and TA’s is also a good idea. It lets them know of your progress, and you know immediately if you are on the right track.

Suggestions for Course Topics for Future Semesters

As a team, we thought that the topics of the course were appropriate. However, we think that more time should be spent on topics that will aid and guide the teams in the development of their projects.

Chapter 10: Appendix

Source Code

AdditionNode-Pineda

```
//package compiler;

public class AdditionNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


	public AdditionNode(Object x , Object y){
		int rvalue = (Integer) x + (Integer)y;
		value = rvalue;
		code = rvalue;
	}
	@Override
	public Object value() {
		// TODO Auto-generated method stub

		return value;
	}
	public String getType(){
		return "int";
	}
	@Override
	public String getCode() {
		// TODO Auto-generated method stub
		return String.valueOf(code);
	}
	@Override
	public String getReturnType() {
		// TODO Auto-generated method stub
		return "int";
	}

}
```

Amaze.g-Francoeur, Mercado,Pineda, Bourdett

```
grammar Amaze;

options {

language = Java;
output = AST;
ASTLabelType = CommonTree;
}
tokens{
       NEGATIONS ;
       REASSIGN;

}
@header{
// package compiler;
}

@lexer::header{
// package compiler;
}

maze : declaration_list
main_declaration
// declaration_list
EOF!
;

declaration_list : declaration declaration_list
|/*empty*/
;

declaration : structure_declaration
| board_declaration
| var_declaration
| function_declaration
;

board_declaration: BOARD ID '{'! board_declaration_list '}'!
;


board_declaration_list: size_declaration start_declaration end_declaration
board_statement_list2 /*board_statement_list*/
;
size_declaration : SIZE ':'! expression ','! expression';'!
;
start_declaration : START ':'! expression','! expression';'!
;

end_declaration : END ':'! expression','! expression';'!
;

var_declaration : int_declaration
| bool_declaration
;

int_declaration : INT ID ';'!
             | INT ID ASSIGN^ expression ';'!
             ;

bool_declaration : BOOLEAN ID ';'!
              | BOOLEAN ID ASSIGN^ expression ';'!
              ;
set_structure : SET '('! ID ')'! ';'!
;


point_declaration : POINT ID ':'! expression ','! expression ';'!
| POINT ID ';'!
;

path_declaration : PATH ID ':'! (ID | function_call) ','! direction','! expression
';'!
| PATH ID';'!
;

board_statement :
board_iteration_stmt
| function_call
| board_selection_stmt
| print_stmt
| assgn_expression
;
board_iteration_stmt : WHILE^ '('! expression ')'! '{'! board_statement_list2 '}'!
;

board_selection_stmt : IF^ '('! expression ')'! '{'! board_statement_list2 '}'!
board_selection_else_stmt
;
board_selection_else_stmt : ELSE^ '{'! board_statement_list2 '}'!
|/*empty*/
;
function_call: ID'(' expression_list ')'
                                            ;
direction : LEFT
| RIGHT
| UPP
| DWN
;

main_declaration : MAIN '{'! main_declaration_list '}'!
              ;

main_declaration_list : DRAW '('! ID ')'! ';'! main_declaration_list
              |/*empty*/
              ;


structure_declaration : STRUCTURE ID '{'! structure_body '}'!
;

structure_body :
structure_statement structure_body
| /*empty*/
;

structure_statement : assgn_expression
```

```
| point_declaration
| structure_path_declaration
| var_declaration
| structure_board_statement
;
structure_board_statement :
structure_iteration_stmt
| structure_selection_stmt
|print_stmt
| function_call
;
structure_path_declaration : PATH ID ':'! (ID | function_call) ','! direction','!
expression ';'!
| PATH ID';'!
;
structure_iteration_stmt : WHILE^ '('! expression ')'! '{'! structure_body '}'!
;

structure_selection_stmt : IF^ '('! expression ')'! '{'! structure_body '}'!
structure_selection_else_stmt
;
structure_selection_else_stmt : ELSE^ '{'! structure_body '}'!
|/*empty*/
;

function_declaration : FUNC INT ID'('!parameters')'! '{'!board_statement_list2
jump_stmt'}'!
              | FUNC BOOLEAN ID'('!parameters')'! '{'!board_statement_list2
jump_stmt'}'!
              | FUNC POINT ID '('!parameters')'! '{'!board_statement_list2
jump_stmt'}'!
              | FUNC ID '('!parameters')'! '{'! board_statement_list2 '}'!
;
parameters : (/* empty */ | var_declaration_unassigned) (','!
var_declaration_unassigned)*
;
var_declaration_unassigned : INT ID
| BOOLEAN ID
;


statement_list : (statement)*
;

board_statement_list2: board_statement_2 board_statement_list2
| /*empty*/
;

board_statement_2: set_structure
| var_declaration
| board_statement
| point_declaration
| path_declaration
;
```

```
statement : assgn_expression
| iteration_stmt
| selection_stmt
| point_declaration
| path_declaration
| var_declaration
;


iteration_stmt : WHILE^ '('! expression ')'! '{'! statement_list '}'!
;

selection_stmt      : IF^ '('! expression ')'! '{'! statement_list '}'!
selection_else_stmt
              ;

selection_else_stmt : ELSE^ '{'! statement_list '}'!
              |/*empty*/
              ;

jump_stmt : RETURN expression';'!
;

print_stmt : PRINT '('! STRING ')'! ';'!
;

term   :      TRUE
                    |FALSE
                    | ID
                    | function_call
                    | CONST
                    | '('! expression ')'!
                    ;

negation
                    : NOT^* term
                    ;
unary
                    :     (unary_minus^)* negation
                    ;

unary_minus
                    : MINUS -> NEGATIONS
                    ;
mult
                    : unary ((TIMES^ | DIVIDE^) unary)*
                    ;

add
      : mult ((PLUS^ | MINUS^) mult)*
      ;
relation
      :     add ((GRTR^ | LESS^ | GRTR_EQL^ | LESS_EQL^ | DBL_EQLS^ | NOT_EQLS^)
add)*
      ;
```

```
expression
       : relation ((AND^ |OR^) relation)*
       ;

assgn_expression : ID ASSIGN^ expression ';'!
;

expression_list : expression ( ','! expression)*
                                                      |/*empty*/
;

WS: (' ' | '\t' | '\n' | '\r' | '\f')+ {$channel = HIDDEN;};

COMMENT : '/*' .* '*/' {$channel = HIDDEN;};

MAIN   :'main';

DRAW   : 'draw';

STRUCTURE
       :      'structure';

FUNC   :'func';

INT    :'int';

AND    :'&&';

OR     :'||';

NOT_EQLS:    '!=';

DBL_EQLS:    '==';

LESS_EQL:    '<=';

GRTR_EQL:    '>=';

LESS   :'<';

NOT    : '!';

GRTR   :'>';

WHILE  :'while';

ASSIGN :'=';

MINUS  :'-';

PLUS   :'+';

TIMES  :'*';

DIVIDE :'/';
```

```
RETURN :'return';

BOOLEAN      :'boolean';

IF     :'if';

ELSE   :'else';

SET    :'set';

SIZE   :'size';

END    :'end';

START  :'start';

FALSE  :'false';

TRUE   :'true';

BOARD  :'board';

LEFT   :      'left';

UPP    : 'up';

DWN : 'down';

RIGHT  :'right';

PATH   :'path';

POINT  :'point';

PRINT  : 'print';

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

ID     : ('a'..'z' | 'A'..'Z')('a'..'z'|'A'..'Z'|'0'..'9')*
              ;

STRING : '"'('a'..'z'| 'A'..'Z'| '0'..'9' |' ' | '\t' | '\n' | '\r' | '\f')* '"'
        ;
```

AndNode - Pineda

```
//package compiler;

public class AndNode implements ExpressionNode {
       ExpressionNode node1;
       ExpressionNode node2;
       Object value;
```

```
	Object code;


public AndNode(Object x , Object y){
	boolean rvalue = Boolean.parseBoolean(String.valueOf(x)) &&
Boolean.parseBoolean(String.valueOf(y));
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
	return "boolean";
}
@Override
public String getCode() {
	// TODO Auto-generated method stub
	return String.valueOf(code);
}
@Override
public String getReturnType() {
	// TODO Auto-generated method stub
	return "boolean";
}

}
```

Board – Francoeur

```
//package codeGeneration.fred;

import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Point;
import java.util.ArrayList;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class Board {
	private int size_x;
	private int size_y;
//	private int end_x;
//	private int end_y;
//	private int start_x;
//	private int start_y;
```

```
	private Matrix matrix;
	private 	JFrame frame = new JFrame("Maze Game");
	private JPanel panel;

	public Board(int size_x, int size_y){
		this.size_x = size_x;
		this.size_y = size_y;
		this.panel = new JPanel(new GridLayout(size_x, size_y));
		//removes run time error after changing dimension
		System.setProperty("java.util.Arrays.useLegacyMergeSort", "true");
		Dimension boardDim = new Dimension(600, 600);
		panel.setPreferredSize(boardDim);

		this.matrix = new Matrix(size_x, size_y);

	}

	public Board(Point xy){
		this.size_x = (int) xy.getX();
		this.size_y = (int) xy.getY();
		this.panel = new JPanel(new GridLayout(size_x, size_y));
		//removes run time error after changing dimension
		System.setProperty("java.util.Arrays.useLegacyMergeSort", "true");
		Dimension boardDim = new Dimension(600, 600);
		panel.setPreferredSize(boardDim);

		this.matrix = new Matrix(size_x, size_y);

	}

	public void addStart(int start_x, int start_y){
//		this.start_x = start_x;
//		this.start_y = start_y;
		matrix.setStart(start_x,start_y);
	}

	public void addStart(Point xy){
//		this.start_x = start_x;
//		this.start_y = start_y;
		matrix.setStart((int) xy.getX(), (int) xy.getY());
	}

	public void addEnd(int end_x, int end_y){
//		this.end_x = end_x;
//		this.end_y = end_y;
		matrix.setEnd(end_x,end_y);
	}

	public void addEnd(Point xy){
//		this.end_x = end_x;
//		this.end_y = end_y;
		matrix.setEnd((int) xy.getX(), (int) xy.getY());
	}

	public void addPath(Path path){
```

```
            matrix.insertPath(path.point, path.direction, path.length);
        }

        public void addStructure(Structure structure){
            ArrayList<Path> path = structure.getPaths();
            for(int i = 0; i < path.size(); i++){
                addPath(path.get(i));
            }
        }

        public void drawBoard(){
            ImageIcon tileIcon = new ImageIcon("tile.jpg");
            JLabel temp;

            ImageIcon blankIcon = new ImageIcon("blank.png");
            JLabel temp2;

            ImageIcon specIcon = new ImageIcon("pacman_tile.png");
            JLabel temp3;
            //draw tiles based on matrix
            int k = 0;
            for (int i=0; i<size_x; i++){
                for (int j=0; j<size_y; j++){
                    if(matrix.isStart(i, j) || matrix.isEnd(i,j)){
                        temp3 = new JLabel(specIcon);
                        panel.add(temp3,k++);
                    }
                    else if (matrix.isActive(i, j)){
                        temp = new JLabel(tileIcon);
                        panel.add(temp, k++);
                    }
                    else{ //point is blank
                        temp2 = new JLabel(blankIcon);
                        panel.add(temp2, k++);
                    }

                }
            }

            frame.add(panel);
//code to finish up frame
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.pack();
            frame.setVisible(true);
        }
}
```

BoardDeclarationListNode – Bourdett

```
//package compiler;

import java.util.ArrayList;

public class BoardDeclarationListNode {
	ArrayList<String> sizeStartEndList;
	String pointsAndPaths;

	public BoardDeclarationListNode(ArrayList<String> sizeStartEndList, String 
pointsAndPaths) {
		this.sizeStartEndList = sizeStartEndList;
		this.pointsAndPaths = pointsAndPaths;
	}

	public ArrayList<String> getSizeStartEndList() {
		return sizeStartEndList;
	}

	public void setSizeStartEndList(ArrayList<String> sizeStartEndList) {
		this.sizeStartEndList = sizeStartEndList;
	}

	public String getPointsAndPaths() {
		return pointsAndPaths;
	}

	public void setPointsAndPaths(String pointsAndPaths) {
		this.pointsAndPaths = pointsAndPaths;
	}



}
```

BoardStatementList2Node – Bourdett, Pineda

```
//package compiler;

import java.awt.Point;
import java.util.ArrayList;

//import codeGeneration.fred.Path;

public class BoardStatementList2Node {

	private ArrayList<Path> paths;
	private ArrayList<Point> points;

	public BoardStatementList2Node(ArrayList<Path> paths, ArrayList<Point> 
points){
		this.paths = paths;
		this.points = points;
	}

	public ArrayList<Path> getPaths() {
		return paths;
	}

	public void setPaths(ArrayList<Path> paths) {
		this.paths = paths;
	}

	public ArrayList<Point> getPoints() {
		return points;
	}

	public void setPoints(ArrayList<Point> points) {
		this.points = points;
	}

}
```

BoardStatementListNode – Bourdett, Pineda

```
//package compiler;

import java.awt.Point;
import java.util.ArrayList;

//import codeGeneration.fred.Path;

public class BoardStatementListNode {

	private ArrayList<Point> sizeStartEndList;
	private ArrayList<Point> points;
	private ArrayList<Path> paths;

	public BoardStatementListNode(ArrayList<Point> sizeStartEndList,
ArrayList<Point> points, ArrayList<Path> paths){
		this.sizeStartEndList = sizeStartEndList;
		this.points = points;
		this.paths = paths;
	}

	public ArrayList<Point> getSizeStartEndList() {
		return sizeStartEndList;
	}

	public void setSizeStartEndList(ArrayList<Point> sizeStartEndList) {
		this.sizeStartEndList = sizeStartEndList;
	}

	public ArrayList<Point> getPoints() {
		return points;
	}

	public void setPoints(ArrayList<Point> points) {
		this.points = points;
	}
	public ArrayList<Path> getPaths() {
		return paths;
	}

	public void setPaths(ArrayList<Path> paths) {
		this.paths = paths;
	}

}
```

BSL2Node – Bourdett, Pineda

```
//package compiler;

import java.util.ArrayList;

//import codeGeneration.fred.Path;

public class BSL2Node {

	String string;
	ArrayList<String> pathIDs;

	public BSL2Node(String string, ArrayList<String> pathIDs){
		this.string = string;
		this.pathIDs = pathIDs;
	}

	public String getString() {
		return string;
	}

	public void setString(String string) {
		this.string = string;
	}

	public ArrayList<String> getPathIDs() {
		return pathIDs;
	}

	public void setPathIDs(ArrayList<String> pathIDs) {
		this.pathIDs = pathIDs;
	}
}
```

DivNode - Pineda

```
//package compiler;

public class DivNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


public DivNode(Object x , Object y){
	int rvalue = (Integer)x / (Integer)y;
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
	return "int";
}
@Override
public String getCode() {
	// TODO Auto-generated method stub
	return String.valueOf(code);
}
@Override
public String getReturnType() {
	// TODO Auto-generated method stub
	return "int";
}


}
```

EqualNode - Pineda

```
//package compiler;

public class EqualNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


public EqualNode(Object x , Object y){
	boolean rvalue = (Integer)x == (Integer)y;
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
      return "boolean";
}
@Override
public String getCode() {
      // TODO Auto-generated method stub
      return String.valueOf(code);
}
@Override
public String getReturnType() {
      // TODO Auto-generated method stub
      return "boolean";
}


}
```

EvaluatorWalker.g – Francoeur, Bourdett, Mercado, Pineda, Contreras

```
tree grammar EvaluatorWalker;

options {
      //k=1;
      backtrack = true;
  language = Java;
  tokenVocab = Amaze;
  ASTLabelType = CommonTree;
}

@header {
//  package compiler;
//  import compiler.*;

  import java.util.HashMap;

  import java.util.ArrayList;
  import java.awt.Point;
//  import codeGeneration.fred.Board;
//  import codeGeneration.fred.Path;
//  import codeGeneration.fred.Structure;

  import java.io.FileWriter;
  import java.io.BufferedWriter;
```

```
}

@members {
  private HashMap<String, Object> variables2 = new HashMap<String, Object>();
  private HashMap<String, Identifier> variables = new HashMap<String, Identifier>();
          private ArrayList<String> function_list = new ArrayList<String>();
          private ArrayList<String> global_variables = new ArrayList<String>();

}


maze returns [String temp]
                              :  d= declaration_list
                                m=main_declaration
      {
                    temp =
                          "import java.awt.Dimension;" + '\n' +
                                                                                  "import java.awt.GridLayout;" + '\n'+
                                                                                  "import java.awt.Point;" + '\n' +
                                                                                  "import javax.swing.ImageIcon;" +
'\n' +
                                                                                  "import javax.swing.JFrame;" + '\n'
+
                                                                                  "import javax.swing.JLabel;" + '\n' +
                                                                                  "import javax.swing.JPanel;" + '\n' +
                                                                                  "import java.util.ArrayList;" + "\n" +
//                                                                                "import
codeGeneration.fred.Board;\n" +
//                                                                                "import
codeGeneration.fred.Path;\n\n" +
                                                                                  "public class Out {" + '\n' ;
                                                                                  for(int i = 0; i <
global_variables.size(); i++){
                                                                                                          temp += "
                    " + global_variables.get(i) + '\n';
                                                                                          }
                                                                                  temp += "\t" + '\n' + m + "\t" + '\n' + d
+ '\n';    ///* + "public static void main(String[] args) {"*/
//                                          for (String aBoard: m)
//                                              temp = temp + aBoard + ".draw();\n";


                       for( int i = 0; i < function_list.size(); i ++){
                         temp += '\n' + function_list.get(i);
```

```
                }

                temp +=  '\n' + "}";

            }

        ;


declaration_list returns [String r]

            : d1=declaration d2=declaration_list
                {if (d2==null){
                    if (d1==null){
                      r = "";
            }
                    else{
                      r = d1;
                    }
                }
                else{
                    if (d1==null){
                      r = d2;
                    }
                    else{
                      r = d1 + d2;
                    }
                }

                }
  | /*empty*/
;

declaration returns [String r]
                                                   :b=board_declaration { r = b;}
            | f=function_declaration {}//{r = "";}
                    | v=var_declaration { global_variables.add("public static " +  v); }
//          | s1=struct_declaration { r = s1;}
            | s2=structure_declaration {r = s2;}
;

//struct_declaration returns [Identifier myID]
//                  : point=point_declaration {/*myID = new Identifier("point", point, "point");*/}
//                  | path=path_declaration {/*myID = new Identifier("path", path, "path");*/}
//                ;

point_declaration returns [String myPointString] : //Point
 POINT ID e1=expression e2=expression
                    {
                      //first make sure e1 is an integer
                      if (!e1.getReturnType().equals("int")){
                        throw new Exception("Point expects an int but has received a " + e1.getType()
                          + " as its 1st parameter");

                        //Throw some sort of exception.

                      }
                      if (!e2.getReturnType().equals("int")){
                        throw new Exception("Point expects an int but has received a " + e2.getType()
                          + " as its 2nd parameter");



                      }

                      myPointString = "Point " + $ID.text + " = new Point(" + e1.getCode() + ", " +
e2.getCode() + ");\n";
                      variables.put($ID.text, new Identifier( $ID.text, $ID.text, "point"));
                    }
                  | POINT ID
                  ;
                      catch[Exception error]{
                      System.out.println( error.getMessage());
                              System.exit(1);
                      }


path_declaration returns [PathNode myPath] //Path //String
: PATH i1=ID i2=term d=direction e=expression
                    {
                        if (i2.getType().equals("function") &&
variables.containsKey(((FunctionNode)i2).Identifier) && e.getReturnType().equals("int")){

                        String myPathString = "Path " + $i1.text + " = new Path(" +
((FunctionNode)i2).getCode() + ", \"" + d + "\", " + e.getCode() + ");\n";

                        myPath = new PathNode(myPathString, $i1.text, ((FunctionNode)i2).Identifier);
                      }
```

```
                    else if (i2.getType().equals("function") &&
((FunctionNode)i2).getReturnType().equals("point")&& e.getReturnType().equals("int")){
                     String myPathString = "Path " + $i1.text + " = new Path(" + i2.getCode() + ", \"" +
d + "\", " + e.getCode() + ");\n";
                     myPath = new PathNode(myPathString, $i1.text, i2.getCode());

                    }
                    else {throw new Exception("Path Expects a point");

                    }
                    }
                  | PATH ID
                  ;
             catch[Exception error]{
                     System.out.println(error.getMessage());
                          System.exit(1);
                     }


direction returns [String myDirection]
          : LEFT {myDirection = $LEFT.text;}
          | RIGHT {myDirection = $RIGHT.text;}
          | UPP {myDirection = $UPP.text;}
          | DWN {myDirection = $DWN.text;}
          ;

var_declaration returns [String temp]
                     : i=int_declaration {temp =i;}
                     | b=bool_declaration { temp = b;}
                     ;

function_call returns [ExpressionNode r]

            : ID  '(' e=expression_list ')'
            {
            if ( variables.containsKey($ID.text)){
              String code = $ID.text + "(" + e + ")";
              r = new FunctionNode(code ,
((ExpressionNode)variables.get($ID.text).value).getReturnType(), $ID.text);
            }
            else {throw new Exception( "Function " + $ID.text + "Not Declared");

            }
            }
            ;
        catch[Exception error]{
                System.out.println(error.getMessage());
                        System.exit(1);
                }


expression_list returns [String code]
                                                                : e=expression  e2=expression_list
                                                                {
                                                                if(e2.isEmpty()) code = e.getCode();
//+ e2;
                                                                else
                                                                code = e.getCode()
                                                                + "," +  e2;

                                                                }
                                                                |/*empty*/ {code = "";}
            ;

bool_declaration        returns [String temp]
                : BOOLEAN c=ID {variables.put($ID.text, new Identifier($c.text, 0, "boolean"));
                   temp = "boolean " + $c.text + ";";}
                | ^(ASSIGN BOOLEAN c=ID e=expression) {
                if(!variables.containsKey($c.text) && e.getType().equals("boolean"))
   variables.put($c.text, new Identifier($c.text, new ValueNode(e.getCode(), "boolean"),
"boolean"));
   else if (!variables.containsKey($c.text) && e.getType().equals("function") &&
e.getReturnType().equals("boolean"))                   variables.put($c.text , new Identifier
($c.text, new FunctionNode("(" + e.getCode() + ")", "boolean"), "function"));
                else throw new Exception("Boolean Expected");
                temp = "boolean " + $c.text + " = " + e.getCode() + " ;";

                }
                ;
        catch[Exception error]{
                System.out.println(error.getMessage());
                        System.exit(1);
                }


int_declaration returns [String temp]
                : INT c=ID  {variables.put($ID.text, new Identifier($c.text, new ValueNode(0,
"int"), "int"));
```

```
                temp = "int " + $c.text + ";";

                }
                | ^(ASSIGN INT c=ID e=expression ) {
                temp = "int " + $c.text + " = " + e.value()+ ";\n";
                //if(!variables.containsKey($c.text) && e.getReturnType().equals("int")){
                if ( e.getType().equals("int"))
                variables.put($c.text, new Identifier($c.text, e, "int"));
                else if (e.getType().equals("function") && e.getReturnType().equals("int")){
   variables.put($c.text, new Identifier ($c.text, new FunctionNode(e.getCode(), "int", $c.text),
"function"));

                 }
                 else{throw new Exception("int expected");

    }}//}
                ;
                        catch[Exception error]{
                System.out.println(error.getMessage());
                        System.exit(1);
                }



expression returns [ExpressionNode r]
        /* All of the mathmatical expressions. Also Handles Function Calls */
        : ^(PLUS c=expression d=expression)
                 {
                if ( c.getType().equals("int") && d.getType().equals("int")){
                r = new AdditionNode(c.value(), d.value());
                }
                else  if (d.getType().equals("function") && d.getReturnType().equals("int")){
                                        r = new FunctionNode( c.value() + " + " + d.getCode(),
"int");

                                }
                                else  if (c.getType().equals("function") &&
c.getReturnType().equals("int")){
                                        r = new FunctionNode( c.getCode() + " + " + d.value() ,
"int");

                                }
                        else{throw new Exception("InputMismatch error");
                         }
```

```
                    }

        | ^(MINUS c=expression d=expression){
                if ( c.getType().equals ("int") && d.getType().equals("int")){
                r = new SubNode(c.value(), d.value());}
                else  if (d.getType().equals("function")&& d.getReturnType().equals("int")){
                                        r = new FunctionNode( c.value() + " - " + d.getCode(),
"int");
                                }
                                else  if (c.getType().equals("function")&&
c.getReturnType().equals("int")){
                                        r = new FunctionNode( c.getCode() + " - " + d.value() ,
"int");

                                }
                        }
        | ^(TIMES c=expression d=expression){
                        //throw new Exception("mult and mult2" + c.value() + d.value());
                                if ( c.getType().equals("int") && d.getType().equals( "int")){
                                r = new MultNode(c.value(), d.value());
                                }
                                else  if (d.getType().equals("function")&&
d.getReturnType().equals("int")){
                                        r = new FunctionNode( c.value() + " * " + d.getCode(),
"int");

                                }
                                else  if (c.getType().equals("function")&&
c.getReturnType().equals("int")){
                                        r = new FunctionNode( c.getCode() + " * " + d.value(),
"int");
                                }
                                else {
                                        throw new Exception("Incorrect Type. Int expected");

                                }
                        }
        |^(DIVIDE c=expression d=expression){
                        if ( c.getType().equals("int") && d.getType().equals("int")){
                                r = new DivNode(c.value(), d.value());
                                }
                                        else  if (d.getType().equals("function")&&
d.getReturnType().equals("int")){
```

```
                                        r = new FunctionNode( c.value() + " / " + d.getCode(),
"int");

                                }
                                else  if (c.getType().equals("function")&&
c.getReturnType().equals("int")){
                                        r = new FunctionNode( c.getCode() + " / " +  d.value(),
"int");

                                }
                                else {
                                        throw new Exception("Incorrect Type. Int expected");

                                }

                        }
        | ^(NEGATIONS d=expression){
                                if ( d.getType().equals("int")){
                                r = new NegationNode(d.value());
                                }
                                else  if (d.getType().equals("function")&&
d.getReturnType().equals("int")){
                                        r = new FunctionNode( "-" + d.getCode(), "int");

                                }
                                else {
                                        throw new Exception("Incorrect Type. Int expected");

                                }
                        }
        /*All of the Logical Operations. */
        | ^(NOT d=expression)  {
                                if ( d.getType().equals("boolean")){
                                r = new NotNode(d.value());
                                }
                                else  if (d.getType().equals("function")&&
d.getReturnType().equals("boolean")){
                                        r = new FunctionNode( "!" + d.getCode(), "boolean");

                                }
                                else {
                                        throw new Exception("Incorrect Type. Boolean expected");

                                }
```

```
                    }
        |^(GRTR c=expression d=expression) {
        if ( c.getType().equals("int") && d.getType().equals("int")){
                                r = new GreaterNode(c.value(), d.value());
                                }
                                else  if (d.getType().equals("function")&&
d.getReturnType().equals("int")){
                                        r = new FunctionNode( c.value() + " > " + d.getCode(),
"boolean");

                                }
                                else  if (c.getType().equals("function")&&
c.getReturnType().equals("int")){
                                        r = new FunctionNode(c.getCode()  + " > " +  d.value(),
"boolean");

                                }
                                else {
                                        throw new Exception("Incorrect Type. Int expected");

                                }



        }
        | ^(LESS c=expression d=expression) {
                if ( c.getType().equals("int") && d.getType().equals("int")){
                                        r = new LessNode(c.value(), d.value());
                                        }
                                        else  if (d.getType().equals("function")&&
d.getReturnType().equals("int")){
                                        r = new FunctionNode( c.value() + " < " + d.getCode(),
"boolean");

                                }
                                else  if (c.getType().equals("function")&&
c.getReturnType().equals("int")){
                                        r = new FunctionNode( c.getCode() + " < " +  d.value() ,
"boolean");

                                }
                                        else {
                                                throw new Exception("Incorrect Type. Int
expected");
```

```
                                        }

          }
          | ^(GRTR_EQL c=expression d=expression){
          if ( c.getType().equals("int") && d.getType().equals("int")){
                              r = new GreaterEqlNode(c.value(), d.value());
                              }
                              else  if (d.getType().equals("function")&&
d.getReturnType().equals("int")){
                                        r = new FunctionNode( c.value() + " >= " + d.getCode(),
"boolean");

                              }
                              else  if (c.getType().equals("function")&&
c.getReturnType().equals("int")){
                                        r = new FunctionNode( c.getCode() + " >= " + d.value(),
"boolean");

                              }
                              else {
                                        throw new Exception("Incorrect Type. Int expected");

                              }
          }

          | ^(LESS_EQL c=expression d=expression)  {
                    if ( c.getType().equals("int") && d.getType().equals("int")){
                              r = new LessEqlNode(c.value(), d.value());
                              }
                              else  if (d.getType().equals("function")&&
d.getReturnType().equals("int")){
                                        r = new FunctionNode( c.value() + " <= " + d.getCode(),
"boolean");

                              }
                              else  if (c.getType().equals("function")&&
c.getReturnType().equals("int")){
                                        r = new FunctionNode( c.getCode() + " <= " +  d.value(),
"boolean");

                              }
                              else {
                                        throw new Exception("Incorrect Type. Int expected");
                                        }
          }
          | ^(DBL_EQLS c=expression d=expression){
                    if ( c.getType().equals("int") && d.getType().equals("int")){
                                        r = new EqualNode(c.value(), d.value());
                                        }
                                        else  if (d.getType().equals("function")&&
d.getReturnType().equals("int")){
                                                  r = new FunctionNode( c.value() + " == " + d.getCode(),
"boolean");

                                        }
                                        else  if (c.getType().equals("function")&&
c.getReturnType().equals("int")){
                                                  r = new FunctionNode(  c.getCode()  + " == " + d.value(),
"boolean");

                                        }
                                        else {
                                                  throw new Exception("Incorrect Type. Int expected");

                                        }
          }
          | ^(NOT_EQLS c=expression d=expression){
                    if ( c.getType().equals("int") && d.getType().equals("int")){
                                        r = new NotEqualNode(c.value(), d.value());
                                        }
                                        else  if (d.getType().equals("function")&&
d.getReturnType().equals("int")){
                                                  r = new FunctionNode( c.value() + " != " + d.getCode(),
"boolean");

                                        }
                                        else  if (c.getType().equals("function")&&
c.getReturnType().equals("int")){
                                                  r = new FunctionNode( c.getCode()  + " != " + d.value() ,
"boolean");

                                        }
                                        else {
                                                  throw new Exception("Incorrect Type. Boolean expected");
```

```
                                }

        }
        |^(AND c=expression d=expression)  {
                if ( c.getType().equals("boolean") && d.getType().equals("boolean")){
                                r = new AndNode(c.value(), d.value());
                                }
                                else  if (d.getType().equals("function")&&
d.getReturnType().equals("boolean")){
                                        r = new FunctionNode( c.value() + " && " + d.getCode(),
"boolean");

                                }
                                else  if (c.getType().equals("function")&&
c.getReturnType().equals("boolean")){
                                        r = new FunctionNode( c.getCode() + " && " +  d.value(),
"boolean");

                                }
                                else {
                                        throw new Exception("Incorrect Type. Bool expected");


                                }

        }
        | ^(OR c=expression d=expression) {
                if ( c.getType().equals( "boolean") && d.getType().equals("boolean")){
                                r = new OrNode(c.value(), d.value());
                                }
                                        else  if (d.getType().equals("function")&&
d.getReturnType().equals("boolean")){
                                        r = new FunctionNode( c.value() + " || " + d.getCode(),
"boolean");

                                }
                                else  if (c.getType().equals("function")&&
c.getReturnType().equals("boolean")){
                                        r = new FunctionNode( c.getCode() + " || " +  d.value(),
"boolean");

                                }
                                else {
                                        throw new Exception("Incorrect Type. Bool expected");
```

```
                        }


        }
        | t=term { r = t;}

        ;
        catch[Exception error]{
                System.out.println(error.getMessage());
                        System.exit(1);
                }


term    returns [ExpressionNode r]
                        : f=function_call {r = f;}
    | d=TRUE {r = new ValueNode(Boolean.valueOf(true), "boolean");}
    | d=FALSE {r = new ValueNode(Boolean.valueOf(false), "boolean");}
                        | ID {


                        //variables.put("x",new Identifier("x", 1, "int"));
                        if (variables.containsKey($ID.text) &&
!variables.get($ID.text).type.equals("function")) r = new FunctionNode($ID.text,
variables.get($ID.text).type);
    else if ( variables.containsKey($ID.text) && variables.get($ID.text).type.equals("function"))
r = new FunctionNode($ID.text, ((ExpressionNode)
variables.get($ID.text).value).getReturnType());   //(ExpressionNode)
variables.get($ID.text).value;
                        else {throw new Exception("Identifier Not Found");
                                                                        }
                        //throws new Exception("Identifier" + $ID.text + "Not Found"); }

                        }
                        |       '(' e=expression ')' {r =e;}
                        |       d=CONST {r = new ValueNode(Integer.parseInt($d.text), "int");}

        //              |        {r = new ValueNode(Integer.parseInt($d.text), "int");}

                        ;
        catch[Exception error]{
                System.out.println(error.getMessage());
                        System.exit(1);
```

```
                    }



//assgn_expression returns [String r]
//          : ^(ASSIGN ID c=expression){
//          if(variables.containsKey($ID.text) &&
((ExpressionNode)variables.get($ID.text).value).getReturnType().equals(c.getReturnType())){
//                    System.out.println("HOLYGUCKOB");
//                    variables.remove($ID.text);
//                    variables.put($ID.text, new Identifier($ID.text, c, c.getReturnType()));
//                    r = $ID.text + " = " + c.getCode() + ";";
//                    System.out.println("Checking assign:" + r);
//                              }
//          else {
//                    System.out.println("This Variable Has not been declared");
//
//}
//}
//                ;

assgn_expression returns [String r]
          : ^(ASSIGN ID c=expression){
          if(variables.containsKey($ID.text) &&
((ExpressionNode)variables.get($ID.text).value).getReturnType().equals(c.getReturnType())){
                    variables.remove($ID.text);
                    variables.put ($ID.text, new Identifier($ID.text, c, c.getReturnType()));
                    r = $ID.text + " = " + c.getCode() + ";";
                              }
          else {
//                    System.out.println(c.getReturnType() + '\n' +
((ExpressionNode)variables.get($ID.text).value).getReturnType());
//                    System.out.println("This Variable Has not been declared");
                    throw new Exception("This Variable Has not been declared");

}
}
                    ;
                    catch[Exception error]{
                    System.out.println(error.getMessage());
                              System.exit(1);
                    }
```

```
main_declaration returns [String r]
: MAIN list=main_declaration_list
  {

    r = "\tpublic static void main(String args[]) { \n";
    r += "\n";
    for (String str : list) {
      //if  (variables.containsKey(str))
      //{
         r += "\t" + "\t" + "\t" + str + "Draw().drawBoard();";
      //}
    }
    r += "\n";
    r += "\t}\n";

    //myBoard = (ArrayList<String>) list.clone();
  }

;

main_declaration_list returns [ArrayList<String> myDrawList]
: DRAW b=ID m = main_declaration_list
              {
              if (m != null) {
                myDrawList = m;
                myDrawList.add($b.text);

                //System.out.println("This board was recognized " + $b.text);
              }
              else {
                myDrawList = new ArrayList<String>();
                myDrawList.add($b.text);
              }
            }


| /* empty*/
;

board_declaration returns [String boardDec]
scope {String id;}
: BOARD ID {$board_declaration::id=$ID.text;}
```

```
    bd=board_declaration_list
    {

      //Creates an String array list from boardDeclaratoinList
      //That should return three strings : size, start, and end respectively.
      ArrayList<String> list1 = bd.getSizeStartEndList();
     // System.out.println("----->" + boardDec);
      boardDec = "\npublic static Board " + $ID.text + "Draw() { \n\n";
      boardDec = boardDec + "\t\tBoard " + $ID.text + " = new Board(" + list1.get(0) + ");\n";

      boardDec = boardDec + "\t\t" + $ID.text + ".addStart(" + list1.get(1) + ");\n";

      boardDec = boardDec + "\t\t" + $ID.text + ".addEnd(" + list1.get(2) + ");\n\n";



      //
      boardDec = "" + "\t" + boardDec + bd.getPointsAndPaths() + "\n";

//      for (String id: bd.getPathIDs()){
//        boardDec = boardDec + "\t\t" + $ID.text + ".addPath(" + id + ");\n";
//      }

      boardDec = boardDec + "\t\treturn " + $ID.text + ";\n" ;

      boardDec = boardDec + "\t}" ;

//      boardDec = boardDec + "\n\t\t" + $ID.text + ".drawBoard();\n" ;
//
//      boardDec = boardDec + '\n' + "\t}\n}";
      variables2.put($ID.text, boardDec);
    }
  ;

//messy starting here

board_declaration_list returns [BoardDeclarationListNode myBDLNode]
scope {String id;}
@init{ArrayList<String> sizeStartEndList = new ArrayList<String>();}
    :  p1=size_declaration p2=start_declaration p3=end_declaration p4=board_statement_list2
    {
              sizeStartEndList.add(p1);
              sizeStartEndList.add(p2);
              sizeStartEndList.add(p3);
     myBDLNode = new BoardDeclarationListNode(sizeStartEndList, p4);

    }

  ;

board_statement_list2 returns[String myBSL2]
                  :  b2=board_statement_2 b2list=board_statement_list2
                    {
                      if (b2list==null){
                          myBSL2 = "\t\t" + (String)b2.getValue();
                      }
                      else{ //list is not null
                          myBSL2 = "\t\t" + (String)b2.getValue() + b2list;
                      }


                    }
                  | /*empty*/ {myBSL2 = "";}
                  ;

board_statement_2  returns [Identifier pathOrPoint] //Identifier //String
              :
              s=set_structure{pathOrPoint = new Identifier("", s, "structure");}
              | point=point_declaration {pathOrPoint = new Identifier("", point, "point");/*pathOrPoint
= point;*/}
              | path=path_declaration {pathOrPoint = new Identifier(path.getId(), path.getName() +
"\t\t" + $board_declaration::id + ".addPath(" + path.getId() + ");\n\n", "path");}
              | var=var_declaration {pathOrPoint = new Identifier("", var, "var");}
        //      | a = assgn_expression {pathOrPoint = new Identifier("", a , "var");}
              | bs=board_statement {pathOrPoint = new Identifier("", bs, "board_statement"); }
         //     | f = function_call {pathOrPoint = new Identifier("",f.getCode(), "function_call");}
              ;

//need to fix  set structure because it needs to return a list of paths from iteration and while
set_structure returns [String myStructure]
          : SET ID {
                  myStructure = "ArrayList<StructureNode> " + $ID.text + "ArrayList = new
ArrayList<StructureNode>();\n";
                  myStructure += $ID.text + "ArrayList = " + $ID.text + "Structure(); \n";
                  myStructure += "for (StructureNode Geforce : " + $ID.text + "ArrayList) { \n" ;
                  myStructure += "Point ati = Geforce.getPoint(); \n";
                  myStructure += $board_declaration::id + ".addPath(Geforce.getPath()); \n } \n";
```

```
                    }
          ;

//need to fix body to return the correct String
structure_declaration returns[String s]
  : STRUCTURE ID  body=structure_body
                    {
                      if (body == null){
                        variables2.put($ID.text, null);
                        s = "";
                      }
                      else{
                        variables2.put($ID.text, body);
                        s = "\n public static ArrayList<StructureNode> " + $ID.text + "Structure() { \n\n";
                        s += "\tArrayList<StructureNode> StructList = new
ArrayList<StructureNode>();\n\n";
                        s += body.toString() + "\n";
                        s += "\treturn StructList;\n";
                        s += "}\n";
                      }
                    }
                    ;

//need to correct structure to return the right string
//Revise Based on fixed grammar
structure_body returns [String myBSL2]
                : b2=structure_statement b2list=structure_body
                  {

                    if (b2list==null){
                                ArrayList<String> paths = new ArrayList<String>();
                                if (b2.getType().equals("path")){
                                  paths.add(b2.getName());
//                                  myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue(), paths);
                                  myBSL2 = "\t\t" + (String)b2.getValue();
                                }
                                else if(b2.getType().equals("point")){
//                                  myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue(), paths);
                                  myBSL2 = "\t\t" + (String)b2.getValue();
                                }
                                else if(b2.getType().equals("var")){
//                                  myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue(), paths);
                                  myBSL2 = "\t\t" + (String)b2.getValue();
                                }
```

```
                          else if(b2.getType().equals("board_statement")){
//                            myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue(), paths);
                            myBSL2 = "\t\t" + (String)b2.getValue();
                          }
                          else if(b2.getType().equals("assgn_expression")){
                            myBSL2 = "\t\t" + (String)b2.getValue();
                          }
                          else{
                            throw new Exception("Type is not path or point. Error.");


                          }
                      }
                      else{ //list is not null
                          if (b2.getType().equals("path")){
//                              ArrayList<String> paths = b2list.getPathIDs();
//                              paths.add(b2.getName());
//                              myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue() +
b2list.getString(), paths);
                            myBSL2 = "\t\t" + (String)b2.getValue() + b2list;
                          }
                          else if(b2.getType().equals("point")){
//                              myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue() +
b2list.getString(), b2list.getPathIDs());
                            myBSL2 = "\t\t" + (String)b2.getValue() + b2list;

                          }
                          else if(b2.getType().equals("var")){
//                              myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue() +
b2list.getString(), b2list.getPathIDs());
                            myBSL2 = "\t\t" + (String)b2.getValue() + b2list;
                          }
                          else if(b2.getType().equals("board_statement")){
//                              myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue() +
b2list.getString(), b2list.getPathIDs());
                            myBSL2 = "\t\t" + (String)b2.getValue() + b2list;
                          }
                          else if(b2.getType().equals("assgn_expression")){
                            myBSL2 = "\t\t" + (String)b2.getValue() + b2list;
                          }
                          else{
                            throw new Exception("Type is not path or point. Error.");
```

```
                    }

                  }

                 }
              | /*empty*/ {myBSL2 ="";}
              ;
        catch[Exception error]{
               System.out.println(error.getMessage());
                       System.exit(1);
               }


structure_statement returns [Identifier ident]:
         bs=structure_board_statement {ident = new Identifier("", bs, "board_statement"); }
        | point = point_declaration {ident = new Identifier("",point,"point");}
        | path = path_declaration {ident = new Identifier(path.getId(),path.getName() + "\n\t" +
                           "if (!StructList.contains(new StructureNode(" +
                           path.getPoint_ID() + ", " + path.getId() + "))) {" +
                           "StructureNode " + path.getPoint_ID() + "Node = new StructureNode("
+
                           path.getPoint_ID() + ", " + path.getId() + "); \n\t" +
                           "StructList.add(" + path.getPoint_ID() + "Node);\n } \n " ,"path");}
        | v = var_declaration {ident = new Identifier("",v,"var");}
        | a = assgn_expression {ident = new Identifier("",a,"assgn_expression");}
        ;

structure_path_declaration returns [PathNode myPath] //Path //String
: PATH i1=ID i2=term d=direction e=expression
             {
               if (i2.getType().equals("function") &&
variables.containsKey(((FunctionNode)i2).Identifier) && e.getReturnType().equals("int")){

               String myPathString = "Path " + $i1.text + " = new Path(" +
((FunctionNode)i2).Identifier + ", \"" + d + "\", " + e.getCode() + ");\n";

               myPath = new PathNode(myPathString, $i1.text, ((FunctionNode)i2).Identifier);
              }
              else if (i2.getType().equals("function") &&
((FunctionNode)i2).getReturnType().equals("point")&& e.getReturnType().equals("int")){
               String myPathString = "Path " + $i1.text + " = new Path(" + i2.getCode() + ", \"" +
d + "\", " + e.getCode() + ");\n";
               myPath = new PathNode(myPathString, $i1.text, i2.getCode());
```

```
                  }
                  else {throw new Exception("Path Expects a point");

                  }
                  }
                | PATH ID
                ;
          catch[Exception error]{
                   System.out.println( error.getMessage());
                            System.exit(1);
                   }


structure_iteration_stmt returns [String str] : ^(WHILE e=expression b=structure_body)
  {
      if(e.getReturnType().equals("boolean") || e.getType().equals("boolean")) {

        str = "while(" + e.getCode() + "){" + b + "}";
//         throw new Exception("````````````````" + b);
      }

      else{
        throw new Exception("Invalid type in while loop. Expected a boolean but received a(n) " +
e.getReturnType());

        }
  }
       ;
          catch[Exception error]{
                   System.out.println( error.getMessage());
                                                System.exit(1);

                   }


structure_selection_stmt returns [String str]
                 : ^(IF e = expression b=structure_body b2=structure_selection_else_stmt)
                  {
                   if(e.getReturnType().equals("boolean") || e.getType().equals("boolean")) {

                   str = "if(" + e.getCode() + "){\n" + b + "} " + b2;}
                   else throw new Exception("Invalid type in if statement. Expected a boolean but
received a(n) " + e.getReturnType()) ;}
                  ;
```

```
                    catch[Exception error]{
            System.out.println( error.getMessage());
                    System.exit(1);
            }



structure_selection_else_stmt  returns [String str]
                    : ^(ELSE b=structure_body) { str= "else{" + b + "}";}
                    |/*empty*/ {str = "";}
                    ;

structure_board_statement returns [String str]:
            i = structure_iteration_stmt {str = i;}
          | b = structure_selection_stmt {str = b;}
          | p = print_stmt {str = p;}
          | f = function_call {str = f.getCode();}
          ;

size_declaration returns [String myPoint] //Point
: SIZE c1=CONST c2=CONST
{/*System.out.println("Size is " + $c1.text + ", " + $c2.text); */
 myPoint = "new Point(" + $c1.text + ", " + $c2.text + ")";
 }
 ;

//board_statement_list : print_stmt
//;

board_statement returns [String str]:
            i = board_iteration_stmt {str = i;}
          | b = board_selection_stmt {str = b;}
          | p = print_stmt {str = p;}
          | a = assgn_expression {str = a;}
           | f = function_call {str = f.getCode() + ";";}
          ;

//statement_list returns [Identifier ident]:
//          (s = statement)* {ident = s;}
//          ;
//something here is probably wrong check again
statement returns [Identifier ident]:
         i = iteration_stmt {ident = new Identifier("",i,"iteration_stmt");}
        | s = selection_stmt {ident = new Identifier("",s,"selection_stmt");}
```

```
         | point = point_declaration {ident = new Identifier("",point,"point");}
         | path = path_declaration {ident = new Identifier(path.getId(), path.getName() ,"path");}
         | v = var_declaration {ident = new Identifier("",v,"var");}
         | a = assgn_expression {ident = new Identifier("",a,"assgn_expression");}
        ;

statement_list returns[String myBSL2] //BoardStatementList2Node //String //BSL2Node
@init{/*ArrayList<Point> points = new ArrayList<Point>(); ArrayList<String> paths = new
ArrayList<String>();*/}
                    :  b2=statement b2list=statement_list
                       {

                       if (b2list==null){
                            ArrayList<String> paths = new ArrayList<String>();
                            if (b2.getType().equals("path")){
                              paths.add(b2.getName());
//                              myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue(), paths);
                              myBSL2 = "\t\t" + (String)b2.getValue();
                            }
                            else if(b2.getType().equals("point")){
//                              myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue(), paths);
                              myBSL2 = "\t\t" + (String)b2.getValue();
                            }
                            else if(b2.getType().equals("var")){
//                              myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue(), paths);
                              myBSL2 = "\t\t" + (String)b2.getValue();
                            }
                            else if(b2.getType().equals("board_statement")){
//                              myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue(), paths);
                              myBSL2 = "\t\t" + (String)b2.getValue();
                            }
                            else if(b2.getType().equals("assgn_expression")){
                                myBSL2 = "\t\t" + (String)b2.getValue();
                            }
                            else{
                              throw new Exception("Type is not path or point. Error.");


                            }
                        }
                        else{ //list is not null
                            if (b2.getType().equals("path")){
//                                ArrayList<String> paths = b2list.getPathIDs();
//                                paths.add(b2.getName());
//                          myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue() +
b2list.getString(), paths);
                          myBSL2 = "\t\t" + (String)b2.getValue() + b2list;
                       }
                       else if(b2.getType().equals("point")){
//                          myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue() +
b2list.getString(), b2list.getPathIDs());
                          myBSL2 = "\t\t" + (String)b2.getValue() + b2list;

                       }
                       else if(b2.getType().equals("var")){
//                          myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue() +
b2list.getString(), b2list.getPathIDs());
                          myBSL2 = "\t\t" + (String)b2.getValue() + b2list;
                       }
                       else if(b2.getType().equals("board_statement")){
//                          myBSL2 = new BSL2Node("\t\t" + (String)b2.getValue() +
b2list.getString(), b2list.getPathIDs());
                          myBSL2 = "\t\t" + (String)b2.getValue() + b2list;
                       }
                       else if(b2.getType().equals("assgn_expression")){
                          myBSL2 = "\t\t" + (String)b2.getValue() + b2list;
                       }
                       else{
                         throw new Exception("Type is not path or point. Error.");


                       }

                   }

                  }
                | /*empty*/ {myBSL2 ="";}
                ;
                 catch[Exception error]{
                 System.out.println( error.getMessage());
                         System.exit(1);
                 }


board_iteration_stmt returns [String str] : ^(WHILE e=expression b=board_statement_list2)
 {
     if(e.getReturnType().equals("boolean") || e.getType().equals("boolean")) {
         str = "while(" + e.getCode() + "){" + b + "}";
//         throw new Exception("````````````````" + b);
       }

       else{
         throw new Exception("Invalid type in while loop. Expected a boolean but received a(n) " +
e.getReturnType());

         }
  }
        ;
                    catch[Exception error]{
                    System.out.println( error.getMessage());
                               System.exit(1);
                    }


board_selection_stmt returns [String str]
                  :
                  ^(IF e=expression b=board_statement_list2 b2=board_selection_else_stmt )
                   {
                   if(e.getReturnType().equals("boolean")|| e.getType().equals("boolean")) {
                   str = "if(" + e.getCode() + "){" + b + "} " + b2 ;
                   }
                   else{
         throw new Exception("Invalid type in conditional statement. Expected a boolean but
received a(n) " + e.getReturnType());

         }
                   }
                   ;
                    catch[Exception error]{
                    System.out.println(error.getMessage());
                              System.exit(1);
                    }



board_selection_else_stmt  returns [String str]:
                      ^(ELSE b=board_statement_list2)
                      {
                        str= "else{" + b + "}";
                      }
                      |/*empty*/ {str = "";}
```

```
                    ;


iteration_stmt returns [String str] : ^(WHILE e=expression b=statement_list)
 {
      if(e.getReturnType().equals("boolean") || e.getType().equals("boolean")) {

        str = "while(" + e.getCode() + "){" + b + "}";
//         System.out.println("````````````````" + b);
      }

      else{
        throw new Exception("Invalid type in while loop. Expected a boolean but received a(n) " +
e.getReturnType());

       }
 }
       ;
          catch[Exception error]{
                   System.out.println(error.getMessage());
                            System.exit(1);
                   }



selection_stmt returns [String str]
                 : ^(IF e = expression b=statement_list b2=selection_else_stmt)
                  {
                   if(e.getReturnType().equals("boolean") || e.getType().equals("boolean")) {

                  str = "if(" + e.getCode() + "){\n" + b + "} " + b2 ;}}
                  ;


selection_else_stmt  returns [String str]
                     : ^(ELSE b=statement_list) { str= "else{" + b + "}";}
                     |/*empty*/ {str = "";}
                     ;



start_declaration returns [String myPoint] //Point
   : START c3=CONST c4=CONST
   {
```

```
      myPoint = "new Point(" + $c3.text + ", " + $c4.text + ")";
    }
      ;
end_declaration returns [String myPoint]
  : END c5=CONST c6=CONST
  {
  myPoint = "new Point(" + $c5.text + ", " + $c6.text + ")";
  }
      ;

 function_declaration returns [String r]

                                        : FUNC INT ID p=parameters bs2=board_statement_list2
j=jump_stmt {


                                        r =  "    public static int " + $ID.text +  p + "{" + '\n' +
                                                                bs2 + '\n' +
                                                                        "\t\t"+ j + '\n'+

                                                                        "}";
                                        variables.put($ID.text, new Identifier($ID.text, new
FunctionNode("", "int", $ID.text), "function"));
                                        function_list.add(r);

                                        }
        | FUNC BOOLEAN ID p=parameters bs2=board_statement_list2 j=jump_stmt {
        r =  "       public static boolean " + $ID.text  + p +"{" + '\n' +
                                                                        "          "+ bs2 + '\n' +
                                                                        "                    " + j + '\n' +
                                                                        "}";
                                         variables.put($ID.text, new Identifier($ID.text, new
FunctionNode("", "boolean", $ID.text), "function"));
                                         function_list.add(r);

                                        }
        | FUNC POINT ID p=parameters bs2=board_statement_list2 j=jump_stmt {
        r =  "       public static Point " + $ID.text + p + "{" + '\n' +
                                                                        "          "+ bs2 + '\n' +
                                                                        "                    " + j + '\n'+
                                                                        "}";
       variables.put($ID.text, new Identifier($ID.text, new FunctionNode("", "point",$ID.text),
"function"));
                                        function_list.add(r);
        }
        | FUNC ID p=parameters bs2=board_statement_list2 {

        r =  "       public static void " + $ID.text + p + "{" + '\n' +

                                                                    "          "+ bs2 + '\n' +
                                                                    "}";
                                    variables.put($ID.text, new Identifier($ID.text, new
FunctionNode("", "void", $ID.text), "function"));
                                    function_list.add(r);
                                    }
 ;

parameters returns [String str]
@init{ ArrayList<String> vars = new ArrayList<String>();} : ( b=var_declaration_unassigned
{vars.add(b);} ( c=var_declaration_unassigned {vars.add(c);})*)
{
   str = "(";
   if(vars.size()!=0)
   {
     for(int x = 0; x < vars.size(); x++)
     {
        if(x==0)
        {
            str+=vars.get(x);
        }
        else
        {
            str+= "," + vars.get(x);
        }
     }
   }
   str += ")";
   vars = new ArrayList<String>(); }
   | /*empty */ {str = "()";}
;


 var_declaration_unassigned returns [String code] : INT d=ID { code = "int " + $d.text + "";


                                                  variables.put($d.text, new Identifier($d.text, new
ValueNode( 0, "int"), "int"));}
                        | BOOLEAN e=ID {code = "boolean " + $e.text + "";
                        variables.put($e.text, new Identifier($e.text, new
ValueNode(Boolean.valueOf(false), "boolean"), "boolean"));

                        }


                        ;

//bs_statement_list : /*empty*/
//;

jump_stmt returns [String code] : RETURN  d=expression {
  if (d.getType().equals("function") && variables.containsKey(((FunctionNode)d).Identifier)){
      code = "return " + ((FunctionNode)d).Identifier + ";";}
  else{
  code = "return " + d.getCode() + ";";}}
    ;

//IF THERE ARE NO QUOTES, ERROR IS NOT HANDLED
print_stmt returns [String str]
         : PRINT STRING {
         if($STRING.text == null){
           str = "System.out.println();";
         }
         else{
           str = "System.out.println("+$STRING.text+");";
         }
        }  //System.out.println( $STRING.text );}
    ;
```

ExpressionNode - Pineda

```
//package compiler;

public interface ExpressionNode {


       String toString();

       Object value();
       String getType();
```

```
	String getCode();
	String getReturnType();

}
```

FunctionNode - Pineda

```
//package compiler;

public class FunctionNode implements ExpressionNode {

	public String code;
	public String return_type;
	public String Identifier;
	public FunctionNode(String function_call, String return_type){
		code = function_call;
		this.return_type = return_type;

	}
	public FunctionNode(String function_call, String return_type, String ID){
		code = function_call;
		this.return_type = return_type;
		Identifier = ID;
	}
	@Override
	public Object value() {
		// TODO Auto-generated method stub
		return code;
	}

	@Override
	public String getType() {
		// TODO Auto-generated method stub
		return "function";
	}
	public String getReturnType(){
		return return_type;
	}

	@Override
	public String getCode() {
		// TODO Auto-generated method stub
		return code;
	}

}
```

GreaterEqlNode - Pineda

```
//package compiler;
```

```
public class GreaterEqlNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


public GreaterEqlNode(Object x , Object y){
	boolean rvalue = (Integer)x >= (Integer)y;
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
	return "boolean";
}
@Override
public String getCode() {
	// TODO Auto-generated method stub
	return String.valueOf(code);
}
@Override
public String getReturnType() {
	// TODO Auto-generated method stub
	return "boolean";
}


}
```

GreaterNode - Pineda

```
//package compiler;

public class GreaterNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


public GreaterNode(Object x , Object y){
	boolean rvalue = (Integer)x > (Integer)y;
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
       // TODO Auto-generated method stub

       return value;
}
public String getType(){
       return "boolean";
}

@Override
public String getCode() {
       // TODO Auto-generated method stub
       return String.valueOf(code);
}
@Override
public String getReturnType() {
       // TODO Auto-generated method stub
       return "boolean";
}

}
```

Identifier – Pineda, Bourdett

```
//package compiler;

public class Identifier {

       String name;
       Object value;
       String type;

       public Identifier (String name, Object value, String type){
              this.name = name;
              this.value = value;
              this.type = type;
       }

       public String getName() {
              return name;
       }

       public void setName(String name) {
              this.name = name;
       }

       public Object getValue() {
              return value;
       }

       public void setValue(Object value) {
              this.value = value;
       }
```

```
	public String getType() {
		return type;
	}

	public void setType(String type) {
		this.type = type;
	}
}
```

LessEqlNode - Pineda

```
//package compiler;

public class LessEqlNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


public LessEqlNode(Object x , Object y){
	boolean rvalue = (Integer)x <= (Integer)y;
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
	return "boolean";
}
@Override
public String getCode() {
	// TODO Auto-generated method stub
	return String.valueOf(code);
}
@Override
public String getReturnType() {
	// TODO Auto-generated method stub
	return "boolean";
}

}
```

LessNode - Pineda

```
//package compiler;
```

```
public class LessNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


public LessNode(Object x , Object y){
	boolean rvalue = (Integer)x < (Integer)y;
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
	return "boolean";
}
@Override
public String getCode() {
	// TODO Auto-generated method stub
	return String.valueOf(code);
}
@Override
public String getReturnType() {
	// TODO Auto-generated method stub
	return "boolean";
}


}
```

Matrix – Francoeur, Bourdett

```
//package codeGeneration.fred;

import java.awt.Point;


public class Matrix {

//	private int width;
//	private int length;
	private int start_x;
	private int start_y;
	private int end_x;
	private int end_y;

	private boolean[][] tiles;
	public Matrix(int width, int length){
//		this.width = width;
//		this.length = length;
		
		tiles = new boolean[width][length];
		//initialize with falses
		for (int i=0; i<width; i++){
			for (int j=0; j<length; j++){
				tiles[i][j] = false;
			}
		}
	}
	
	public void insertPath(Point startPoint, String dir, int pathLength){
		int startX = (int) startPoint.getX();
		int startY = (int) startPoint.getY();
		
		//TODO check whether startPoint is valid
		
		int endX = startX;
		int endY = startY;
		
		if(dir.equals("right")){
			endX = startX + pathLength;
			for (int i=startX; i<=endX; i++){
				tiles[startY][i] = true;
			}
		}
		else if (dir.equals("down")){
			endY = startY + pathLength;
			for (int i=startY; i<=endY; i++){
				tiles[i][startX] = true;
			}
		}
		else if(dir.equals("up")){
			endY = startY - pathLength;
			for (int i=startY; i>=endY; i--){
				tiles[i][startX] = true;
			}
		}
		else if(dir.equals("left")){
			endX = startX - pathLength;
			for (int i=startX; i>=endX; i--){
				tiles[startY][i] = true;
			}
		}
		else {
			//TODO deal with incorrect direction
		}
		
		//TODO check whether path is valid by checking endpoint
	}
	
	public boolean isActive(int x, int y){
		return tiles[x][y];
	}

	public void setStart(int x, int y){
		this.start_x = x;
		this.start_y = y;
	}

	public void setEnd(int x, int y){
		this.end_x = x;
		this.end_y = y;
	}
	public boolean isStart(int x, int y){
		if(this.start_x == y && this.start_y == x){
			return true;
		}
		else{
			return false;
		}
	}

	public boolean isEnd(int x, int y){

		if(this.end_x == y && this.end_y == x){
			return true;
		}
		else{
			return false;
		}
	}
}
```

MultiNode - Pineda

```
//package compiler;

public class MultNode implements ExpressionNode {
		ExpressionNode node1;
		ExpressionNode node2;
		Object value;
		Object code;


	public MultNode(Object x , Object y){
		int rvalue = (Integer)x * (Integer)y;
		value = rvalue;
		code = rvalue;
	}
	@Override
	public Object value() {
		// TODO Auto-generated method stub

		return value;
	}
	public String getType(){
		return "int";
	}
	@Override
	public String getCode() {
		// TODO Auto-generated method stub
		return String.valueOf(code);
	}
	@Override
	public String getReturnType() {
		// TODO Auto-generated method stub
		return "int";
	}


}
```

NegationNode - Pineda

```
//package compiler;

public class NegationNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


public NegationNode(Object x ){
	int rvalue = - (Integer)x;
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
	return "int";
}
@Override
public String getCode() {
	// TODO Auto-generated method stub
	return String.valueOf(code);
}
@Override
public String getReturnType() {
	// TODO Auto-generated method stub
	return "int";
}
}
```

NotEqualNode - Pineda

```
//package compiler;

public class NotEqualNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


public NotEqualNode(Object x , Object y){
	boolean rvalue = (Integer)x != (Integer)y;
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
	return "boolean";
}
@Override
public String getCode() {
	// TODO Auto-generated method stub
	return String.valueOf(code);
}
@Override
public String getReturnType() {
	// TODO Auto-generated method stub
	return "boolean";
}


}
```

NotNode - Pineda

```
//package compiler;

public class NotNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;
public NotNode(Object x){
	boolean rvalue = !(Boolean) x;
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
	return "boolean";
}
@Override
public String getCode() {
	// TODO Auto-generated method stub
	return String.valueOf(code);
}
@Override
public String getReturnType() {
	// TODO Auto-generated method stub
	return "boolean";
}


}
```

OrNode - Pineda

```
//package compiler;

public class OrNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


public OrNode(Object x , Object y){
	boolean rvalue = Boolean.parseBoolean(String.valueOf(x)) ||
Boolean.parseBoolean(String.valueOf(y));
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
	return "boolean";
}
@Override
public String getCode() {
	// TODO Auto-generated method stub
	return String.valueOf(code);
}
@Override
public String getReturnType() {
	// TODO Auto-generated method stub
	return "boolean";
}

}
```

Path - Francoeur

```
//package codeGeneration.fred;

import java.awt.Point;

public class Path {
	public Point point;
	public String direction;
	public int length;

	public Path(Point point, String direction, int length){
		this.point = point;
		this.direction = direction;
		this.length = length;
	}

	public Point getPoint() {
		return point;
	}

	public void setPoint(Point point) {
		this.point = point;
	}

	public String getDirection() {
		return direction;
	}

	public void setDirection(String direction) {
		this.direction = direction;
	}

	public int getLength() {
		return length;
	}

	public void setLength(int length) {
		this.length = length;
	}

}
```

PathNode - Bourdett

```
//package compiler;

//import codeGeneration.fred.Path;

public class PathNode {

	String string;
	String id;
	String point_id;
	String generation;

	public PathNode(String name, String id, String point_id){
		this.string = name;
		this.id = id;
		this.point_id = point_id;

	}

	public String getName() {
		return string;
	}

	public void setName(String name) {
		this.string = name;
	}

	public String getId() {
		return id;
	}

	public String getPoint_ID()
	{
		return point_id;
	}

	public String getPathDrawBoard(String board){
		generation = board + "." + "addPath(" + id + ");";
		return generation;


	}
	public String getPathDrawStructure(String structure){
		generation = structure + "." + "setPath(" + id + ");";
		return generation;


	}
	public void setId(String id) {
		this.id = id;
	}

}
```

StatementListNode – Mercado, Bourdett

```
//package compiler;

import java.util.ArrayList;

//import codeGeneration.fred.Path;

public class StatementListNode {

	String string;
	ArrayList<String> pathIDs;

	public StatementListNode(String string, ArrayList<String> pathIDs){
		this.string = string;
		this.pathIDs = pathIDs;
	}

	public String getString() {
		return string;
	}

	public void setString(String string) {
		this.string = string;
	}

	public ArrayList<String> getPathIDs() {
		return pathIDs;
	}

	public void setPathIDs(ArrayList<String> pathIDs) {
		this.pathIDs = pathIDs;
	}
}
```

Structure - Francoeur

```
//package codeGeneration.fred;

import java.util.ArrayList;

public class Structure {
	private ArrayList<Path> paths = new ArrayList<Path>();
//	private String ID;

	public Structure(){

	}
	public void setPath(Path path){
		paths.add(path);
	}

	public ArrayList<Path> getPaths(){
		return paths;
	}

//	public String getString(){
//		String ret = "";
//		for (Path p: paths){
////			ret = "" + p.getName();
//		}
//		return ret;
//	}


}
```

StructureNode - Mercado

```
//package compiler;
//import codeGeneration.fred.Path;
import java.awt.Point;


public class StructureNode {

	private Point p;
	private Path q;

	public StructureNode (Point p , Path q) {
		this.p = p;
		this.q = q;
	}

	public Point getPoint()
	{
		return p;
	}

	public Path getPath()
	{
		return q;
	}


}
```

SubNode - Pineda

```
//package compiler;
```

```
public class SubNode implements ExpressionNode {
	ExpressionNode node1;
	ExpressionNode node2;
	Object value;
	Object code;


public SubNode(Object x , Object y){
	int rvalue = (Integer)x - (Integer)y;
	value = rvalue;
	code = rvalue;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
	return "int";
}
@Override
public String getCode() {
	// TODO Auto-generated method stub
	return String.valueOf(code);
}
@Override
public String getReturnType() {
	// TODO Auto-generated method stub
	return "int";
}


}
```

ValueNode - Pineda

```
//package compiler;

public class ValueNode implements ExpressionNode {
	ExpressionNode node1;
	Object value;
	String type;

public ValueNode(Object x ,String y){
	value = x;
	type= y;
}
@Override
public Object value() {
	// TODO Auto-generated method stub

	return value;
}
public String getType(){
       return type;
}
@Override
public String getCode() {
       // TODO Auto-generated method stub
       return String.valueOf(value);
}
@Override
public String getReturnType() {
       // TODO Auto-generated method stub
       return type;
}

}
```

Test-Francoeur,Bourdett

```
//package compiler;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.InputStreamReader;
//import java.util.ArrayList;

import org.antlr.runtime.*;
import org.antlr.runtime.tree.CommonTreeNodeStream;

//import compiler.AmazeParser.maze_return;

public class Test {

       /**+ '\n' + "}";
        * @param args
        */
       public static void main(String[] args)throws Exception {
              FileInputStream fileInput = new FileInputStream(args[0]);

              String[] check = args[0].split("\\.");
              if(check[check.length-1].equals("amz")){

              DataInputStream dataInput = new DataInputStream(fileInput);
              BufferedReader buffReader = new BufferedReader(new
InputStreamReader(dataInput));
              String line;
              String inLine="";
              while((line = buffReader.readLine())!= null){
                     inLine = inLine + line;
//                   System.out.println(line);
              }
              dataInput.close();
```

```
                CharStream charStream = new ANTLRStringStream(inLine);

                AmazeLexer lexer = new AmazeLexer(charStream);

                TokenStream tokenStream = new CommonTokenStream(lexer);
                AmazeParser parser = new AmazeParser(tokenStream);

                AmazeParser.maze_return maze = parser.maze();
                CommonTreeNodeStream nodeStream = new CommonTreeNodeStream(maze.tree);
                EvaluatorWalker walker = new EvaluatorWalker(nodeStream);
                String temp = walker.maze();

try{
                 FileWriter fstream = new FileWriter("Out.java");
                 BufferedWriter out = new BufferedWriter(fstream);
                 out.write(temp);
                 out.close();
}
                 catch (Exception e){//Catch exception if any
                 System.out.println("Error: " + e.getMessage());
                 }
                }
                else{
                      System.out.println("Not a .amz file!");
                }
        }

}
```

Extra Unintegrated Source Code

TesterPLT – Bourdett

```
package codeGeneration;
//import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Point;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
//import javax.swing.border.LineBorder;

public class TesterPLT {

	public Point[][] matrix;

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		JFrame frame = new JFrame("Maze Game");

		//size in pixels
		int boardWidth = 600;
		int boardLength = 600;

		int numRows = 100;
		int numColumns = 100;

		JPanel myPanel = new JPanel(new GridLayout(numRows,numColumns));

		//removes run time error after changing dimension
		System.setProperty("java.util.Arrays.useLegacyMergeSort", "true");
		Dimension boardDim = new Dimension(boardWidth, boardLength);
		myPanel.setPreferredSize(boardDim);

		ImageIcon tileIcon = new ImageIcon("imgs/tile.jpg");
		JLabel temp;

		ImageIcon blankIcon = new ImageIcon("imgs/blank.jpg");
		JLabel temp2;

		Matrix myMatrix = new Matrix(numRows, numColumns);

		//define paths
		myMatrix.insertPath(new Point(20, 80), "up", 60);
		myMatrix.insertPath(new Point(20, 20), "right", 15);
		myMatrix.insertPath(new Point(30, 20), "down", 30);
		myMatrix.insertPath(new Point(30, 50), "left", 10);
		myMatrix.insertPath(new Point(35, 20), "down", 60);
		myMatrix.insertPath(new Point(35, 80), "right", 30);
		myMatrix.insertPath(new Point(65, 80), "up", 60);
		myMatrix.insertPath(new Point(50, 20), "right", 30);

		//draw tiles based on matrix
		int k = 0;
		for (int i=0; i<numRows; i++){
			for (int j=0; j<numColumns; j++){
				if (myMatrix.isActive(i, j)){
```

```
                                temp = new JLabel(tileIcon);
                                myPanel.add(temp, k++);
                        }
                        else{ //point is blank
                                temp2 = new JLabel(blankIcon);
                                myPanel.add(temp2, k++);
                        }
                }
        }

        frame.add(myPanel);
//code to finish up frame
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.pack();
                frame.setVisible(true);

        }

}
```

TesterFred – Francoeur

```
package codeGeneration.fred;



//import java.awt.BorderLayout;
import java.awt.Point;


public class TestFred {
        /**
         * @param args
         */
        public static void main(String[] args) {
                Point g = new Point(20,10);
                Path o = new Path(g,"left", 10);
                Point s = new Point(20,10);
                Point e = new Point (20,5);
                Path bleach = new Path(g,"down",6);
                Path naruto = new Path(g,"up",8);

//              Function func = new Function();

                Structure bob = new Structure();
                bob.setPath(bleach);
                bob.setPath(naruto);
                Board k = new Board(25,25);
                k.addStart(s.x, s.y);
                k.addEnd(e.x, e.y);
                k.addPath(o);
                k.addStructure(bob);
                k.drawBoard();
                Board jin = new Board(50,50);
                jin.addStart(s.x,s.y);
        jin.addEnd(e.x, e.y);
        jin.addStructure(bob);
        jin.drawBoard();
    }

}
```

PacMan – Mercado

```
package codeGeneration;
import java.awt.*;
import javax.swing.*;

/**
* @author Amaze Group
*
*/
public class Pacman {

    public static void main(String[] args) {

        //Title for the frame
        JFrame frame = new JFrame("Maze Game");

        //Size of a typical Pacman maze
        int numRows = 20;
        int numColumns = 20;

        //Creating gridLayout for the frame
        JPanel myPanel = new JPanel(new GridLayout(numRows,numColumns));

        //Setting the panel background to blue
        myPanel.setBackground(Color.blue);

        //Create a label to add to the panel
        JLabel pacmanTileLabel;
        JLabel pacmanBlankLabel;
        JLabel pacmanWallLabel;

        //Create an image icon from pacman tile image
        ImageIcon pacmanTile = new ImageIcon("imgs/pacman_tile.png");
        ImageIcon pacmanBlank = new ImageIcon("imgs/pacman_blank.png");
        ImageIcon pacmanWallTile = new ImageIcon("imgs/pacman_wall_tile.png");

        PacmanMatrix mazeLayout = new PacmanMatrix(numRows, numColumns);

        mazeLayout.insertPath(new Point(19,19), "left", 20);

        //Instantiate label for each point to be added to the maze
        //Add label to the panel
        for (int i=0; i<numRows; i++) {
            for (int j=0; j<numColumns; j++) {
                if (mazeLayout.getPointStatus(i, j) == 1) {
                    pacmanTileLabel = new JLabel(pacmanTile);
                    myPanel.add(pacmanTileLabel);
                }
```

```
                    else if (mazeLayout.getPointStatus(i, j) == 0) {
                          pacmanWallLabel = new JLabel(pacmanWallTile);
                          myPanel.add(pacmanWallLabel);
                    }
                    else {
                          pacmanBlankLabel = new JLabel(pacmanBlank);
                    }
              }
        }

        //Add panel to the frame
        frame.add(myPanel);

        //Draw the frame and set default closing operation
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);

  }
}
```

PacmanMatrix - Mercado

```
package codeGeneration;
import java.awt.Point;


public class PacmanMatrix {

      private int width;
      private int length;
      private int[][] tiles;

      public PacmanMatrix(int width, int length) {
            this.width = width;
            this.length = length;

            tiles = new int[this.width][this.length];

            //Initializes with zeroes
            for (int i=0; i<width; i++){
                  for (int j=0; j<length; j++){
                        tiles[i][j] = 0;
                  }
            }
      }

      //Public method to insert a blank space in the maze
      public void insertBlank (int x, int y) {
            tiles[x][y] = 2;
      }

      //Public method to insert a point that is traversable in the maze
      public void insertPoint (int x, int y) {
```

```
		tiles[x][y] = 1;
	}
	//Public method to insert a wall in the maze
	public void insertWall (int x, int y){
		tiles[x][y] = 0;
	}

	public void insertPath(Point startPoint, String dir, int pathLength){
		int startX = (int) startPoint.getX();
		int startY = (int) startPoint.getY();

		//TODO check whether startPoint is valid

		int endX = startX;
		int endY = startY;

		if(dir == "right"){
			endX = startX + pathLength;
			for (int i=startX; i<=endX; i++){
				tiles[startY][i] = 1;
			}
		}
		else if (dir == "down"){
			endY = startY + pathLength;
			for (int i=startY; i<=endY; i++){
				tiles[i][startX] = 1;
			}
		}
		else if(dir == "up"){
			endY = startY - pathLength;
			for (int i=startY; i>=endY; i--){
				tiles[i][startX] = 1;
			}
		}
		else if(dir == "left"){
			endX = startX - pathLength;
			for (int i=startX; i>=endX; i--){
				tiles[startY][i] = 1;
			}
		}
		else {
			//TODO deal with incorrect direction
		}

		//TODO check whether path is valid by checking endpoint
	}

	public int getPointStatus(int x, int y){
		return tiles[x][y];
	}

	public int getWidth() {
		return this.width;
	}
	public int getLength() {
		return this.length;
	}

}
```

Tester - Bourdett

```
package codeGeneration;
//import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Point;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
//import javax.swing.border.LineBorder;

public class Tester {

	public Point[][] matrix;

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		JFrame frame = new JFrame("Maze Game");

		//size in pixels
		int boardWidth = 600;
		int boardLength = 600;

		int numRows = 100;
		int numColumns = 100;

		JPanel myPanel = new JPanel(new GridLayout(numRows,numColumns));

		//removes run time error after changing dimension1
		System.setProperty("java.util.Arrays.useLegacyMergeSort", "true");
		Dimension boardDim = new Dimension(boardWidth, boardLength);
		myPanel.setPreferredSize(boardDim);

		ImageIcon tileIcon = new ImageIcon("imgs/tile.jpg");
		JLabel temp;

		ImageIcon blankIcon = new ImageIcon("imgs/blank.jpg");
		JLabel temp2;

		Matrix myMatrix = new Matrix(numRows, numColumns);
		//define paths
		myMatrix.insertPath(new Point(99, 99), "up", 99);
		myMatrix.insertPath(new Point(99, 0), "left", 99);
		myMatrix.insertPath(new Point(78, 56), "down", 10);

		//draw tiles based on matrix
//		int k = 0;
		for (int i=0; i<numRows; i++){
			for (int j=0; j<numColumns; j++){
				if (myMatrix.isActive(i, j)){
					temp = new JLabel(tileIcon);
					//myPanel.add(temp, k++);
					myPanel.add(temp);
				}
				else{ //point is blank
					temp2 = new JLabel(blankIcon);
					//myPanel.add(temp2, k++);
					myPanel.add(temp2);
				}
			}
		}

		frame.add(myPanel);
//code to finish up frame
	frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.pack();
		frame.setVisible(true);

	}

}
```

BooleanFunc - Francoeur

```
package codeGeneration.fred;



import java.util.ArrayList;


public class BooleanFunc {
	private Boolean type;
	private ArrayList params;
	private ArrayList<Path> paths = new ArrayList<Path>();

	public BooleanFunc(ArrayList params){
		this.params = params;
	}

	public void setPath(Path path){
		paths.add(path);
	}
	public Boolean getType(){
		return type;
	}
}
```

IntFunc - Francoeur

```
package codeGeneration.fred;



import java.util.ArrayList;


public class IntFunc {
	private int type;
	private ArrayList params;
	private ArrayList<Path> paths = new ArrayList<Path>();

	public IntFunc(ArrayList params){
		this.params = params;
	}

//	public void setPath(Path path){
//		paths.add(path);
//	}

	public int getType(){
		return type;
	}

}
```

PathFunc - Francoeur

```
package codeGeneration.fred;



import java.awt.Point;
import java.util.ArrayList;


public class PointFunc {
	private Point type;
	private ArrayList params = new ArrayList();
	private ArrayList<Path> paths = new ArrayList<Path>();

	public PointFunc(){

	}
	public void setPath(Path path){
		paths.add(path);
	}

	public Point getType(){
		return type;
	}
}
```

PointFunc - Francoeur

```
package codeGeneration.fred;



import java.awt.Point;
import java.util.ArrayList;


public class PointFunc {
	private Point type;
	private ArrayList params = new ArrayList();
	private ArrayList<Path> paths = new ArrayList<Path>();

	public PointFunc(){

	}

	public void setPath(Path path){
		paths.add(path);
	}

	public Point getType(){
		return type;
	}
}
```

MazeGenerator - Contreras

```
package amazeFiles;
import java.awt.Point;
import java.util.*;
import codeGeneration.fred.*;
import java.io.*;

public class MazeGenerator {
	public static void main (String[] args)
	{
		try{
		  FileWriter fstream = new FileWriter("src/amazeFiles/testtester.amz");
		  BufferedWriter out = new BufferedWriter(fstream);
		  //specify size of maze and number of points to connect
		  out.write(genAmaze(50,50,10));
		  out.close();
    }
```

```
            catch (Exception e)
            {
              System.out.println("Error: " + e.getMessage());
            }
      }
      public static String genAmaze(int x, int y, int amntPoints)
      {
            ArrayList<Point> points = new ArrayList<Point>(amntPoints);
            ArrayList<Path> paths = new ArrayList<Path>();
            for(int r = 0; r < amntPoints; r++)
            {
                  int temp1 = (int)(Math.random()*x);
                  int temp2 = (int) (Math.random()*y);
                  points.add(new Point(temp1,temp2));
            }
            for(int r = 0; r < amntPoints-1; r++)
            {
                  int rand = (int)(Math.random()*(4));
                  Point currentLocation = points.get(r);
                  String direction="";
                  int initDistance=0;

                  if(rand==0)
                  {
                        direction = "up";
                        while(initDistance<0)
                        {
                              initDistance = 
(int)(Math.random()*(currentLocation.getY()))-1;
                        }

                  }
                  else if (rand==1)
                  {
                        direction = "down";
                        initDistance = (int)(Math.random()*(y-
currentLocation.getY()))-1;
                  }
                  else if (rand==2)
                  {
                        direction = "left";
                        initDistance = 
(int)(Math.random()*(currentLocation.getX()))-1;
                  }
                  else if (rand==3)
                  {
                        direction = "right";
                        initDistance = (int)(Math.random()*(x - 
currentLocation.getX()))-1;
                  }
                  paths.add(new Path(currentLocation, direction, initDistance));
                  currentLocation = nextLoc(currentLocation, direction, 
initDistance);
                  for(Path p: directions(currentLocation,points.get(r+1)))
                        paths.add(p);
```

```
		}
		String amzCode =
				"board omg " +
				"\n{" +
					"\n\t size: " + x + ","  + y +"; " +
					"\n\t start: " + (int)points.get(0).getX() + ", " +
(int)points.get(0).getY()+ ";" +
					"\n\t end:" + (int)points.get(points.size()-
1).getX() + ", " + (int)points.get(points.size()-1).getY()+";";
		for(int r = 0; r < paths.size(); r++)
		{
			amzCode += "\n\t\tpoint point" + r + " : " +
(int)((paths.get(r).getPoint()).getX()) + ", " +
(int)(paths.get(r).getPoint().getY()) + ";";
			amzCode += "\n\t\tpath path" + r + " : point" + r + " , " +
paths.get(r).getDirection() + " , " + (paths.get(r).getLength()-1) + ";" ;
		}
		amzCode+= "\n}\nmain" +
				"\n{" +
				"\n\t draw(omg);" +
				"\n}";
		System.out.println(amzCode);
		return amzCode;
	}
	public static Point nextLoc(Point p, String dir, int dist)
	{
		Point nu;
		if(dir.equals("left"))
		{
			nu = new Point((int)(p.getX()-dist), (int)(p.getY()));
		}
		else if(dir.equals("right"))
		{
			nu = new Point((int)(p.getX()+dist), (int)(p.getY()));
		}
		else if(dir.equals("down"))
		{
			nu = new Point((int)(p.getX()), (int)(p.getY()+dist));
		}
		else
		{
			nu = new Point((int)(p.getX()), (int)(p.getY()-dist));
		}
		return nu;
	}
	public static Path[] directions(Point curr, Point tar)
	{
		Path[] direct = new Path[2];
		int rand = (int)(Math.random()*2);
		int xdiff = (int)(curr.getX()-tar.getX());
		int ydiff= (int)(curr.getY()-tar.getY());
		if(rand==0)
		{
			if(xdiff<0)
			{
```

```
                        xdiff = -xdiff;
                        direct[0] = new Path(curr,"right",xdiff);
                }
                else
                {
                        direct[0] = new Path(curr,"left",xdiff);
                }
                if(ydiff<0)
                {
                        ydiff = - ydiff;
                        direct[1] = new Path(tar,"up",ydiff);
                }
                else
                {
                        direct[1] = new Path(tar,"down",ydiff);
                }
        }
        else if(rand==1)
        {
                if(ydiff<0)
                {
                        ydiff = -ydiff;
                        direct[0] = new Path(curr,"down",ydiff);
                }
                else
                {
                        direct[0] = new Path(curr,"up",ydiff);
                }
                if(xdiff<0)
                {
                        xdiff = - xdiff;
                        direct[1] = new Path(tar,"left",xdiff);
                }
                else
                {
                        direct[1] = new Path(tar,"right",xdiff);
                }

        }
        return direct;
    }
}
```

Division of Work

(Copied from Chapter 6)

Lexer/Parser

In terms of the modules, everyone helped to develop the lexer, because that mostly meant hashing out the grammar. Orlando helped to convert the grammar to LL* for use in ANTLR. The parser was also developed from Amaze.g, so again, most of our team members worked to edit the grammar. Syntax analysis was really combined within code generation in EvaluatorWalker.g. The whole group generally developed our individual parts for made checks for validity, but Orlando handled all exceptions being thrown.

Semantic Analysis

All of us generally tried to make checks for semantic analysis, but Orlando made sure of every production one by one and handled exceptions.

Division of Work in Code Generation

Rouault worked on iteration and selection statements. In addition, Rouault helped develop and organize many of the classes that interfaced with the Java code generation. These classes include board, path, point, etc. These were the classes that took the final snippets of code and performed the commands we actually wanted in Java.

Dan worked on main declaration, fixed board_statement_2, organized board declaration for proper code generation, and completed structure.

Orlando created many of the node classes that we implemented to deal with bubbling up Strings and Identifiers up the abstract syntax tree. He also worked with int declarations, function declarations, and all of the productions having to do with expressions.

Jonathan worked on path declarations, point declarations, board list and statements(specifically the simple ones, size, start, and end), and just generally bubbling up strings up the tree throughout the more abstract levels in tree.

Jose worked on while loops and if statements in code generation.

.Sh File

Dan and Rouault worked on the script to build the project jar, include the antlr jar, run the Test file to take in the .amz file, and output the .class file.