Fantasy League Object Oriented Development (FLOOD)

Project Report

COMS W4115: Programming Languages & Translators

Team 9
is

Stephanie Aligbe
System Tester
sna2111@columbia.edu

Elliot Katz
Project Manager
epk2102@columbia.edu

Tam Le
System Architect
tvl2102@columbia.edu

Dillen Roggensinger
System Integrator
der2127@columbia.edu

Anuj Sampathkumaran
Language Guru
as4046@columbia.edu

May 10, 2011

Contents

1

Introduction

As defined on Wikipedia: *fantasy sport (also known as* ***rotisserie****,* ***roto****, or* ***owner simulation****) is a game where participants act as owners to build a team that competes against other fantasy owners based on the statistics generated by the real individual players or teams of a professional sport.*

The popularity of fantasy sports has exploded in recent years. A 2007 study by the *Fantasy Sports Trade Association (FSTA)* estimated nearly 30 million people in the U.S. and Canada, ranging in age from 12 and above, participated in organized fantasy sports leagues. In comparison, an estimated 3 million people played in the early 1990s, swelling to 15 million by the early 2000s. This impressive growth looks to continue since the study revealed teenagers in both the U.S. and Canada play fantasy sports at a higher rate than the national average, with 13 percent of teens playing in the U.S. and 14 percent playing in Canada.

The FSTA study also estimated the spending habits and overall economic impact of fantasy sports players. Consumers engaging in this ever growing hobby spent $800 million directly on fantasy sports products and an additional $3 billion worth of related media products (such as DirecTV's NFL Sunday Ticket and satellite radio's coverage of MLB). Moreover, the growing popularity of fantasy sports is not restricted to North America alone.

A recent 2008 study by a European-based market research company estimated the number of fantasy sports players in Britain range between 5.5 and 7.5 million and vary in age between 16-64, of which 80 percent participated in fantasy soccer. Fantasy gaming has become so entrenched in popular culture, there's even an American comedy sitcom called *The League* in which the main characters obsess over winning their fantasy football league.

Thus, the **FLOOD** programming language is targeted to address a specific problem domain: the creation of fantasy gaming league applications. From the ground up, the language is designed to make it as straightforward as possible for programmers to create fantasy gaming applications. This task will entail defining a league and its type (sports, financial markets, election polls, etc.), establish the rules of governance, enumerate the users/teams and individual league players, and set various other control parameters.

In this sense, **FLOOD** is very much a "high-level" and "domain-specific" language. Such as **R** and **S** are languages designed specifically to perform statistics calculations, **SQL** for relational database queries, and **Mathematica** and **Maxima** for symbolic mathematics, **FLOOD** is dedicated to solving the particular problem of fantasy gaming. Given the ubiquity of fantasy sports as a recreational hobby for millions of people, the creation of a domain-specific language to attack this problem in a clear and concise native programming etymology, as opposed to the application of a general purpose language such as **C** and **Java**, is a challenging and worthwhile undertaking.

2

A Quick Tutorial

The goal of this tutorial is to jump into the **FLOOD** language by creating a simple fantasy league. We won't concern ourselves with implementation details just yet. The language is meant to be simple yet sufficiently extensible. However, the tutorial will not delve into the various features that make this possible.

Experienced Java programmers will be familiar with most of the syntax since it is a subset of the popular C-like syntax. In certain instances we have opted to use Python style syntax for improved readability or Pascal syntax for ease of use. Where possible we compare and contrast a specific feature with Java, Python or some other language which comprise the main influences of **FLOOD**.

Some examples which aren't strictly necessary for the sample program are included to clarify points of possible confusion. These are noted when used. Additionally, we repeat portions of the **FLOOD** standard library for ease of reading. The full library is included in the *Reference Manual* and *Appendix* where appropriate. Please be aware of the distinction made between programmer, the person actually writing a **FLOOD** program (most likely yourself if you are reading this), and *User*, the end-user of the application (which could be thought of as a team entitity). Additionally, *Player* refers to a player of the fantasy league, either an actual person as in the case of a quarterback in football or a non-human entitity

as is the case with *AAPL* (Apple) in the stock market.

2.1 Getting Started - The "Hello World" Program

FLOOD has a very specific application domain. It's goal is to create fantasy leagues and thus, the ability to print to console is fruitless. However, the GUI will have a message box where messages can be displayed to the *User*. **FLOOD** hides the details of the user-interface so the programmer can concentrate on the rules of their specific fantasy league. While the first simple program may seem more complicated than the simple Java "hello world" example, most programs generally will not be any more elaborate than this case.

Since **FLOOD** is built on top of Java byte-code, the programmer will not need to worry about system compatibility issues. Heeding the eternal wisdom of Kernighan & Ritchie, we also point out that compilation will only succeed if the programmer hasn't botched anything, such as missing characters, misspelled keywords, or some other mistake that can cause an error in compilation.

Let's start by showing what the *main* class will look like. Defining the *main* class should look somewhat familiar to those who have programmed in Python.

Listing 2.1: Sample.fld

```
1  /*
2  * Define the league and configure various settings
3  */
4  DefineLeague
5     /* Set league parameters */
6     Set LeagueName("Happy League");
7     ...
8     ...
9     Set MinTeamSize(5);
10
11 /*
12 * Define custom functions
13 */
14 DefineFunctions
15    Void myFunction1(Str param1, Int param2)
```

```
    {
        ...
        ...
    }

    ...

    Void myFunction2(Bool param1, Flt param2)
    {
        ...
        ...
    }
```

Let's now review the notable differences between Python, Java and **FLOOD**. First, the convention in **FLOOD** is to name each file with the *.fld* extension. In addition, each file is a self-contained program and unlike Java, there is no ability to add separate classes. The first letter of any keyword in **FLOOD** is capitalized.

The language was originally intended to be object-oriented in order to facilitate modularity and extensibility of code. However, due to time constraints, it was decided to reduce the scope of the language without losing the basic functionalities as originally promised. Therefore, the ability to add classes and inherit from pre-existing libraries was left for a possible future iteration of the language.

The entry point for a **FLOOD** program is to specify the `DefineLeague` keyword. Since there is no question of class scope within a file, code below this program block is considered part of the program while any code above (excluding comments, naturally) is an error. It is important to note the `DefineLeague` block is semantically equivalent to the `main` function in a standard Java program and all the parameters and settings specified within this block can be viewed as passing parameters via the constructor method in a Java program. Moreover, it is not possible to declare local variables here since meaningful computations do not exist in this section of a **FLOOD** program.

Listing 2.2: Defining the league and setting parameters.

```
/*
* Define the league and configure various settings
```

```
3  */
4  DefineLeague
5     /* Set league parameters */
6     Set LeagueName("Happy League");
7     Set MaxUser(10);
8     Set MinUser(2);
9     Set MaxTeamSize(10);
10    Set MinTeamSize(5);
11
12    /* Add Users */
13    Add User("Eli");
14    Add User("Dillen");
15    Add User("Anuj");
16    Add User("Tam");
17    Add User("Steph");
18
19    /* Add Actions */
20    Add Action("Field Goal Attempt", -0.45);
21    Add Action("Field Goal Made", 1.0);
22    Add Action("Free Throw Attempt", -0.75);
23
24    /* Add Players */
25    Add Player("Lebron James", "forward");
26    Add Player("Chris Bosh", "forward");
27    Add Player("Dwyane Wade", "guard");
```

Every **FLOOD** program implicitly instantiates a *League* object. The *League* object contains most of the other objects needed for the **FLOOD** program including *User*, *Player* and *Action* lists.

For the minimum program, it may be possible to leave the maximum and mininum user settings at their respective default values, but as an example, we set them above. In order to change certain attributes of the *League*, use the `Set` keyword with the attribute to be changed. Additionally, to create new *Users*, *Actions* and *Players* use the `Add` keyword. Since local variables can not be set in the *League* definition, all attributes must be literals. Note that *Users* and *Players* are simply a *string* representing a name, while `Action` is a name-value pair consisting of the name of the `Action` and the value that will be added to a *User's* points through that `Action`.

A **FLOOD** developer need not worry about passing any values to the back-end—**FLOOD** encapsulates the process of passing the *League* to the GUI and calling the GUI.

Listing 2.3: Defining functions.

```
/*
* Defining custom user functions
*/
DefineFunctions
   Void myFunction1(Str param1, Int param2)
   {
      ...
      ...
   }

   ...

   Void myFunction2(Bool param1, Flt param2)
   {
      ...
      ...
   }
```

Before custom functions can be defined, the keyword `DefineFunctions` must be specified. Certain **FLOOD** functions are built into the language and the compiler will check whether the programmer has overridden them. They are as follows:

Listing 2.4: Predefined FLOOD functions.

```
Int draftFunction(int turn) {...}
Bool draftPlayer(User u, Player p) {...}
Bool trade(User u1, Player[] p1, User u2, Player[] p2) {...}
Bool dropPlayer(User u, Player p) {...}
```

If one of the above-mentioned functions is defined in the source file, then **FLOOD** will simply use the programmer's defined function. However, if any of the functions are left out, then **FLOOD** will generate default code for them (*Note: the appendix has a full listing for reference). A function in **FLOOD** is similar to a method in Java or a function in C++. We call a function by using the function's name with an optional argument list in between mandatory parenthesis.

In the following example, assume the function was defined as:

Listing 2.5: Predefined FLOOD functions

```
1 Int draftFunction(Int turn)
2 {
3    /* Number of Users is 10 */
4    Int currentTurn;
5    currentTurn = turn % 10;
6    Return currentTurn;
7 }
```

There are several things to note in the above code snippet. **FLOOD** uses $/*\ldots*/$ comments similar to Java and can span multiple lines. Variables are declared before they are used and all variables are instantiated to default values. Using a variable that hasn't been instantiated will result in a logical error rather than a semantic error. All variable declarations must occur before any other code in the function body.

A `Return` must only occur at the end of the function and must match the stated return type. In the example above, `currentTurn` is declared as an `Int` and then instantiated. Only following the rest of the production body does `currentTurn` get returned.

A function can be called similarly to Java:

Listing 2.6: Function call.

```
1 draftFunction(10);
```

or in an assignment:

Listing 2.7: Function call in an assignment.

```
1 Int a;
2 a = draftFunction(10);
```

Hiding the GUI is one of the main features of **FLOOD**. There is no need to program any part of the user-interface. The GUI will know how to hook into the **FLOOD** source and connect the buttons on the interface to the actions defined by the programmer. The output will be a GUI window which will control the flow of the program and in turn, the flow will depend upon the user's interactions. For instance, the only way to update the scores of the

Users is by adding player-action files to the program. *Player-Action* files would contain new statistics of the format *Player, Action, Quantity* such as:

LeBron James, Rebound, 7
Lebron James, Assist, 4
Carmelo Anthony, Steal, 5
Carmelo Anthony, Point Scored, 25

This is everything needed to run a simple **FLOOD** program. This basketball league will mirror equivalent leagues in *Yahoo! Sports* or *ESPN Fantasy* without network connectivity. The minimum "Hello World" program is included below:

Listing 2.8: Minimal FLOOD program to create a basektball fantasy league.

```
DefineLeague
    Set LeagueName("Basketball League");
    Add User("Anuj");
    Add User("Tam");
    Add Action("Field Goal", 2.0);
    Add Action("Rebound", 1.0);
    Add Player("Lebron James", "forward");
    Add Player("Kobe Bryant", "guard");
    Add Player("Dwight Howard", "center");
    Add Player("Kevin Durant", "forward");

DefineFunctions
  /* None declared */
```

2.2 Variables & Arithmetic Expressions

Just as any robust programming language requires a comprehensive computational model, **FLOOD** provides the user with a set of arithmetic expressions and variable types to work with. To use a variable and work with it, the variable must be declared before it is used for the first time. A declaration defines the properties of the variables. A declaration is of the form `type name` where `type` is the data type of the variable and `name` is the identifier of the variable.

Listing 2.9: Variable declarations.

```
/* variable declarations */
Int i;
Bool b;
Flt f = 1.2; /* ...assigning value at point of declaration */
Str s;

/* ...or assigning values at a later point. */
i = 1;
b = True;
f = 2.1;
s = "Hello World";
```

Here, variables are declared before they are used. It's also possible to assign values to the variables in the declaration. An important distinction between **FLOOD** and a programming language like Java is the location of actual declaration which as noted ***must be at the top*** of the function body. Following the declarations, the variables can be used as needed. Note that there is no coercion between `Flt` and `Int` since **FLOOD** does not support implicit type coercions, as shown here:

Listing 2.10: Implicit type coersion is not supported.

```
Int i;
Flt f;
i = 2.0; /* Error since 2.0 is a Flt */
f = 1; /* Error since 1 is an Int */
```

More examples of assignment expressions which will throw errors due to mismatch types:

Listing 2.11: More errors due to mistmatch type declarations and assignments

```
Int i;
Int f;
f = 1.0;
i = f / f; /* Error: Int assigned a Flt expression */
```

FLOOD offers the set of arithmetic expressions required to create a comprehensive fantasy league of the developer's choice. This set comprises of the standard addition, subtraction, multiplication, division and modulus operators. In addition, statements can include functions as in the case:

Listing 2.12: Function used in an arithmetic statement.

```
1 Int i;
2 i = someFunction() + 10 * 4; /* someFunction() returns an Int
     */
```

The above code snippet provides a glimpse of the arithmetic capabilities of **FLOOD**. The '+', '−', '/', '∗', and '%' operators are binary operators and can be used to add, subtract, divide, multiply and obtain the modulo of `Flt` (floats) and `Int` (integers).

2.3 Loops & Conditionals

Creating a fantasy league in **FLOOD** can range from simple computations to complex algorithms involved in drafts. To facilitate the latter, a **FLOOD** developer has the choice of using loops to make life easier.

The syntax for the *while* loop follows the standard convention:

Listing 2.13: *while* loop

```
1 While (total < 4)
2 {
3    centers = centers + 1;
4    guards = guards - centers;
5    forwards = forwards / 2;
6    total + 1;
7 };
```

The *while* loop operates as follows: the condition in parentheses is tested. If it is true (`total` less than 4), the body of the loop (the three statements enclosed in braces) is executed. Then the condition is checked again and if true, the body is executed again. When the test becomes false (`total` equals or exceeds 4) the loop ends and execution continues at the statement immediately following the loop.

FLOOD provides the developer with a conditional in the form of the *if* expression that is defined as follows:

Listing 2.14: *If* conditional

```
If (position == "center")
{
   position = position + 1;
};
```

Here, the program checks the condition enclosed in the bracket. If this condition is met (in this case, if the person's position is `center`, then the program executes the next statement. **FLOOD** also incorporates $if \ldots else$ conditional statements as follows:

Listing 2.15: *If... Else* conditional statement.

```
If (points > 100)
{
   trade = True;
}
Else
{
   trade = False;
};
```

If the first condition (`points` $>$ 100) is not met, then the statement or statements enclosed in body of the `Else` condition will be executed i.e. assign `False` to the `trade` variable.

2.4 Functions & Scope

As in countless other programming languages, **FLOOD** employs the concept of a function (also referred to as a method, subroutine, or procedure), a logical grouping of code within the larger program which carries out a specific task and is relatively independent of the rest of the code base. The idea of a function is analogous to the notion of the "black box" when discussing the concept of encapsulation in object-oriented programming. For a well-designed function, the particulars of "how" the function performs its task(s) is not of critical importance and just knowing "what" it does suffices. Functions can be "called" or "executed" any number of times and from within other functions.

Listing 2.16: Syntax of a FLOOD function.

```
Bool evaluate(User u, Player p)
{
   ...
}
```

An important requirement to remember is that functions ***must be defined before*** they are used, as shown here:

Listing 2.17: A function must be defined before being called.

```
Bool function1()
{
   Return False;
}

...

Bool function2()
{
   Return function1();
}
```

In this example, function `function1()` is defined before `function2()` and therefore can be used in the body of `function2()`.

The scope of a variable is limited to the function it is declared in—***variables cannot be declared global***. In order to modify variables between functions they must be passed as parameters to the specific functions.

2.5 Users & Players (Arrays)

The types `User` and `Player` are specific to **FLOOD**. They can only be declared as a formal parameter in the argument lists of functions:

Listing 2.18: `User` and `Player` declared in function argument list

```
Bool draftPlayer(User u, Player p) {...}
Bool trade(User u1, Player[] p1, User u2, Player[] p2) {...}
Bool dropPlayer(User u, Player p) {...}
```

The GUI knows to look for these specific functions and populate them correctly. It is possible to define custom functions that take `User` and `Player` as arguments, though **FLOOD** convention recommends against it.

Within **FLOOD**, arrays exists only in the context of `User` and `Player`. Both types can be passed as single values or array types. The array declaration is similar to Java:

Listing 2.19: Passing `User` and `Player` arrays

```
1 Bool trade(User u1, Player[] p1, User u2, Player[] p2) {...}
```

Square brackets are placed after the type before the name of the variable. An array can be accessed in the body of the function using an `Int` index. Both of the examples below are correct:

Listing 2.20: Assessing array elements using index.

```
1 Int i = 5;
2 draft(a[1]);
3 draft(b[i]);
```

FLOOD has a few built-in functions that give the programmer more flexibility in writing functions. The functions are associated with `Users` and `Players` and as such are included in this section. The first function `ArrayLength` simply returns the length of the array that is passed to it. Note that Python similarly uses this kind of syntax to find the length of a list. The other two functions `AddPlayer()` and `RemovePlayer()` are functions that alert the GUI to the addition or removal of a `Player` from a `User`. For the full list of functions, see the *Reference Manual* section *Function Calls*. The syntax is as follows:

Listing 2.21: Some FLOOD utility functions and usage.

```
1 Int i;
2 i = ArrayLength(p);  /* Where p is an array of Players */
3 AddPlayer(u, p); /* Where u is a User and p is a Player */
4 RemovePlayer(u, p); /* Where u is a User and p is a Player */
```

2.6 Alert & Error

FLOOD programs are run through a GUI so print statements conform to this interface. As an alternative to print streams to a console, **FLOOD** allows the programmer to display boxes of text. There are two kinds of boxes, `Alert` and `Error`:

Listing 2.22: Launching `Alert` and `Error` message boxes.

```
1 Alert("Alert Box Title", "Alert message......");
2 Error("Error Box Title", "Error message......");
```

Both message boxes have the same structure. The keyword `Alert` or `Error` must be followed by the standard parenthesis with two arguments. Both arguments must be of type `Str`. The first argument will be the box's title while the second argument will be the message body. The alert message box displays a black exclamation point in a yellow triangle, while the error message box displays a white 'X' in a red circle.

3

Reference Manual

3.1 Introduction

The following is a brief Reference Manual for the **FLOOD** language. The reference manual is based on the K & R C manual which has for the most part inspired Java, the language **FLOOD** is based on. In certain instances, where there was no distinction between **FLOOD** and C, the C reference manual definition was used. Additionally, certain explanations were adapted from the Oracle (formerly Sun) online collection of tutorials.

Many parts of the manual should be familiar to the seasoned programmer. We were able to use **FLOOD** to experiment with ideas that have been born from our collective programming experience.

We begin with a high level view of the structure of a **FLOOD** program. It is divided into two logical blocks. The program begins with the very first block which sets the attributes of the *League*. This block needs to be present and is specified as follows:

Listing 3.1: `DefineLeague`

```
1 /*
2 * Define the League and configure various settings
3 */
4 DefineLeague
5    /* Set League parameters */
```

```
6     Set LeagueName("The League Name");
7     ...
8     ...
9     Set MinTeamSize(5);
```

In this block, various attributes and properties of the *League* are set such as the name of the *League*, the list of *Users*, the list of *Players*, etc. We enumerate the list of possible statements allowed in this block below:

Listing 3.2: Setting league name.

```
1 Set LeagueName("The League Name");
```

The above statement sets the name of the *League*.

Listing 3.3: Setting maximun number of `Users`.

```
1 Set MaxUser(10);
```

Sets a limit on the maximum number of *Users* of the *League*.

Listing 3.4: Setting minimum number of `Users`.

```
1 Set MinUser(2);
```

Sets a limit on the minimum number of *Users* of the *League*.

Listing 3.5: Setting maximum size of each team.

```
1 Set MaxTeamSize(10);
```

Places a limit on the size of each *User's* team.

Listing 3.6: Setting minimum size of each team.

```
1 Set MinTeamSize(2);
```

The minimum number of players that need to be in a *User's* team.

Listing 3.7: Adding a `User`.

```
1 Add User("Name of User");
```

Adds a `User` to the *League*.

Listing 3.8: Adding an `Action`.

```
1 Add Action("Name of Action", 10.0);
```

Adds an `Action` to the *League* and the number of points associated with it.

Listing 3.9: Adding a `Player`.

```
1 Add Player("Name of Player", "Position");
```

Adds the `Player` to the *League* object, along with the position of that `Player`.

The next logical block is comprised of defining functions and invoking functions within them if needed. This segment needs to begin with the `DefineFunctions` keyword as follows:

Listing 3.10: Defining functions.

```
1  DefineFunctions
2      Void myFunction1(Str param1, Int param2)
3      {
4          ...
5          ...
6      }
7
8      ...
9
10     Flt myFunction2(Bool param1, Flt param2)
11     {
12         ...
13         ...
14     }
```

The variables in a function have to be defined at the start of the function and a function needs to be defined before it can be invoked.

3.2 Lexical Conventions

The first phase in the interpretation of the source files is to do a low-level lexical transformation transforming every line to a series of tokens to be compiled in a later stage.

3.2.1 Tokens

There are 5 different types of tokens that exist: identifiers, keywords, string literals, operators and separators. White space and new line are inherently ignored and are merely used as a means of making code legible.

3.2.2 Comments

The sequence of characters initially starting with '/∗' and ending with '∗/' disqualify any of the surrounded characters from the lexical tokenization. Comments do not nest or occur within literals. There are no single-line comments however a single-line comment can be simulated using the above sequence on a single-line.

3.2.3 Identifiers

An identifier is a sequence of letters and digits beginning with a letter. Underscore is also considered a letter. Additionally, the language is case sensitive.

3.2.4 Keywords

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

`DefineLeague`	`DefineFunctions`	`LeagueName`	`Set`	`Add`
`MaxTeamSize`	`MinTeamSize`	`Action`	`Return`	`Str`
`MaxUser`	`MinUser`	`User`	`True`	`False`
`AddPlayer`	`RemovePlayer`	`Player`	`Void`	`While`
`GetUserName`	`GetNumPlayers`	`GetPlayerName`	`If`	`Else`
`GetPlayerPosition`	`GetPlayerPoints`	`ArrayLength`	`Int`	`Flt`

3.2.5 String Literals

A character constant is a sequence of one or more characters enclosed in double quotes, as in "..." Character constants do not contain the ' character or newlines in order to represent them but rather use different escape characters. They are the following:

newline	NL	\n	backslash	\	\\
horizontal tab	HT	\t	double quote	"	\"

3.3 Type Specifiers

FLOOD supports the following datatypes:

- `Void`: Indicates absence of type information.
- `Flt`: A number with decimal values precise up to 6 decimal places.
- `Int`: An integer number.
- `Str`: A String of characters.
- `Bool`: True/ False values.

```
type-specifier:
     Void
     Flt
     Int
     Str
     Bool
```

3.4 Syntax Notation

3.4.1 Variable Declaration

Variables can be declared at the start of each function and must be declared before any statement. Variables can either be simply declared, or can be set to a value during definition as follows:

```
variable-declaration:
     type-specifier variable
     type-specifier variable = value
```

3.4.2 Expressions

Primary Expressions

Identifiers, constants, strings, or expressions in parentheses.

```
primary-expression:
   identifier
   constant
   string
   ( expression )
```

Arithmetic Expressions

FLOOD supports the arithmetic expressions of addition, subtraction, multiplication, division and modulus.

The addition '+,' subtraction '−,' multiplication '∗' and division '/' operators are left associative. The modulus '%' operator is non-associative.

The result of the '+' operator is the sum of the operands. A string may also be added to another string. In this manner, concatenation is performed where the result is a string containing the first operand and then the second beginning at the end of the first.

The result of the '−' operator is the difference of the operands. A string may not be subtracted from another string.

Multiplication and division can be performed only on `Int` and `Flt` types. Additionally, **FLOOD** does not enforce coercion of operand types. This means that any of the above operations can be performed only on operands of the same type. For example, an `Int` variable can only be added to an `Int` variable.

Relational Expressions

The relational operators group left-to-right. The relational expression returns a boolean value.

```
relational-expression:
   variable < variable-or-constant
   variable > variable-or-constant
   variable <= variable-or-constant
   variable >= variable-or-constant
   variable == variable-or-constant
   variable != variable-or-constant
   (relational-expression)
```

The operators '==' and '!=' have lower precedence than the operators '<,' '>,' '<=,' and '>=.' String comparisons are accomplished with the '==' operator and are case sensitive.

Boolean Expressions

The three basic operators involved in boolean expressions are boolean *AND* (&&), boolean *OR* (||) and boolean *NOT* (!). The *AND* and *OR* operators group left-to-right. The *NOT* operator is right associative. The *AND* operator returns true if both operands compared are true, and false otherwise. The *OR* operator returns true if at least one operand is true, and false otherwise. The *NOT* operator returns true if the operand evaluates to false, and false otherwise.

```
boolean-expression:
   boolean-expression && boolean-expression
   boolean-expression || boolean-expression
   relational-expression && relational-expression
   relational-expression || relational-expression
   relational-expression && boolean-expression
   relational-expression || boolean-expression
   boolean-expression && relational-expression
   boolean-expression || relational-expression
   ! boolean-expression
   (boolean-expression)
```

3.4.3 Function Declaration

FLOOD allows the user to declare their own functions in addition to the default functions provided as follows:

```
function-declaration:
   returnType functionName (argumentList)
   {
       statements
   }
```

Now the function *functionName* can be invoked by any other function succeeding it in the program. It is important to note that the user needs to declare the function before invoking it. Another point worth noting is that all variables need to be declared at the start of the function declaration before they can be used.

3.4.4 Statements

Statements in **FLOOD** encompass conditionals, loops, assignments and function calls. Every statement needs to be succeeded by a semicolon.

Conditionals

A conditional statement in **FLOOD** is a statement which checks for a certain condition and processes the succeeding statements depending on the boolean value of the condition evaluated. The syntax is as follows:

```
If (expression)
{
    statements
};


If (expression)
{
    statements
```

```
}
Else
{
    statements
};
```

Here, *expression* can be a boolean expression or relational expression. If the value of the expression evaluates to `True`, then the statement is evaluated, otherwise control is passed to the next statement.

Loops

FLOOD provides the user with a looping structure in the form of the *while* loop. The syntax is as follows:

```
While (expression)
{
    statements
};
```

Here, *expression* can be either a relational expression or boolean expression. In the *while* body, *statements* is executed repeatedly as long as the value of *expression* remains true. The *expression* must have boolean type.

Function Calls

The programmer can invoke functions at any point in the `DefineFunctions` scope as long as the function has been declared before the function call.

```
functionName(parameterList);
```

There are additionally a number of built-in functions the users can invoke without needing to define them:

```
Void AddPlayer(user, player);
```

The above function adds the `Player` object *`player`* to the `User` object *`user`*.

```
Void RemovePlayer(user, player);
```

The above function removes the `Player` object *`player`* from the `User` object *`user`*.

```
Int ArrayLength(user, player);
```

The above function takes as an argument either a `User` or `Player` *`array`* object as the parameter and returns the length of the array.

```
Str GetUserName(user);
```

The above retuns the name of the `User` object *`user`*.

```
Int GetNumPlayers(user);
```

The above returns the number of players of the `User` object *`user`*.

```
Str GetPlayerName(player);
```

The above returns the name of the `Player` object *`player`*.

```
Str GetPlayerPosition(player);
```

The above returns the postion of the `Player` object *`player`*.

```
Flt GetPlayerPoints(player);
```

The above returns the points of the `Player` object *`player`*.

Assignments

Values can be assigned to variables depending on their type. For example, a `Bool` variable can be assigned a value of either `True` or `False`. Similarly, an `Int` variable can be assigned any integer value.

Listing 3.11: Assignments

```
var1 = True; /* if var1 is of type Bool */
var2 = 1; /* if var2 is of type Int */
var3 = 1.0; /* if var3 is of type Flt */
var4 = "string"; /* if var4 is of type Str */
```

Additionally, the return value of an invoked function can be assigned to a variable.

Listing 3.12: Function assignment.

```
var5 = function(); /* if the return type of the function
   matches the data type of var5 */
```

3.5 Scope

The scope to be considered while programming in **FLOOD** is lexical scope of a variable. Variables can only exist within a function, and need to be defined at the start of the function body. Therefore, the scope of a variable is limited to the function in which it is defined. Global variables are not supported in **FLOOD**.

3.6 Grammar

Below is a recapitulation of the grammar that was given throughout the earlier parts of this *Reference Manual*.

program: definitions functions

definitions: DefineLeague definitionlist

```
definitionlist: definitionlist definitionproductions
     empty

definitionproductions:
     Set LeagueName ( string-constant )
     Set MaxUser ( int )
     Set MinUser ( int )
     Set MaxTeamSize ( int )
     Set MinteamSize ( int )
     Add User ( string-constant )
     Add Action ( string-constant, float )
     Add Action ( string-constant, -float )
     Add Player ( string-constant, string-constant )

functions: DefineFunctions functionProductions

functionProductions:
     functionProductions returnType functionName ( argumentLists )
                         { declarations statements returnProduction }
     functionProductions returnType functionName ( argumentLists )
                         { empty }
     functionProductions returnType functionName ( argumentLists )
                         { empty statements returnProduction }
     functionProductions returnType functionName ( argumentLists )
                         { declarations empty returnProduction }
     empty

returnType:
     Void
     Str
     Bool
     Int
     Flt

functionName: identifier

argumentLists:
     argumentLists , argumentList
     argumentList
     empty

argumentList:
     returnType identifier
     User [ ] identifier
     Player [ ] identifier
```

```
     User identifier
     Player identifier

statements:
     statements statement
     statement

statement:
     conditionals
     loop
     relational-expression
     assignment
     functionCall

returnProduction:
     Return identifier
     Return string-constant
     Return int
     Return float
     empty

conditionals:
     If ( relational-expression ) { statements }
     If ( relational-expression ) { statements } Else { statements }
     If ( relational-expression ) { empty }
     If ( relational-expression ) { empty } Else { empty }
     If ( relational-expression ) { statements } Else { empty }
     If ( relational-expression ) { empty } Else { statements }
     If ( boolean-expression ) { statements }
     If ( boolean-expression ) { statements } Else { statements }
     If ( boolean-expression ) { empty }
     If ( boolean-expression ) { empty } Else { empty }
     If ( boolean-expression ) { statements } Else { empty }
     If ( boolean-expression ) { empty } Else { statements }

loop:
     While ( relational-expression ) { statements }
     While ( relational-expression ) { empty }
     While ( boolean-expression ) { statements }
     While ( boolean-expression ) { empty }

declarations:
     declarations declaration
     declaration
```

```
declaration:
     Flt identifier
     Int identifier
     Bool identifier
     Str identifier
     Flt identifier = float
     Int identifier = integer
     Bool identifier = True
     Bool identifier = False
     Str identifier = string-constant

relational-expression:
     identifier <= constant-or-variable
     identifier >= constant-or-variable
     identifier != constant-or-variable
     identifier < constant-or-variable
     identifier > constant-or-variable
     identifier == constant-or-variable
     ( relational-expression )

boolean-expression:
     boolean-expression && boolean-expression
     boolean-expression || boolean-expression
     relational-expression && relational-expression
     relational-expression || relational-expression
     relational-expression && boolean-expression
     relational-expression || boolean-expression
     boolean-expression && relational-expression
     boolean-expression || relational-expression
     ( boolean-expression )
     ! boolean-expression
     identifier
     True
     False

constant-or-variable:
     float
     integer
     identifier

arithmetic-expression:
     arithmetic-expression + arithmetic-expression
     arithmetic-expression - arithmetic-expression
     arithmetic-expression * arithmetic-expression
     arithmetic-expression / arithmetic-expression
```

```
     arithmetic-expression % arithmetic-expression
     ( arithmetic-expression )
     identifier
     float
     integer

assignment: leftSide = rightSide

leftSide: identifier

rightSide:
     arithmetic-expression
     functionCall
     string-constant
     True
     False

functionCall:
     functionName ( parameterList )
     AddPlayer ( identifier , identifier )
     RemovePlayer ( identifier , identifier )
     ArrayLength ( identifier )
     GetUserName (identifier)
     GetNumPlayers (identifier)
     GetPlayerName (identifier)
     GetPlayerPosition (identifier)
     GetPlayerPoints (identifier)

parameterList:
     parameterList , parameterList
     identifier
     integer
     float
     string-constant
     identifier [ integer ]
     identifier [ identifier ]

empty:
```

4

Project Plan (Elliot)

4.1 Processes

Our team was a mix of three undergraduates majoring in Computer Engineering and the Computer Science *Applications* and *Vision and Graphics* tracks, as well as two graduates majoring in the Computer Science *Foundations* and *Systems* tracks The team members hail from Vietnam, Nigeria, India, Israel and Switzerland with backgrounds in industry, the military and various international educational systems. As such, the team came in with very different ideas on development and team-work.

Our team decided early on that group working sessions and pair-programming would be an effective way to ensure constant communication among team members. We set out a schedule to meet every Monday night. Gradually we added Thursday night and then one weekend afternoon. Information was shared among team members through online documentation using **Google Docs** and a special **Google Group** alias. Every submission was drafted on **Google Docs** and every team member contributed. Additionally, the *Product Manager's* log and Bug Tracker were both available online and updated frequently. The code repository was hosted on **GitHub**.

Modules that were worked on by more than one team member would be pair-programmed

with one team member acting as the "driver" and the other as the "observer." Additionally, the observer would act as a code reviewer which speed up development time. Working sessions were generally held in an open space where team members could work on their own sections but also communicate with each other in real-time.

Our team tried to ensure maximum development freedom for each member. There were no restrictions on development tools as long as it was tested and found consistent with the existing systems in place.

4.2 Roles and Responsibilities

Responsibilities for each team member were generally less defined than the roles listed below. Although a division is made, team members frequently contributed code and ideas to other parts of the product. Entire sections of code have evolved over the course of the project making it almost impossible to clearly discern the primary author.

- Elliot, *Product Manager* - Semantics and Error Checking.
- Anuj, *Langauge Guru* - Parser and Grammar.
- Tam, *System Architect* - Parser and Grammar.
- Dillen, *System Integrator* - Backend GUI and Error Checking.
- Stephanie, *System Tester* - Semantics and Unit Testing.

4.3 Style Sheet

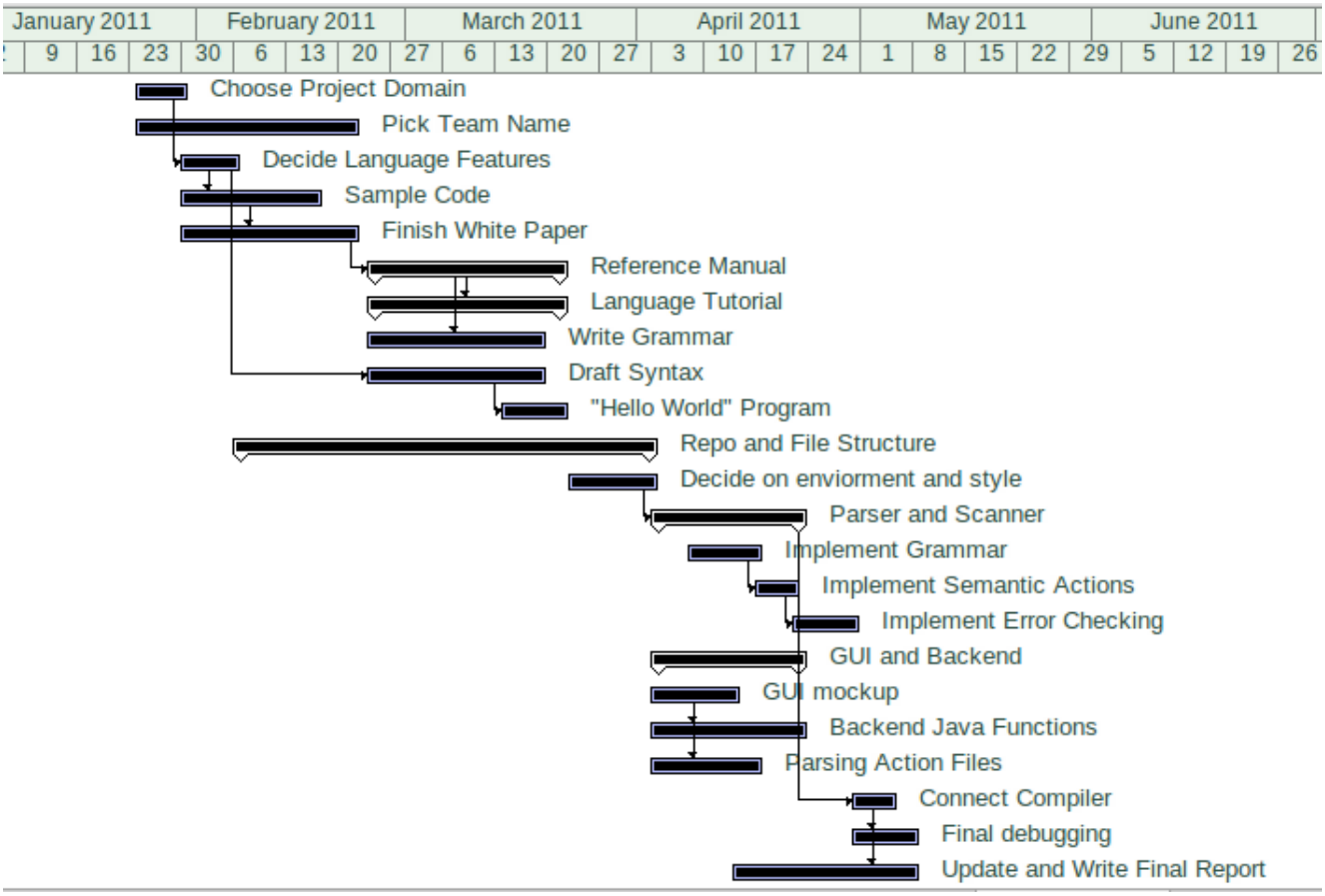
Recognizing that every team member has a unique style of coding, only very specific recommendations were made where differences would change readability. Small issues (such as where to put curly brackets) were left to the team member who wrote the specific section of code.

- Comments:
  - Every method must have a description above it.
  - Sophisticated functions must have comments within the body.
  - Comments should not include `TODO` messages.
- Bugs:
  - All bugs must be tracked using the Google Doc spreadsheet "Bug Tracker."
  - If a bug was found in code that was added to the repo, the team member who found the bug must notify all team members by email.
  - When a bug is corrected, all unit tests regarding the specific section must be re-run to assure that no new code takes away from existing functionality.
- Repository:
  - Check in code to the repository after every major change.
  - Check in code after every session and every four hours within one session.
  - Do not check code in unless it has been tested and a unit test has been written for the functionality added.
  - Update from the repository before a commit to minimize conflicts.
- Code Style:
  - Variable and Function names should be clear and understandable.
  - White Space should be used liberally in order to make code easily readable.
  - The semantic checks exist with the semantic object. The backend runs through GUI. The grammar must treat each one as a blackbox.

- Java naming conventions are the preferred method for style (including but not limited to camel-case, indentation and programming practices) as appears in *Code Conventions for the Java Programming Language, Revised April 20, 1999.*

- Modules:
  - Every module should exist within its own Java file and ideally its own folder.
  - Unit tests should exist with the folder that includes the *makefile* and given names indicative of its use.

4.4 Project Timeline



4.5 Meeting Log

Meeting	Date	Minutes
1	Jan 24th	Brainstormed ideas for programming language. An idea that was discussed: we give users the option of using simple statements to design say maybe a Rubik cube game with a high degree of customisation; not only can the sides of the cubes be coloured, we can leave it to the imagination of the users to say, maybe use texts on the smaller cubes and come up with a creative game involving the Rubik cube or say even a sphere.
2	Jan 26th	More ideas discussed: The Programmer gets to upload the instrumental, type the lyrics in the code/upload the lyrics as well. They get to choose speed of highlighting for any given interval. The onus of syncing it with the corresponding part in the music is on the user. Another idea: Just graphics and the ability to build complex games from shapes. You could have enough power to build tower defense, Angry Birds, Asteroids etc. without the fancy pictures. Just with pure shapes.
3	Jan 30th	Settled on some sort of Fantasy League Game. Some questions discussed: Is there a problem with connecting to the Database? What kind of data-types? Will it be object-oriented? How extensible should our language be? How should we design the syntax? Started draft of white paper.
4	Feb 6th	Continued Working on white-paper. Agreed that language should contain OO and be similar to Java but take elements from Python. Created Git Repo. Created team google-group.
5	Feb 9th	Agreed on Language Breakdown A FLOOD program can: • Create a league and define attributes such as team size. • Establish Rules for play using the existing libraries or overriding them. • Define features of the league such as draft and trade functions. • Connect the league to a database of the user's choice. • Trigger the build for the front end GUI and deploy the program to users.
6	Feb 13th	Continuing work on white-paper. Sketch of what a program would look like. Settled on possible name for project: FLOOD First Commit to new repo.
7	Feb 17th	Discussing feedback from language brainstorming session.

Meeting	Date	Minutes
8	Feb 21th	Final Meeting to finish white-paper. Finalized language features and highlights, data-types, keywords and control-flow. Discussed possible application of a FLOOD program.
9	Feb 23rd	Submitted White Paper.
10	Mar 5th	Discussed Feedback from White-Paper. Where is our computation? How do we want to implement OO?
11	Mar 14th	All day session to work on RM and LT.
12	Mar 17th	All day session to work on RM and LT.
13	Mar 19th	Finishing the grammar, continuing the reference manual. Updated the Language Tutorial to reflect changes in grammar.
14	Mar 23rd	Submitted LT and RM.
15	Apr 3rd	Discussed ways to implement OO and whether to alter to the language and simplify it. Discuss times for final presentation.
16	Apr 8th	Feedback on LT and RM. Beginning to scale down the language to make it easier to write a parser and scanner.
17	Apr 10th	Project File Structure finalised and committed to repo. Division of labor and initial deadlines drawn up. First files committed to repo. Java and FLOOD files used for testing created. Meeting times for the rest of the project: Monday 6:15 pm, Thursday 6:15 pm. One weekend day to be chosen by Thursday 1:00 pm - Longer meeting to work (in CLIC lab).
18	Apr 11th	Revamped Syntax and started sketching GUI mockups.
19	Apr 14th	Working meeting in CLIC lab (see commit comments).
20	Apr 21th	Progress Report. Basic FLOOD file can be parsed.
21	Apr 26th	Working Meeting in CLIC lab (see commit comments).
22	Apr 28th	Working Meeting in CLIC lab (see commit comments).
23	May 2nd	Working Meeting in CLIC lab (see commit comments).
24	May 4th	Putting the final product together. Finished updating the reference manual and language tutorial. Error Checking in place. File FLOOD program can be compiled.
25	May 6th	Last minute debugging. Adding more error productions. Last run-through of every unit test with whole team present.
26	May 8th	Preparing Presentation. Looking over Final Report last time.

5

Language Evolution (Anuj)

At the onset of our language development, we decided to provide the user with as much flexibility as possible. With this as the main focus, our language was aimed at providing object-oriented development to the users. Bearing this in mind, we began developing our language in an incremental manner. We first decided on what the output of the front-end should be, based on what the back-end would expect. In short, we settled on the rendezvous point for the front-end and back-end and worked our way towards it.

As we deliberated on what this intermediate Java code should be, we realized it would be sufficient if the back-end is object-oriented. We therefore decided that the onus of object-oriented programming should not be on the programmer, keeping in mind that users of **FLOOD** will most likely not be familiar with object-oriented concepts. Moreover, we concluded that given the nature of the domain, the programming language being object-oriented would make the learning curve steeper for the user with not much added value. Fully convinced along these lines and the ease-of-use for the programmer in mind, we decided to do away with the object-oriented paradigm.

With the face of the language now modified to provide the user with as much flexibility as needed, yet within the domain of the language as well as keeping it simple to program, we set out to decide the logical division of the program code to aid the user in programming. We

carefully analysed fantasy leagues and realized there exists a common denominator between all fantasy leagues. We decided to push this common denominator to the back-end, leaving the user with the bare minimum yet retain full flexibility to design a fantasy league with the rules of his/her choice.

The language now was developed with a logical division of the program the developer would have to write. The first would need to be the definition of the league; the programmer would need to specify details of the number of users, the names of the users, the players and their scores, valid actions and points associated with these valid actions. This allows the user to define rather quickly the parameters and setup the league.

The next logical block we decided on was defining functions and invoking them within other functions. This section needs to begin with the keyword `DefineFunction`. There are four default functions that each league needs to have which are: `draftPlayer`, `dropPlayer`, `trade`, `draftFunction`. The user can override these functions, although the user can choose not to define these functions, and the default function definitions for these functions will be invoked.

Over and above these four functions, the user can define custom functions and invoke them. In its first incarnation, functions could take any number of parameters of the following types: `Bool`, `Str`, `Int`, `Flt`. Variable declarations were allowed at the beginning of the function before any other statements. The language at this point also provided looping structures, arithmetic expressions, relational expressions as well as conditionals. The back-end required array constructs of objects of a couple of back-end classes, namely the `User` and `Player` classes. Since the programmer does not know the contents of the `User` and `Player` classes, we decided to introduce array constructs in the formal parameters and actual parameter list of functions as follows:

```
functionName(User[] u,Player[] p) { statements }
```

In this manner, when changes were made in the grammar they were incorporated in the back-end and conversely, when changes were made to the back-end, they were incorporated

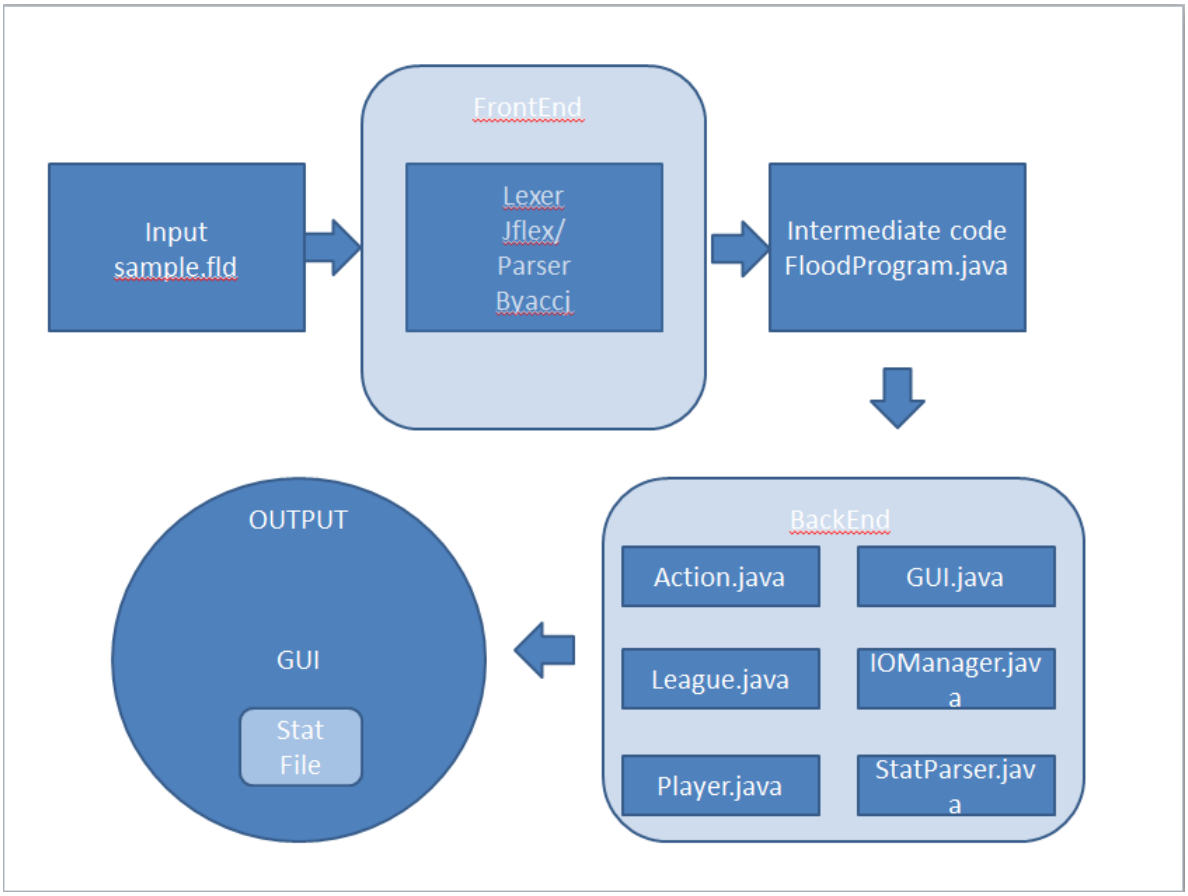
in the grammar. The evolution of the language thus meant the grammar and the back-end evolving in tandem and incrementally.

6

Language Architecture (Tam)

The basic tools used to design the translator are **JFlex** for lexical tokenization and **BYACC/J** for the compiler generator. The lexical tokens are listed in the **JFlex** file, *flood_lex.flex*. **JFlex** tokenizes the input and passes the input in the form of these tokens to the parser, **BYACC/J**.

The grammar that the language follows is defined in the **BYACC/J** file, *flood_grammar.y*. Once this file is compiled, the generated Java file, *Parser.java* is the parser generator for our language. The semantic actions to be performed on the input file given to our compiler are listed in the semantic action of each production in the *flood_grammar.y* file. These semantic actions involve type checking, arithmetic logic, function definitions and variable definitions. If any error is detected in the input file, our parser will not generate the output and will display the error. The high level view is illusrated in the following figure:



If the input is accepted by our strongly typed language, the front-end generates the intermediate code, *FloodProgram.java*. On compilation and execution, the back-end generates the output in the form of a GUI.

The back-end of the system consists of all the Java files that the intermediate code references to generate the output. The GUI of the system makes use of a *Statistics* file which holds the statistics of the proceedings of the league. Based on these statistics, the GUI generates the ranking of each user in the league which determines the winning user at any point.

6.1 Module Authors

- Grammar design and implementation : Tam and Anuj.

- Semantic actions: Dillen, Stephanie and Elliot.
- Backend: Dillen.

7

Development Enviornment (Dillen)

During the development of the **FLOOD** language, development was spread across two operating systems. The majority of the group developed with **Ubuntu** (10.04/10.10/11.04) while one other worked exclusively in **Windows** 7 which ended up working out rather smoothly despite some issues setting up the **Git** repo on **Windows** and maintaining multiple *makefiles*. Yet this cross system development also ensures that our language is not system dependent. In order to maintain a current version of all parts of the language, our group set up an online repository on **GitHub** where changes could be made at any time at the programmer's convenience.

The majority of Java code for the source files used in the intermediate code compilation level was written nearly exclusively in **Eclipse** 3.6 *Helios*. The majority of the back-end was written from scratch with a select few data structures to promote efficiency. Those libraries included primarily the Java API's `HashMap` and `ArrayList` classes when necessary so that looking up references to various *Users*, *Players* and *Actions* would all be $O(1)$. They were also used in a manner to prevent duplicate copies of any particular instance of class so that references (values of references since Java is pass-by-value) were used instead.

In addition to these source files, a GUI was also created to allow user interaction with the program during runtime, a key component of any fantasy league. The development of

the GUI was restricted to purely Java's **Swing** and **AWT** library components with the framework created using Google's **WindowBuilder Pro** web toolkit. With the assistance of the development environment provided by that plugin to **Eclipse**, the GUI's rough layout was generated with ease but was completely devoid of any functionality besides its good looks.

The remainder of the GUI was developed by hand adding various functionalities to different components and promoting ease of integration with files generated by the compiler. No other outside sources were used in the remainder of the production of the source files that integrate with the compiler output besides the occasional Google search on how some feature of **Swing** truly worked, disregarding one's intuition. The final features added to the GUI were the ability for the program to write its current state to a file that could be saved in a directory and then imported at a later date, so that a fantasy league could be easily maintained over the course of any sports season without leaving a program running for the entire duration.

The parser and grammar were implemented with **JFlex** and **BYACC/J** respectively. These are both the Java versions of **Lex** and **Yacc** that the language **FLOOD** compiles to in its intermediate code generation. The majority of the development of the parser and grammar were done in a combination of **Gedit**, **TextPad**, terminal along with some good ol' fashioned paper and pencil. Paper and pencil is really the best (and almost only) way to really work out and recognize shift/reduce conflicts which inevitably helped to understand in general their origins and prevent them. The only exclusion to this trend was the development of the semantic actions that enforced type checking and other semantic checks, such as determining if functions and variables had been previously declared, which were primarily developed in **Eclipse** 3.6 *Helios*. This was done due to the fact that it was exclusively written in Java in which instantaneous compilation helped with simple syntax errors that may have been overlooked.

The most important tool, however, used in the joint production of **FLOOD** was indu-

bitably **Google Docs**. The ability to jointly work on the *White Paper/Language Tutorial/Reference Manual* and watch them come together smoothly as opposed to very disjoint and extremely different written components was truly invaluable. The ability to share and jointly edit other's documents such as schedules, bug reports and other files simultaneously without the burden of constantly pulling and pushing vastly improved efficiency especially when every group member can just consult a checklist to see what components were still incomplete and most urgent. Additionally, the real time updating of files provided a great base for the conversion to **LaTeX** of the final reports. It was a crucial component in the merging of everyone's work throughout the process.

8

Test Plan (Stephanie)

In the early stages of developing the grammar for **FLOOD** most testing simply followed this scheme:

- Write a line of code.
- Test it.
- Does it work?
  - Yes
    - Jump to first statement.
  - No
    - Rewrite line of code.
    - Jump to second statement.

This practice was less about writing production code, and more just to explore and understand **Lex** and **Yacc**.

Soon, the grammar began to take shape. As we developed the expressions and loops, fuller and more thorough unit test programs needed to be generated. As each complete production was a made, at least two accompanying unit test programs were written to test

that production. The two unit tests would test for correctness as well as for failure, meaning that one unit test was meant to be successful, while the other was meant to fail. Often there was more than one unit test intended to fail, because there are multiple ways in which any single part of the program could fail.

Once the grammar reached a self sustainable level, semantic checks began to be implemented. This checks would cover errors that the grammar could not handles on its own. The testing therefore, became more rigorous in order to cover all possible bases. At this point, unit testing included the full team in order to be as thorough as possible. As each member wrote semantic checks or edited the grammar they were encouraged to think of unit tests to break whatever they just wrote. These tests would be written and compiled together as they applied to each specific part of the language, and tested against the compiler. As it became increasingly more difficult to come up with unit tests, we would move on to the next semantic check, determining that the current tests sufficiently checked all possible error and success cases.

8.1 Example Unit Tests

8.1.1 Unit Test 1

Listing 8.1: arithmetic_test.fld

```
1  DefineLeague
2      Set LeagueName("Happy League");
3      Set MaxUser(10);
4      Set MinUser(12);
5      Add User("Eli");
6      Add Action("Field Goal Attempt", 0.5);
7
8  DefineFunctions
9      Int myFunction(Int turn, Flt turn2)
10     {
11         Str a;
12         Str c;
```

```
13        Str d;
14        Bool b = True;
15
16        a = a + c + d;
17        Return a;
18    }
```

The above unit test was meant to test the concatenation of strings. It is a functioning simple **FLOOD** program.

Below is one of it's counterparts. This program is meant to fail because it tries to add a boolean to the strings.

Listing 8.2: arithmetic_test.fld

```
1  DefineLeague
2     Set LeagueName("Happy League");
3     Set MaxUser(10);
4     Set MinUser(12);
5     Add User("Eli");
6     Add Action("Field Goal Attempt", 0.5);
7
8  DefineFunctions
9     Int myFunction(Int turn, Flt turn2)
10    {
11       Str a;
12       Str c;
13       Str d;
14       Bool b = True;
15
16       a = a + c + b;
17       Return a;
18    }
```

8.1.2 Unit Test 2

Listing 8.3: array_var_index.fld

```
1  DefineLeague
2     Set LeagueName("Happy League");
3     Set MaxUser(10);
4     Set MinUser(12);
```

```
5     Add User("Eli");
6     Add Action("Field Goal Attempt", 0.5);
7
8  DefineFunctions
9     Bool random(Player p1)
10    {
11       Bool b;
12       b = True;
13       Return b;
14    }
15
16    Bool trade(User u1,Player[] p1, User u2, Player[] p2)
17    {
18       Int i = 1;
19       Bool b;
20       b = random(p1[i]);
21    }
```

The above unit test is meant to check user defined functions as well as array indexing. Two of its counterparts are shown below. The first fails because the function is defined after it is called.

Listing 8.4: array_var_index.fld

```
1  DefineLeague
2     Set LeagueName("Happy League");
3     Set MaxUser(10);
4     Set MinUser(12);
5     Add User("Eli");
6     Add Action("Field Goal Attempt", 0.5);
7
8  DefineFunctions
9     Bool trade(User u1,Player[] p1, User u2, Player[] p2)
10    {
11       Int i = 1;
12       Bool b;
13       b = random(p1[b]);
14    }
15
16    Bool random(Player p1)
17    {
18       Bool b;
19       b = True;
20       Return b;
```

```
21      }
```

The second fails because it attempts to use a boolean as the index of an array.

Listing 8.5: array_var_index.fld

```
1  DefineLeague
2     Set LeagueName("Happy League");
3     Set MaxUser(10);
4     Set MinUser(12);
5     Add User("Eli");
6     Add Action("Field Goal Attempt", 0.5);
7
8  DefineFunctions
9     Bool random(Player p1)
10    {
11       Bool b;
12       b = True;
13       Return b;
14    }
15
16    Bool trade(User u1,Player[] p1, User u2, Player[] p2)
17    {
18       Int i = 1;
19       Bool b;
20       b = random(p1[b]);
21    }
```

9

Conclusions

9.1 Lessons Learned as a Team

We feel that as a team we have a better understanding of the magnitude of large programming projects. We learned that initial ideas are seldom translated into the final product. When the project first began, we had intended to write a very large language with many features. At the time, none of us truly understood the work that went into implementing even a small language. As we progressed in the semester, the difficulty of the task became more apparent and we had to change our entire language. We feel that we learned a great deal about the more complex aspects of programming languages specifically because we were not able to implement what we had originally intended. Trying to brainstorm features like inheritance and polymorphism taught us about how those components work in popular programming languages.

We learned that teamwork doesn't mean everyone does their own part. It means that people help out and don't stay contained within their respective responsibilities. It means that if someone is having an off-week and/or has an exam the next day, other team members can carry the weight that meeting. We learned that work-schedules need to be implemented early and ambitiously so that deadlines can be stretched within reason if needed.

We each learned different ideas on programming and development from our teammates. Every session of programming there were arguments of proper programming practices or the design of the language. Team members contributed ideas to our language from a wealth of programming experience from languages such as Java, C++, Python and Pascal. We learned that that every team member has particular strengths and those should be utilized as often as possible.

9.2 Lessons Learned by Each Team Member

9.2.1 Stephanie

1. The greatest lesson I learned from this assignment was to *start small and work your way up from there*. Designing a project that implements something you have no idea how to do is extremely difficult. It is all too easy to fall into a very complicated design when the process for creating the pieces is not yet known. In the beginning, my team and I were rather ambitious and naively so. We had dreams of an object-oriented language complete with inheritance, polymorphism, and encapsulation. We even wanted to expand the language onto the **Android** platform if we had time. Somehow we had forgotten what college was: four years of spending every minute of every hour wishing you had more time. We soon realized the we were in way over our heads, and that there was only so much weight Java would carry for us. And so, we had to significantly modify our approach to this project. We first started by simply making cuts to the language, but that left a convoluted and disorganized language. So instead we began anew. We identified all the bare minimum requirements needed for a fully functional language for generating fantasy leagues. From there we could build a base language and then add to that. This process takes away depressing decisions to remove ideas, and replaces it with the exciting realization that we can add things.

2. *Projects of this magnitude need to be constantly worked on.* You can't simply wait until a week before a deadline, put in endless hours of work and expect that to be sufficient. When a project spans four months work must be put into it every week of every month. By getting into lulls where no work is being done on the language, you risk losing focus and generating a bit of a learning curve, if you will, in order to return to the language. Right from the beginning, we made sure to meet regularly, regardless of how much we needed to do. There needed to, at the very least, be a constant reminder of the project, no time for the language to be forgotten.

3. While at times it may appear to be rather difficult, it is important to *always be aware of what each team member is working on.* Not in the sense that you know specifically what they are doing every minute they are working, but just where they are. Indeed, that is primarily the job of the project manager, but I found that it is rather helpful for each member of the team to communicate with the team what their next steps were and how they were going to go about completing their goals. This way there is minimal disconnect between the various parts of the language, especially between semantics and the grammar, where changes in one greatly affect the other. Our group maintained this awareness with the use of **GitHub**, **Google Docs**, and a **Google Group**, in addition to our frequent team meetings and coding sessions. This way, all changes were documented and spread throughout the entire group.

4. *The project is a not a group of people designing pieces of a puzzle that will fit together, after quite a bit of massaging and molding, to produced a result.* It is a team building block after block, each block being guided onto the last, until the structure is complete. The idea to split the group into 5 specific roles helps with ensuring that everyone has a part to do, however I find that this can actually create a disconnect. Instead of creating a strict line on who does what, I feel that having each person contribute to each part of the language builds a more continuous final product. Moreover, it makes

the process of merging the parts of the language significantly more efficient because each member of the teams understands the language and many errors are avoided or resolved quickly by simply just knowing what to look for. In summary it creates a cohesive team, with a thorough understanding of the language.

5. *Commitment is a vital key to any successful team project*, and for a project that has lasted as long as this, it is all too easy to feel overwhelmed. Over the course of the semester fluctuations in one's workload and personal life can take quite a toll on one's ability to function in a group. However, it is in these times that the true power of the group shines through. In the first couple weeks after I fractured my hand, the pain was enough to inhibit my ability to type. In that same time, I had a web site to build, an essay to write, and coding sessions for this project. Had it not been for my team, I would have likely given in to the stress and crumpled under the workload. However, my determination to not let my teammates down allowed me to continue pushing forward. This level of commitment can develop in a team regardless of whether or not the team member knew each other before the formation of the team. When the members of **FLOOD** were established, I only knew half of my teammates. However, in a team of only 5, it is hard to remain cliquish and as a result the team developed rather strongly. Throughout the course, I've have become rather close friends with my team members and will look forward to taking classes with them in the future.

9.2.2 Elliot

While this is not my first time as a project leader, it is my first time leading a software development project. The most important lesson I've learned is the need for constant communication among the team. At times though not often, information would be bottle-necked to a certain person and other team members would not be informed of important changes. The best way to deal with this situation is transparent documentation and meetings. I've also learned how difficult it can be for each team member's unique coding and work style to

be integrated into a cohesive unit. For instance, our version control choice was unfamiliar to some team members at first and it took some time for everyone to get used to working with it.

A project of this magnitude can easily overwhelm any group no matter how experienced. The main goal should be constant pressure. Constant pressure does not mean stress but rather always making progress regardless of the difficulty or clarity of the task. Tasks should be constantly broken down into smaller pieces and distributed as evenly as possible. When a task is assigned, it shouldn't be assumed that the team member will be able to complete it but rather teammates should look out for each other and help when needed. The role of a product manager is not to dictate and assign tasks as they please. Every team member should be given the opportunity to pick specific tasks that they find most interesting. Tasks that were deemed uninteresting were distributed evenly so that no one took more than their fair share of "grunt" work. Decisions should not only be democratic but accepted by the whole team. If even one team member didnt like an idea or an implementation detail, the team worked hard to come up with an alternative that was acceptable to everyone. A product manager should also keep accurate records and have awareness of every moving piece in the project. Ultimately, as the saying goes, the "The Buck Stops Here." This is a lesson every good product manager must learn and usually the hard way.

On a personal note, I feel comfortable saying I understand how a compiler works on a very applied level. I still don't feel comfortable with much of the theory that goes into compilers and in a way it has given me a healthy respect for theorists who have paved the way for modern programming languages. I learned to appreciate the intricacies of programming languages and notice subtle differences that were not obvious to me beforehand. I have already seen a difference in my code production in other projects. In my opinion, one of the most fascinating aspects of software development is the relationship among the developers. Working with other people, especially in an academic setting, is always challenging but it can also be very rewarding. I truly enjoyed the time spent working with my teammates

and regardless of the final outcome of this project or the class in general, I feel that I have learned a great deal from them.

9.2.3 Tam

Ah, the project post mortem. Having recently worked in industry, I have personally been an active (and at times, unwilling) participant of countless such "taking stock" couch sessions. More often than not, and especially after a grueling projects which taxed the mind and body as well as the soul, my own thoughts inevitably turned towards alcohol—nothing like dousing the brain cells which served me so well during those sleepless nights of feverish coding!

But after the welcome, and hopefully pre-paid, happy hour courtesy of a sympathetic Project Manager, gathering one's thoughts and indeed "taking stock" to formulate a list of lessons learned is an invaluable exercise, one which will yield far-reaching benefits down the line for both developers and project managers alike.

At this stage of the game, I find myself asking a similar set of questions:

1. **First and foremost, are you proud of the final product?**

   I will not be so presumptuous as to answer for my fellow group members, but I can say without hesitation that yes, I am very proud of **FLOOD**. Admittedly, certain features and overall scope were made road kill on the highway to making delivery, but such are the vagaries of software development. In fact, I now recall with particular fondness our formative group meetings in which, undoubtedly pie-in-the-sky-ish in retrospect, we brainstormed and flung around countless ideas about what our language will be capable of once completed. An Android-enabled interface! A Facebook port! Automatic texting of scores and live updates to subscribed listeners! Object-orientation that will make Java blush!

   Not surprisingly, the majority of these "nice to haves" fell by the wayside as the project progressed, but did we as a group accomplish what we set out to do originally, which was to write a language that made creating a fantasy gaming league relatively easy for

the programming novice? I would say the answer is a resounding *YES!*

And in undertaking this project, did I myself take away what is perhaps the most crucial aspect of completing the project and taking this class in the first place, which was to gain and learn from first-hand experience the inner workings of compilers and programming languages? Again, an uniequivocal *YES!* Lexical analysis, semantic analysis and actions, code generation and optimization...nothing like rolling up one's sleeves and learning these concepts the hardway: by actually doing it.

2. **What was the single most frustrating part of the project?**

   Hands down, tackling the lexical and semantic analysis of the grammar. Hours upon hours consummed in devising regular expressions that would do what we wanted it to do, then tracking down and eliminating with extreme prejudice countless shift/reduce reduce/reduce errors. With my comrade in arms, Anuj, the two of us now feel like battle-hardened veterans returning from the front-line trenches in our seemingly neverending battle against the Lexical/Semantic axis of power.

   Did **FLOOD**'s grammar put up a valiant fight? Yes. Was it frustating to slay this merciless dragon? Yup...Anuj and I certainly have the mental scars to show for it. But in the end, was it rewarding? Most definitely. I have been involved in few software development projects where the most frustrating part was also the most personally fulfilling. This was one of them.

3. **How would you do things differently next time to avoid this frustration?**

   Start learning the concepts and obtaining the knowledge one will need as early in the project timeline as possible. I believe my biggest mistake was taking such a timid approach towards the grammar—instead of waiting for the material to be presented and covered in class, we (I'm certain Anuj would agree) should have been more proactive in acquiring the know-how and skill sets needed to write **FLOOD**'s grammar. And speaking for myself once more, there was also an intimidation factor at play since

the tools for creating the front-end components of a compiler seemed truly daunting to me to work with. It's difficult at times teaching an old dog new tricks.

4. **What was the most gratifying and personally satisfying part of the project?**
   This old dog learned a new trick. I didn't know how to write a lexical parser and scanner before. Now I do.

5. **Which of the development methods or processes worked particularly well?**
   The pair programming Anuj and I undertook in developing the grammar. Over time we developed a comfortable working style that proved to be most productive. I was skeptical at first regarding this programming paradigm since nearly all of my industry experience has been working and coding in isolation, but engaging in pair progamming for the very first time was a truly eye-opening experience. Where before I would run into a dead-end in some coding dark alley, Anuj was often there to shine a flash light and lead the way out.

6. **If you could wave a magic wand and change anything about the project, what would you change?**
   More time with my fellow **FLOOD**-ites. I was very blessed to have lucked into this group of highly motivated, extremely hard working, and very intelligent individuals. Sure, there were disagreements, some heated discussions, the occasional philosophical differences, but in the end we had a blast working together to give birth to **FLOOD**. I wish all of my future (and certainly several past) collaborative efforts will be as enjoyable and mutually beneficial as this one. I learned a great deal from my team and I hope I have reciprocated in kind—a true highlight of my academic career at Columbia.

9.2.4 Dillen

1. **Choose your group members wisely.**

   Choosing the right people for a group is beyond the most important part of an assignment. Although it may seem unclear what everyone's exact roles may be for a given project, be sure to pick people who are passionate and have deep interest in different components as well as the overall idea of a project. Nothing can be more detrimental to a group that to have members who are unenthusiastic or not willing to pull their own weight, not due to a lack of ability (because let's be honest, no one knew how a compiler worked when we first started) but rather a lack of motivation. Our group functioned extremely well because we all split up the parts into relatively equal work loads and when one part "overflowed" due to unforeseen complexities, those with "easier" tasks jumped in to help. Another key part of our success was the general interest in our idea, which became my next lesson learned.

2. **Thoroughly develop your idea, and then do it again.**

   It is important to spend adequate time to thoroughly develop an idea and delve past the surface challenges and really research potential dead-ends. Unlike most groups (I believe), we met every week from the moment our group was formed until the end of the semester, even when we all lacked any detailed view of how to complete the task. The first several meeting were literally 4+ hour long discussions on potential programming languages where everyone would express interest, play devil's advocate and expand on ideas. The most difficult part for us was to propose ideas feasible for a semester long project without simply creating a markup language. This was crucial in creating a thorough blueprint for our language and the writing of the *White Paper/Language Tutorial/Reference Manual*.

3. **Don't reinvent the wheel.**

   One of the most valuable parts of any programming project is to utilize tools provided

online to ease the pain. Development tools such as **WindowBuilder Pro** for the GUI, **BYACC/J** and **JFlex** for the grammar and parser, **GitHub** for the online repository and **Google Docs** for the written portions provided greater ease and efficiency during the entire process. *USE THEM!*

4. **Pick a good project manager who can bring everyone together.**

   It is easy to get lost in time when everyone is specializing on a specific part of a project and it's the product manager's job to keep bringing everyone together and making sure everything is running smoothly and no individual is lagging behind. Collaboration and communication are absolutely key in the software development world because otherwise, come submission time, everyone has a bunch of extremely well developed but independent portions of the project with integration being nearly impossible. I noticed that our group was exceptionally productive when we all worked in one long line in the CLIC lab, each with the *Google Bug Report Document* constantly asking questions, finding bugs and delegating work with various priority. I realize it is difficult for a bunch of college students to find time to all meet and work for a few hours but those hours spent working together are far more valuable than any work you will do by yourself.

9.2.5 Anuj

1. **Grammar design is not a walk in the park.**

   Designing the grammar for a language is not easy. Designing a small grammar for small units, like say, just arithmetic expressions, is easy. As the grammar evolves, the complexity increases linearly. It helps to have a clear idea of what the language will look like and stub out a rough design of the grammar first. This methodoloy exposes the potential shift/reduce and reduce/reduce errors right from the beginning. It also helps to understand fully how these errors occur and how to eliminate them.

2. **Decide the scope of the language/project realistically.**

   One of the most important lessons I learned from this project is to do a strong feasibility study before proposing a project. A thorough study should be performed to roughly estimate what problem statement the project proposes to tackle. It should neither be too ambitious nor too trivial. Getting the right balance is a tricky task and it seems safer to project the latter and extend the scope of the project if time permits. Another important lesson I learned is never to commit any major feature in the name of the project unless absolutely certain that the feature will be implemented!

3. **Working in a group.**

   To begin with, I had an interesting time working with my group. It not only made working on a project of this magnitude seem effortless, but I also learned a lot from my group on various fronts. This made me realise how important team members are and the dynamics of the group can make or break a project. While it may seem more hands on deck means less work per person, I also realised it requires good management on the part of the Project Manager to coordinate and integrate the project well; something I was privy to during the course of this project.

9.3 Advice for Future Teams

- ***Never*** promise features of your programming language in its name. We tried very hard to come up with a "cool" name that described our language. The name is still "cool" but it doesn't describe our language anymore.

- Start early and meet often. It may seem futile to meet when you have nothing to discuss but just by being in the same room you may start thinking of an idea that changes your project.

- Ask for help no matter how small of an issue. We utilized our TA, Hemanth, as much

as possible and we are truly indebted to him for all his valuable advice.

- Don't be scared of unfamiliar technology; free software is your friend. Our entire project was written and shared in **Google Docs**, the code will forever be saved on **GitHub** and the documentation lives beautifully in **LaTeX**.

- Revisit your idea as much as possible. Our language has morphed from an object-oriented language with Java-like features into a smaller, more compact version. If we had decided to go ahead with our original idea we would not have finished our project or it would have been poorly implemented.

- Pair programming may seem like a waste of time but it dramatically reduces net production time. Most of the code produced was written at one work station with two team members working at a time. Often code that seemed very difficult to one programmer was trivial to another. When a programmer was tired, they could switch and development continued.

- Pick a project that is interesting to you rather than trying to impress anyone. Every team member was truly interested in our project which made brainstorming more fun and productive.

- A team that eats together works better. It's better to work on a full stomach and it's more fun to brainstorm over dinner. We personally recommend the Dominoes 7-7-7 deal.

9.4 Suggestions for Future Improvement

It's very difficult to write this section as a team since each team member had different topics that they enjoyed. We all agree that the project was the highlight of the course and very much enjoyed working on it. Some of us feel that the classes were a little theory-heavy in what was originally expected to be a more applied course in compilers. Some of the theory,

especially the later topics such as code-generation and optimization as well as data-flow analysis might have been better served with more homework and possibly programming exercises.

The first homework gave everyone a taste of **Flex** and **Yacc** which helped both in understanding the theory and what was expected later in the project. Those of us who had taken *Computer Science Theory* enjoyed seeing how the theory we had previously learned applied in the "real-world." We all feel that this was a difficult class but in the end truly rewarding.

Appendix A

FLOOD Source Code

Listing A.1: Action.java

```
1  /*
2   * Action.java
3   * This class handles the various actions in the fantasy
       league
4   */
5
6  public class Action {
7     private String action;
8     private float points;
9
10    /**Constructor.
11     *
12     * @param String action
13     * @param float points
14     */
15    public Action(String action,double points){
16       this.action=action;
17       this.points=(float)points;
18    }
19
20    /**Returns the string representation of the action
21     *
22     * @return String action
23     */
24    public String getAction() {
25       return action;
26    }
```

```
27
28      /**Sets the action
29       *
30       * @param String action
31       */
32      public void setAction(String action) {
33          this.action = action;
34      }
35
36      /**Return the points gained for completing the action
37       *
38       * @return float points
39       */
40      public float getPoints() {
41          return points;
42      }
43
44      /**Sets the points gained for completing the action
45       *
46       * @param float points
47       */
48      public void setPoints(int points) {
49          this.points = points;
50      }
51
52      /**Determines if two actions are equal by comparing
53       * the action and points received
54       *
55       * @return boolean isEqual
56       */
57      public boolean equals(Object obj){
58          if (this == obj)  //Reference to seld
59              return true;
60          if (obj == null)  //Null reference
61              return false;
62          if (getClass() != obj.getClass())   //Not the same class
63              return false;
64          final Action other = (Action) obj;
65          if (action.equals(other.action) && points==other.points)
                //Equal action and points
66              return true;
67          return false;
68      }
69
70      /**Returns a string representation of the action
```

```
71      *
72      * @return String representation
73      */
74     public String toString() {
75        return "Action [action=" + action + ", points=" + points
              + "]";
76     }
77
78 }
```

Listing A.2: flood_grammar.y

```
1  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
2  *
3  * flood_grammar.y
4  * FLOOD
5  * Syntactic/Semantic Analyzer
6  *
7  * Lasciate ogne speranza, voi ch'intrate.
8  *
9  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
10
11 %{
12   import java.lang.Math;
13   import java.io.*;
14   import java.util.Hashtable;
15   import java.util.ArrayList;
16   import java.util.Iterator;
17   import java.util.HashMap;
18   import java.util.*;
19 %}
20
21 /* YACC Declarations */
22 %token PLUS                  /* + */
23 %token OPEN_CURLY            /* { */
24 %token CLOSE_CURLY           /* } */
25 %token OPEN_PARAN            /* ( */
26 %token CLOSE_PARAN           /* ) */
27 %token OPEN_SQUARE           /* [ */
28 %token CLOSE_SQUARE          /* ] */
29 %token <dval> FLT            /* Float */
30 %token <ival> INT            /* Integer */
31 %token <sval> STRING_CONST   /* String literal onstants */
32 %token COMMA                 /* , */
33 %token DOT                   /* . */
```

```
34 %token <sval> ID              /* Identifier */
35 %token EQUAL                  /* = */
36 %token NOTEQUAL               /* != */
37 %token LESSEQUAL              /* <= */
38 %token GREATEQUAL             /* >= */
39 %token ISEQUAL                /* == */
40 %token LESS                   /* < */
41 %token GREAT                  /* > */
42 %token PLUS                   /* + */
43 %token MINUS                  /* - */
44 %token MULT                   /* * */
45 %token DIV                    /* / */
46 %token NOT                    /* ! */
47 %token AND                    /* && */
48 %token OR                     /* || */
49 %token MOD                    /* % */
50 %token SEMICOLON              /* ; */
51
52 /* Keywords */
53 %token DefineLeague           /* Define the league */
54 %token DefineFunctions        /* Define functions */
55 %token LeagueName             /* Set league name */
56 %token MaxUser                /* Set the max user */
57 %token MinUser                /* Set the min user */
58 %token MaxTeamSize            /* Set the max team size */
59 %token MinTeamSize            /* Set the min team size */
60 %token Set                    /* Set keyword */
61 %token Add                    /* Add keyword */
62 %token Action                 /* Action keyword */
63 %token Alert            /* Alert message*/
64 %token Error            /* Error message*/
65 %token User                   /* User keyword */
66 %token Player                 /* Player keyword */
67 %token Void                   /* Void keyword */
68 %token Str                    /* String keyword */
69 %token Bool                   /* Boolean keyword */
70 %token Flt                    /* Float keyword */
71 %token Int                    /* Integer keyword */
72 %token Return                 /* Return keyword */
73 %token If                     /* If keyword */
74 %token Else                   /* Else keyword */
75 %token While                  /* While keyword */
76 %token True                   /* True keyword */
77 %token False                  /* False keyword */
78 %token RemovePlayer           /* Remove Player function */
```

```
79 %token AddPlayer              /* Add Player function */
80 %token ArrayLength            /* Length of an array */
81 %token GetUserName         /* Gets the name of the user object
      */
82 %token GetNumPlayers          /* Gets the number of players of
       the user*/
83 %token GetPlayerName          /* gets the name of the player
      object*/
84 %token GetPlayerPosition      /* Gets the position of the
      player*/
85 %token GetPlayerPoints        /* Gets the points of the player
      */
86
87 /* Associativity and Precedence */
88 %left MINUS PLUS COMMA
89 %left MULT DIV
90 %right NOT NEG
91 %nonassoc EQUAL NOTEQUAL LESSEQUAL GREATEQUAL ISEQUAL LESS
      GREAT AND OR MOD DOT
92
93 /* Types */
94 %type <sval> definitions
95 %type <sval> definitionlist
96 %type <sval> definitionproductions
97 %type <sval> functions
98 %type <sval> functionProductions
99 %type <sval> returnType
100 %type <sval> dataType
101 %type <sval> functionName
102 %type <sval> argumentLists
103 %type <sval> argumentList
104 %type <sval> empty;
105 %type <sval> returnProduction
106 %type <sval> returnProd
107 %type <sval> statement
108 %type <sval> statements
109 %type <sval> conditionals
110 %type <sval> loop
111 %type <sval> declarations
112 %type <sval> declaration
113 %type <sval> relationalExp
114 %type <sval> arithmeticExp
115 %type <sval> booleanExp
116 %type <sval> constOrVar
117 %type <sval> leftSide
```

```
%type <sval> rightSide
%type <sval> assignment
%type <sval> functionCall
%type <sval> parameterList

%%

/*************************************************
* program
*************************************************/
program: definitions functions
{
   if(semantics.validProgram)
   {
      generateFloodProgram($1, $2);
      System.out.println("Total number of lines in the input:
         " + (yyline - 1));
   }
   else
   {
      System.out.println(semantics.printErrors());
   }
};

definitions: DefineLeague definitionlist { $$ = $2; };

definitionlist: definitionlist definitionproductions { $$ = $1
    + $2; }
              | empty { $$ = $1; }
              ;

definitionproductions: Set LeagueName OPEN_PARAN STRING_CONST
   CLOSE_PARAN SEMICOLON { $$ = "myLeague = new League(" + $4 +
    ");\n"; }
                     | Set MaxUser OPEN_PARAN INT CLOSE_PARAN
                        SEMICOLON { $$ = "myLeague.setMaxUser("
                         + $4 + ");\n"; }
                     | Set MinUser OPEN_PARAN INT CLOSE_PARAN
                        SEMICOLON { $$ = "myLeague.setMinUser("
                         + $4 + ");\n"; }
                     | Set MaxTeamSize OPEN_PARAN INT
                        CLOSE_PARAN SEMICOLON { $$ = "myLeague.
                        setMaxTeamSize(" + $4 + ");\n"; }
                     | Set MinTeamSize OPEN_PARAN INT
                        CLOSE_PARAN SEMICOLON { $$ = "myLeague.
```

```
                        setMinTeamSize(" + $4 + ");\n"; }
152                   | Add User OPEN_PARAN STRING_CONST
                        CLOSE_PARAN SEMICOLON { $$ = "myLeague.
                        addUser(new User(" + $4 + "));\n";
                        semantics.addUser($4, yyline); }
153                   | Add Action OPEN_PARAN STRING_CONST
                        COMMA FLT CLOSE_PARAN SEMICOLON { $$ =
                        "myLeague.addAction(new Action(" + $4 +
                         ", " + $6 + "));\n"; semantics.
                        addAction($4, yyline); }
154           | Add Action OPEN_PARAN STRING_CONST COMMA MINUS
                 FLT CLOSE_PARAN SEMICOLON %prec NEG { $$ = "
                 myLeague.addAction(new Action(" + $4 + ", -" + $7
                  + "));\n"; semantics.addAction($4, yyline); }
155                   | Add Player OPEN_PARAN STRING_CONST
                        COMMA STRING_CONST CLOSE_PARAN
                        SEMICOLON { $$ = "myLeague.addPlayer(
                        new Player(" + $4 + ", " + $6 + "));\n"
                        ; semantics.addPlayer($4, yyline); }
156                   ;
157
158 functions: DefineFunctions functionProductions { $$ = $2 +
      semantics.setFlags(); };
159
160 functionProductions: functionProductions returnType
      functionName OPEN_PARAN argumentLists CLOSE_PARAN OPEN_CURLY
        declarations statements returnProduction CLOSE_CURLY { $$
      = $1 +"public static "+ $2 + " " + $3 + "(" + $5 + ")\n{\n"
      + $8 + $9 + $10 + "\n}\n"; this.scope = $3; semantics.
      addToFunctionTable($3, $2, $5, yyline); semantics.
      checkReturnTypeMatch($2, yyline); }
161                   | functionProductions returnType
                        functionName OPEN_PARAN argumentLists
                        CLOSE_PARAN OPEN_CURLY empty
                        CLOSE_CURLY {$$ = $1 +"public static "+
                         $2 + " " + $3 + "(" + $5 + ")\n{\n}\n"
                        ; this.scope = $3; semantics.
                        addToFunctionTable($3, $2, $5, yyline);
                         semantics.checkReturnTypeMatch($2,
                        yyline); }
162                   | functionProductions returnType
                        functionName OPEN_PARAN argumentLists
                        CLOSE_PARAN OPEN_CURLY  empty
                        statements returnProduction CLOSE_CURLY
                         {$$ = $1 +"public static "+ $2 + " " +
```

```
                            $3 + "(" + $5 + ")\n{\n" + $8 + $9 +
                           $10 + "\n}\n"; this.scope = $3;
                           semantics.addToFunctionTable($3, $2, $5
                           , yyline); semantics.
                           checkReturnTypeMatch($2, yyline); }
163                      | functionProductions returnType
                           functionName OPEN_PARAN argumentLists
                           CLOSE_PARAN OPEN_CURLY  declarations
                           empty returnProduction CLOSE_CURLY {$$
                           = $1 +"public static "+ $2 + " " + $3 +
                            "(" + $5 + ")\n{\n" + $8 + $9 + $10 +
                           "\n}\n"; this.scope = $3; semantics.
                           addToFunctionTable($3, $2, $5, yyline);
                            semantics.checkReturnTypeMatch($2,
                           yyline); }
164                      | functionProductions returnType
                           functionName OPEN_PARAN argumentLists
                           CLOSE_PARAN OPEN_CURLY returnProd
                           CLOSE_CURLY {$$ = $1 +"public static "+
                            $2 + " " + $3 + "(" + $5 + ")\n{\n" +
                           $8 + "\n}\n"; this.scope = $3;
                           semantics.addToFunctionTable($3, $2, $5
                           , yyline); semantics.
                           checkReturnTypeMatch($2, yyline); }
165                      | empty { $$ = $1; }
166                      ;
167
168 dataType: Str { $$ = "String"; }
169         | Bool { $$ = "boolean"; }
170         | Int { $$ = "int"; }
171         | Flt { $$ = "float"; }
172         ;
173
174 returnType: Void { $$ = "void"; }
175           | Str { $$ = "String"; }
176           | Bool { $$ = "boolean"; }
177           | Int { $$ = "int"; }
178           | Flt { $$ = "float"; }
179           ;
180
181 functionName: ID { $$ = $1; };
182
183 argumentLists: argumentLists COMMA argumentList { $$ = $1 + ",
      " + $3; }
184              | argumentList { $$ = $1;}
```

```
185                | empty { $$ = $1; }
186                ;
187
188 argumentList: dataType ID { $$ = $1 + " " + $2; semantics.
       addVar($2,$1,yyline); }
189             | User OPEN_SQUARE CLOSE_SQUARE ID { $$ = "User[]
                   " + $4; semantics.addVar($4,"User[]",yyline); }
190             | Player OPEN_SQUARE CLOSE_SQUARE ID { $$ = "
                   Player[] " + $4; semantics.addVar($4,"Player[]",
                   yyline); }
191             | User ID { $$ = "User " + $2; semantics.addVar($2
                   ,"User",yyline); }
192             | Player ID { $$ = "Player " + $2; semantics.
                   addVar($2,"Player",yyline); }
193             ;
194
195 statements: statements statement SEMICOLON { $$ = $1 + $2; }
196           | statement SEMICOLON { $$ = $1; }
197           ;
198
199 statement: conditionals { $$ = $1; }
200          | loop { $$ = $1; }
201          | relationalExp { $$ = $1; }
202          | assignment { $$ = $1; }
203          | functionCall { $$ = $1 + ";\n"; }
204          ;
205
206 returnProduction: Return ID SEMICOLON { $$ = "return " + $2 +
       ";"; semantics.setReturnProdType(semantics.getType($2)); }
207                 | Return STRING_CONST SEMICOLON { $$ = "return
                       " + $2 + ";"; semantics.setReturnProdType("
                      String"); }
208                 | Return INT SEMICOLON { $$ = "return " + $2 +
                       ";"; semantics.setReturnProdType("int"); }
209                 | Return FLT SEMICOLON { $$ = "return " + $2 +
                       ";"; semantics.setReturnProdType("float");
                      }
210                 | Return True SEMICOLON { $$ = "return true;";
                       semantics.setReturnProdType("boolean"); }
211                 | Return False SEMICOLON { $$ = "return false;
                      "; semantics.setReturnProdType("boolean"); }
212                 | empty { $$ = $1; semantics.setReturnProdType
                      ("void"); }
213                 ;
214
```

```
215 returnProd: Return ID SEMICOLON { $$ = "return " + $2 + ";";
      semantics.setReturnProdType(semantics.getType($2)); }
216                 | Return STRING_CONST SEMICOLON { $$ = "return
                      " + $2 + ";"; semantics.setReturnProdType("
                     String"); }
217                 | Return INT SEMICOLON { $$ = "return " + $2 +
                      ";"; semantics.setReturnProdType("int"); }
218                 | Return FLT SEMICOLON { $$ = "return " + $2 +
                      ";"; semantics.setReturnProdType("float");
                     }
219                 | Return True SEMICOLON { $$ = "return true;";
                      semantics.setReturnProdType("boolean"); }
220                 | Return False SEMICOLON { $$ = "return false;
                     "; semantics.setReturnProdType("boolean"); }
221     ;
222
223 conditionals: If OPEN_PARAN relationalExp CLOSE_PARAN
      OPEN_CURLY statements CLOSE_CURLY { $$ = "if(" + $3 + ")\n{\
      n" + $6 + "}\n"; }
224             | If OPEN_PARAN relationalExp CLOSE_PARAN
                   OPEN_CURLY statements CLOSE_CURLY Else
                   OPEN_CURLY statements CLOSE_CURLY { $$ = "if(" +
                    $3 + ")\n{\n" + $6 + "}\nelse\n{\n" + $10 + "}\
                   n"; }
225             | If OPEN_PARAN relationalExp CLOSE_PARAN
                   OPEN_CURLY empty CLOSE_CURLY { $$ = "if(" + $3 +
                    ")\n{\n}\n"; }
226             | If OPEN_PARAN relationalExp CLOSE_PARAN
                   OPEN_CURLY empty CLOSE_CURLY Else OPEN_CURLY
                   empty CLOSE_CURLY { $$ = "if(" + $3 + ")\n{\n}\
                   nelse\n{\n}\n"; }
227             | If OPEN_PARAN relationalExp CLOSE_PARAN
                   OPEN_CURLY statements CLOSE_CURLY Else
                   OPEN_CURLY empty CLOSE_CURLY { $$ = "if(" + $3 +
                    ")\n{\n" + $6 + "}\nelse\n{\n}\n"; }
228             | If OPEN_PARAN relationalExp CLOSE_PARAN
                   OPEN_CURLY empty CLOSE_CURLY Else OPEN_CURLY
                   statements CLOSE_CURLY { $$ = "if(" + $3 + ")\n
                   {\n}" + "\nelse\n{\n" + $10 + "}\n"; }
229             | If OPEN_PARAN booleanExp CLOSE_PARAN OPEN_CURLY
                   statements CLOSE_CURLY { $$ = "if(" + $3 + ")\n
                   {\n" + $6 + "}\n"; }
230             | If OPEN_PARAN booleanExp CLOSE_PARAN OPEN_CURLY
                   statements CLOSE_CURLY Else OPEN_CURLY
                   statements CLOSE_CURLY { $$ = "if(" + $3 + ")\n
```

```
                {\n" + $6 + "}\nelse\n{\n" + $10 + "}\n"; }
231             | If OPEN_PARAN booleanExp CLOSE_PARAN OPEN_CURLY
                  empty CLOSE_CURLY { $$ = "if(" + $3 + ")\n{\n}\n
                  "; }
232             | If OPEN_PARAN booleanExp CLOSE_PARAN OPEN_CURLY
                  empty CLOSE_CURLY Else OPEN_CURLY empty
                  CLOSE_CURLY { $$ = "if(" + $3 + ")\n{\n}\nelse\n
                  {\n}\n"; }
233             | If OPEN_PARAN booleanExp CLOSE_PARAN OPEN_CURLY
                  statements CLOSE_CURLY Else OPEN_CURLY empty
                  CLOSE_CURLY { $$ = "if(" + $3 + ")\n{\n" + $6 +
                  "}\nelse\n{\n}\n"; }
234             | If OPEN_PARAN booleanExp CLOSE_PARAN OPEN_CURLY
                  empty CLOSE_CURLY Else OPEN_CURLY statements
                  CLOSE_CURLY { $$ = "if(" + $3 + ")\n{\n}" + "\
                  nelse\n{\n" + $10 + "}\n"; }
235             ;
236
237 loop: While OPEN_PARAN relationalExp CLOSE_PARAN OPEN_CURLY
      statements CLOSE_CURLY { $$ = "while(" + $3 + ")\n{\n" + $6
      + "}\n"; }
238       | While OPEN_PARAN relationalExp CLOSE_PARAN OPEN_CURLY
            empty CLOSE_CURLY { $$ = "while(" + $3 + ")\n{\n}\n"; }
239       | While OPEN_PARAN booleanExp CLOSE_PARAN OPEN_CURLY
            statements CLOSE_CURLY { $$ = "while(" + $3 + ")\n{\n" +
             $6 + "}\n"; }
240       | While OPEN_PARAN booleanExp CLOSE_PARAN OPEN_CURLY empty
            CLOSE_CURLY { $$ = "while(" + $3 + ")\n{\n}\n"; }
241       ;
242
243 declarations: declarations declaration SEMICOLON{ $$ = $1 + $2
      ; }
244             | declaration SEMICOLON{ $$ = $1; }
245             ;
246
247 declaration: Flt ID  { $$ = "float " + $2 + ";\n";  semantics.
      addVar($2, "float", yyline); }
248             | Int ID  { $$ = "int " + $2 + ";\n";  semantics.
                  addVar($2, "int", yyline); }
249             | Bool ID  { $$ = "boolean " + $2 + ";\n";
                  semantics.addVar($2, "boolean", yyline); }
250             | Str ID  { $$ = "String " + $2 + " = new String()
                  ;\n";  semantics.addVar($2, "String", yyline); }
251             | Flt ID EQUAL FLT { $$ = "float " + $2 + " = " +
                  $4 + ";\n"; semantics.addVar($2, "float", yyline
```

```
                ); }
252             | Int ID EQUAL INT { $$ = "int " + $2 + " = " + $4
                  + ";\n"; semantics.addVar($2, "int", yyline); }
253             | Bool ID EQUAL True { $$ = "boolean " + $2 + " =
                  true;\n"; semantics.addVar($2, "boolean", yyline
                  ); }
254             | Bool ID EQUAL False { $$ = "boolean " + $2 + " =
                   false;\n"; semantics.addVar($2, "boolean",
                  yyline); }
255             | Str ID EQUAL STRING_CONST { $$ = "String " + $2
                  + " = " + $4 + ";\n"; semantics.addVar($2, "
                  String", yyline); }
256             ;
257
258 relationalExp: ID LESSEQUAL constOrVar { $$ = $1 + " <= " + $3
    ; semantics.checkRelExp($1, $3, yyline); semantics.
    checkRelationNumber($1, yyline); }
259             | ID GREATEQUAL constOrVar { $$ = $1 + " >= " +
                  $3; semantics.checkRelExp($1, $3, yyline);
                  semantics.checkRelationNumber($1, yyline); }
260             | ID NOTEQUAL constOrVar { $$ = $1 + " != " + $3;
                   semantics.checkRelExp($1, $3, yyline);
                  semantics.checkRelationNotString($1, yyline); }
261             | ID LESS constOrVar { $$ = $1 + " < " + $3;
                  semantics.checkRelExp($1, $3, yyline);
                  semantics.checkRelationNumber($1, yyline); }
262             | ID GREAT constOrVar { $$ = $1 + " > " + $3;
                  semantics.checkRelExp($1, $3, yyline);
                  semantics.checkRelationNumber($1, yyline); }
263             | ID ISEQUAL constOrVar {    if(semantics.
                  isString($1, yyline)){$$ = $1 + ".equals(" + $3
                   + ")"; semantics.checkRelationString($1,yyline
                  ); }else{$$ = $1 + " == " + $3; semantics.
                  checkRelationNotString($1, yyline); } semantics
                  .checkRelExp($1, $3, yyline); }
264         | OPEN_PARAN relationalExp CLOSE_PARAN { $$ = "(" + $2
              + ")"; }
265             ;
266
267 booleanExp: booleanExp AND booleanExp { $$ = $1 + " && " + $3;
    }
268           | booleanExp OR booleanExp { $$ = $1 + " || " + $3;
                }
269           | relationalExp AND relationalExp { $$ = $1 + " && "
                  + $3; }
```

```
270           | relationalExp OR relationalExp { $$ = $1 + " || "
                + $3; }
271           | relationalExp AND booleanExp { $$ = $1 + " && " +
                $3; }
272           | relationalExp OR booleanExp { $$ = $1 + " || " +
                $3; }
273           | booleanExp AND relationalExp { $$ = $1 + " && " +
                $3; }
274           | booleanExp OR relationalExp { $$ = $1 + " || " +
                $3; }
275           | OPEN_PARAN booleanExp CLOSE_PARAN { $$ = "(" +  $2
                 + ")"; }
276           | NOT booleanExp { $$ = "!" + $2; }
277           | ID { $$ = $1; }
278           | True { $$ = "true"; }
279           | False { $$ = "false"; }
280           ;
281
282 constOrVar: FLT { $$ = "" + $1; }
283           | INT { $$ = "" + $1; }
284           | ID  { $$ = "" + $1; }
285      | STRING_CONST { $$ = $1; }
286           ;
287
288 arithmeticExp: arithmeticExp PLUS arithmeticExp { $$ = $1 + "
      + " + $3 ; semantics.checkForBadAdditionType(yyline);}
289              | arithmeticExp MINUS arithmeticExp { $$ = $1 + "
                   - " + $3; semantics.checkForBadArithmeticType(
                   yyline);}
290              | arithmeticExp MULT arithmeticExp { $$ = $1 + "
                   * " + $3; semantics.checkForBadArithmeticType(
                   yyline);}
291              | arithmeticExp DIV arithmeticExp { $$ = $1 + " /
                    " + $3; semantics.checkForBadArithmeticType(
                   yyline);}
292              | arithmeticExp MOD arithmeticExp { $$ = $1 + " %
                    " + $3; semantics.checkForBadArithmeticType(
                   yyline);}
293              | OPEN_PARAN arithmeticExp CLOSE_PARAN { $$ = "("
                    + $2 + ")"; }
294              | ID  { $$ = $1; semantics.assignmentCheckVar($1,
                    yyline); }
295              | FLT { $$ = "" + $1; semantics.
                   assignmentCheckLeftIsOfType("float", yyline); }
296              | INT { $$ = "" + $1; semantics.
```

```
                assignmentCheckLeftIsOfType("int", yyline); }
297             ;
298
299 assignment: leftSide EQUAL rightSide { $$ = $1 + " = " + $3;
    semantics.funcReturnFlag=false;}
300
301 leftSide: ID { $$ = $1; semantics.assignmentCheckLeft($1,
    yyline); semantics.funcReturnFlag=true;}
302
303 rightSide: arithmeticExp { $$ = $1 + ";\n"; }
304          | functionCall { $$ = $1 + ";\n"; }
305          | STRING_CONST { $$ = $1 + ";\n"; semantics.
               assignmentCheckLeftIsOfType("String", yyline); }
306          | True { $$ = "true" + ";\n"; semantics.
               assignmentCheckLeftIsOfType("boolean", yyline); }
307          | False { $$ = "false" + ";\n"; semantics.
               assignmentCheckLeftIsOfType("boolean", yyline); }
308          ;
309
310 functionCall: functionName OPEN_PARAN parameterList
    CLOSE_PARAN { $$ = $1 + "(" + $3 + ")";  semantics.
    assignmentCheckFunction($1, yyline);}
311             | AddPlayer OPEN_PARAN ID COMMA ID CLOSE_PARAN {
                  $$ = $3 + ".addPlayer(" + $5 + ")" ; semantics.
                  checkIDagainstType($3,"User", yyline);semantics.
                  checkIDagainstType($5,"Player", yyline);}
312             | RemovePlayer OPEN_PARAN ID COMMA ID CLOSE_PARAN
                  { $$ = $3 + ".removePlayer(" + $5 + ")" ;
                  semantics.checkIDagainstType($3,"User", yyline);
                  semantics.checkIDagainstType($5,"Player", yyline
                  ); semantics.assignmentCheckLeftIsOfType("void",
                   yyline); }
313             | ArrayLength OPEN_PARAN ID CLOSE_PARAN { $$ = $3
                  + ".length"; semantics.checkArrayType($3,yyline)
                  ; semantics.assignmentCheckLeftIsOfType("int",
                  yyline); }
314         | GetUserName OPEN_PARAN ID CLOSE_PARAN { $$ = $3 + ".
              getName()"; semantics.checkIDagainstType($3,"User",
              yyline); semantics.assignmentCheckLeftIsOfType("
              String", yyline); }
315         | GetNumPlayers OPEN_PARAN ID CLOSE_PARAN { $$ = $3 +".
              getNumPlayers()";semantics.checkIDagainstType($3,"
              User", yyline); semantics.assignmentCheckLeftIsOfType
              ("int", yyline); }
316         | GetPlayerName OPEN_PARAN ID CLOSE_PARAN { $$ = $3 + "
```

```
            .getName()";semantics.checkIDagainstType($3,"Player",
             yyline); semantics.assignmentCheckLeftIsOfType("
            String", yyline); }
317       | GetPlayerPosition OPEN_PARAN ID CLOSE_PARAN { $$= $3
            + ".getPosition()";semantics.checkIDagainstType($3,"
            Player", yyline); semantics.
            assignmentCheckLeftIsOfType("String", yyline); }
318       | GetPlayerPoints OPEN_PARAN ID CLOSE_PARAN { $$ =$3 +
            ".getPoints()";semantics.checkIDagainstType($3,"
            Player", yyline); semantics.
            assignmentCheckLeftIsOfType("float", yyline); }
319       | Alert OPEN_PARAN STRING_CONST COMMA STRING_CONST
            CLOSE_PARAN {$$ ="GUI.alert("+$3+","+$5+")";
            semantics.assignmentCheckLeftIsOfType("void", yyline)
            ;}
320       | Error OPEN_PARAN STRING_CONST COMMA STRING_CONST
            CLOSE_PARAN {$$ ="GUI.error("+$3+","+$5+")";
            semantics.assignmentCheckLeftIsOfType("void", yyline)
            ;}
              ;
321
322
323 parameterList: parameterList COMMA parameterList { $$ = $1 + "
    , " + $3; }
324               | ID { $$ = $1; }
325               | INT { $$ = "" + $1; }
326               | FLT { $$ = "" + $1; }
327               | STRING_CONST { $$ = $1; }
328               | ID OPEN_SQUARE INT CLOSE_SQUARE { $$ = $1 + "["
                    + $3 + "]"; }
329               | ID OPEN_SQUARE ID CLOSE_SQUARE { $$ = $1 + "["
                   + $3 + "]"; semantics.checkIndex($3, yyline);}
330               | empty { $$ = $1; }
331               ;
332
333 empty: ; { $$ = ""; }
334
335 %%
336
337 /*************************************************
338 * Variables
339 *************************************************/
340 private Yylex lexer;
341 public int yyline = 1;
342 public int yycolumn = 0;
343 public boolean createPositionFile = false;
```

```
344 //Semantic Object
345 Flood_Sem semantics = new Flood_Sem();
346 String scope = "main";
347
348 /*************************************************
349 * generateFloodProgram()
350 *************************************************/
351 public void generateFloodProgram(String definitions, String
      functions)
352 {
353   String classStart = "public class FloodProgram\n{\n";
354   String staticDeclarations = "public static League myLeague;\
        npublic static GUI run;\n";
355   String classEnd = "}\n";
356
357   String main_start = "public static void main(String[] args)\
        n{\n";
358   String main_preEndAutogenerate = "run = new GUI(myLeague);\
        nrun.drawBoard();\n";
359   String main_end = "}\n";
360
361   try
362   {
363     FileWriter writer = new FileWriter(new File("FloodProgram.
          java"));
364     String buffer = classStart + staticDeclarations +
          main_start + definitions + main_preEndAutogenerate +
          main_end + functions + classEnd;
365     writer.write(buffer);
366     writer.close();
367     System.out.println("Compilation successful.");
368   }
369   catch (IOException e)
370   {
371   }
372 }
373
374 /*************************************************
375 * yylex()
376 *************************************************/
377 private int yylex()
378 {
379   int yyl_return = -1;
380
381   try
```

```
382   {
383     yylval = new ParserVal(0);
384     yyl_return = lexer.yylex();
385   }
386   catch (IOException e)
387   {
388     System.err.println("IO error: " + e.getMessage());
389   }
390
391   return yyl_return;
392 }
393
394 /************************************************
395 * Parser()
396 ************************************************/
397 public Parser(Reader r, boolean createFile)
398 {
399   lexer = new Yylex(r, this);
400   this.createPositionFile = createFile;
401 }
402
403 /************************************************
404 * getErrorLocationInfo()
405 ************************************************/
406 public String getErrorLocationInfo(boolean justLine)
407 {
408   if(justLine)
409     return "Error on line(" + yyline + "): ";
410   else
411     return "Error on line(" + yyline + ") and column(" +
          yycolumn + "): ";
412 }
413
414 /************************************************
415 * yyerror()
416 ************************************************/
417 public void yyerror(String error)
418 {
419     try{
420       if(stateptr > 0) {
421         System.out.print("Syntax " + getErrorLocationInfo(true
              ));
422         System.out.println(": Illegal token '" + lexer.yytext
              () + "'");
423       }
```

```
424     }
425     catch(Exception ex){
426     }
427 }
428
429 /*************************************************
430 * main()
431 *************************************************/
432 public static void main(String args[]) throws IOException
433 {
434
435   Parser yyparser;
436   boolean createFile = false;
437
438   if (args.length < 1)
439   {
440     System.out.println("Usage: java Parser <flood_progam.txt>"
        );
441     return;
442   }
443   else if (args.length == 2)
444   {
445     createFile = Boolean.parseBoolean(args[1]);
446   }
447
448   // parse a file
449   yyparser = new Parser(new FileReader(args[0]), createFile);
450
451   System.out.println("\nCompiling ...\n");
452   yyparser.yyparse();
453 }
```

Listing A.3: flood_lex.flex

```
1  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
2  *
3  * flood_lex.flex
4  * FLOOD
5  * Lexical Analyzer
6  *
7  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
8
9  %%
10
11 %byaccj
```

```
12
13 %{
14   private Parser yyparser;
15
16   public Yylex(java.io.Reader r, Parser yyparser)
17   {
18     this(r);
19     this.yyparser = yyparser;
20   }
21 %}
22
23 ID    = [a-zA-Z"_"]([a-zA-Z"_"] | [0-9])*
24 NL    = \n | \r | \r\n
25 WP    = " "
26 INT   = [0-9]+ /*| "-"[0-9]+ */
27 FLT   = [0-9]+ ("." [0-9]+)? /*| "-"[0-9]+ ("." [0-9]+)?*/
28 HT    = \t
29 COMMENTS = "/*" [^*] ~"*/" | "/*" "*" + "/"
30
31 %%
32
33 /* Keywords */
34 DefineLeague      { yyparser.yycolumn += yytext().length();
     return Parser.DefineLeague;     }
35 DefineFunctions   { yyparser.yycolumn += yytext().length();
     return Parser.DefineFunctions;  }
36 LeagueName        { yyparser.yycolumn += yytext().length();
     return Parser.LeagueName;       }
37 MaxUser           { yyparser.yycolumn += yytext().length();
     return Parser.MaxUser;          }
38 MinUser           { yyparser.yycolumn += yytext().length();
     return Parser.MinUser;          }
39 MaxTeamSize       { yyparser.yycolumn += yytext().length();
     return Parser.MaxTeamSize;      }
40 MinTeamSize       { yyparser.yycolumn += yytext().length();
     return Parser.MinTeamSize;      }
41 Set               { yyparser.yycolumn += yytext().length();
     return Parser.Set;              }
42 Add               { yyparser.yycolumn += yytext().length();
     return Parser.Add;              }
43 Action            { yyparser.yycolumn += yytext().length();
     return Parser.Action;           }
44 User              { yyparser.yycolumn += yytext().length();
     return Parser.User;             }
45 Void              { yyparser.yycolumn += yytext().length();
```

```
    return Parser.Void;             }
Int             { yyparser.yycolumn += yytext().length();
    return Parser.Int;              }
Bool            { yyparser.yycolumn += yytext().length();
    return Parser.Bool;             }
Str             { yyparser.yycolumn += yytext().length();
    return Parser.Str;              }
Flt             { yyparser.yycolumn += yytext().length();
    return Parser.Flt;              }
Return          { yyparser.yycolumn += yytext().length();
    return Parser.Return;           }
If              { yyparser.yycolumn += yytext().length();
    return Parser.If;               }
Else            { yyparser.yycolumn += yytext().length();
    return Parser.Else;             }
While           { yyparser.yycolumn += yytext().length();
    return Parser.While;            }
Player          { yyparser.yycolumn += yytext().length();
    return Parser.Player;           }
True            { yyparser.yycolumn += yytext().length();
    return Parser.True;             }
False           { yyparser.yycolumn += yytext().length();
    return Parser.False;            }
RemovePlayer    { yyparser.yycolumn += yytext().length();
    return Parser.RemovePlayer;     }
AddPlayer       { yyparser.yycolumn += yytext().length();
    return Parser.AddPlayer;        }
ArrayLength     { yyparser.yycolumn += yytext().length();
    return Parser.ArrayLength;      }
Alert     { yyparser.yycolumn += yytext().length(); return
    Parser.Alert;           }
Error     { yyparser.yycolumn += yytext().length(); return
    Parser.Error;           }
GetUserName     { yyparser.yycolumn += yytext().length();
    return Parser.GetUserName;      }
GetNumPlayers   { yyparser.yycolumn += yytext().length();
    return Parser.GetNumPlayers;    }
GetPlayerName   { yyparser.yycolumn += yytext().length();
    return Parser.GetPlayerName;    }
GetPlayerPosition { yyparser.yycolumn += yytext().length();
    return Parser.GetPlayerPosition;}
GetPlayerPoints   { yyparser.yycolumn += yytext().length();
    return Parser.GetPlayerPoints;  }

```

```
69
70 /* Comments */
71 {COMMENTS}          { /* ignore */ }
72
73 /* Identifier */
74 {ID}                {
75                       yyparser.yycolumn += yytext().length();
76                       yyparser.yylval = new ParserVal(yytext());
77                       return Parser.ID;
78                     }
79
80 /* Newline */
81 {NL}                {
82                       yyparser.yycolumn = 0;
83                       yyparser.yyline++;
84                     }
85
86 {WP}                {
87                       yyparser.yycolumn++;
88                     }
89
90 /* Integer */
91 {INT}               {
92                       yyparser.yycolumn += yytext().length();
93                       yyparser.yylval = new ParserVal(Integer.
                            parseInt(yytext()));
94                       return Parser.INT;
95                     }
96
97 /* Float */
98 {FLT}               {
99                       yyparser.yycolumn += yytext().length();
100                      yyparser.yylval = new ParserVal(Double.
                            parseDouble(yytext()));
101                      return Parser.FLT;
102                    }
103
104
105 [\t]+               { yyparser.yycolumn += yytext().length(); }
106
107 "\""[^\"]*"\""      {
108                       yyparser.yycolumn += yytext().length();
109                       yyparser.yylval = new ParserVal(yytext());
110                       return Parser.STRING_CONST;
111                    }
```

```
112
113 ,                 { yyparser.yycolumn++; return Parser.COMMA;
                }
114
115 "."               { yyparser.yycolumn++; return Parser.DOT;
                  }
116
117 "+"               { yyparser.yycolumn++; return Parser.PLUS;
                 }
118
119 "{"               { yyparser.yycolumn++; return Parser.
      OPEN_CURLY;      }
120
121 "}"               { yyparser.yycolumn++; return Parser.
      CLOSE_CURLY;     }
122
123 "("               { yyparser.yycolumn++;return Parser.
      OPEN_PARAN;       }
124
125 ")"               { yyparser.yycolumn++;return Parser.
      CLOSE_PARAN;      }
126
127 "=="              { yyparser.yycolumn++; return Parser.ISEQUAL
      ;         }
128
129 "="               { yyparser.yycolumn++; return Parser.EQUAL;
                }
130
131 "!="              { yyparser.yycolumn++; return Parser.
      NOTEQUAL;        }
132
133 "<="              { yyparser.yycolumn++; return Parser.
      LESSEQUAL;       }
134
135 ">="              { yyparser.yycolumn++; return Parser.
      GREATEQUAL;      }
136
137 ">"               { yyparser.yycolumn++; return Parser.GREAT;
                }
138
139 "<"               { yyparser.yycolumn++;return Parser.LESS;
                  }
140
141 "+"               { yyparser.yycolumn++; return Parser.PLUS;
                 }
```

```
142
143 "-"                 { yyparser.yycolumn++; return Parser.MINUS;
                }
144
145 "*"                 { yyparser.yycolumn++; return Parser.MULT;
                 }
146
147 "/"                 { yyparser.yycolumn++; return Parser.DIV;
                  }
148
149 "!"                 { yyparser.yycolumn++; return Parser.NOT;
                  }
150
151 "&&"                { yyparser.yycolumn++; return Parser.AND;
                  }
152
153 "||"                { yyparser.yycolumn++; return Parser.OR;
                   }
154
155 "%"                 { yyparser.yycolumn++; return Parser.MOD;
                  }
156
157 [;]+                {
158                       yyparser.yycolumn += yytext().length();
159                       return Parser.SEMICOLON;
160                     }
161
162 "["                 { yyparser.yycolumn++; return Parser.
      OPEN_SQUARE;     }
163
164 "]"                 { yyparser.yycolumn++; return Parser.
      CLOSE_SQUARE;     }
165
166 /* Error Fallback */
167 [^]                 {
168                       System.err.println("Error: unexpected
                             character '" + yytext() + "'");
169                       return -1;
170                     }
```

Listing A.4: Flood_Sem.java

```
1 /*
2  * Flood_Sem.java
3  * This class handles semantics of the FLOOD language
```

```
4   */
5
6  import java.util.*;
7
8  public class Flood_Sem {
9
10    HashMap<String, Function> functionTable = new HashMap<
         String, Function>();
11    HashMap<String, String> varList = new HashMap<String,
         String>();
12    ArrayList<String> errorList=new ArrayList<String>();
13    String returnProductionType;
14    static boolean debugging = false,validProgram=true,
         funcReturnFlag=false;;
15    boolean draftFunFlag = false;
16    boolean draftPlayFlag = false;
17    boolean tradeFlag = false;
18    boolean dropPlayFlag = false;
19
20
21    //Add Variables
22    LinkedList<String> actionNames = new LinkedList<String>();
23    LinkedList<String> playerNames = new LinkedList<String>();
24    LinkedList<String> userNames = new LinkedList<String>();
25
26    //Assignment Variables
27    String leftSide = "";
28
29    public Flood_Sem(){
30       if(debugging)System.out.print("Starting Semantic Object
            Checker");
31    }
32
33    public String printErrors(){
34       String errors="";
35       for(String s:errorList){
36          errors+=s+"\n";
37       }
38       return errors;
39    }
40
41    /*Adds an Action */
42    public void addAction(String name, int line){
43       if (actionNames.contains(name)){
44          if (debugging){System.out.println(name + " already
```

```
            exists as an Action");}
45          validProgram=false;
46          errorList.add("Error at Line " + line + ": " +name+"
               has already been defined.");
47          return;
48       }
49       actionNames.add(name);
50       if (debugging){System.out.println(name + " was added as
            an Action");}
51       return;
52    }
53
54    /* Add a User */
55    public void addUser(String name, int line){
56       if (userNames.contains(name)){
57          if (debugging){System.out.println(name + " already
               exists as a User");}
58          validProgram=false;
59          errorList.add("Error at Line " + line + ": " +name+"
               has already been defined.");
60          return;
61       }
62       userNames.add(name);
63       if (debugging){System.out.println(name + " was added as
            a User");}
64    }
65
66    /* Add a Player */
67    public void addPlayer(String name, int line){
68       if (playerNames.contains(name)){
69          if (debugging){System.out.println(name + " already
               exists as a Player");}
70          validProgram=false;
71          errorList.add("Error at Line " + line + ": " +name+"
               has already been defined.");
72          return;
73       }
74       playerNames.add(name);
75       if (debugging){System.out.println(name + " was added as
            a Player");}
76    }
77
78    /* Adds a function to the function table. */
79    public void addToFunctionTable(String functionName, String
         returnType, String paramList, int line){
```

```
80          if (functionTable.containsKey(functionName)){
81              if (debugging){System.out.println(functionName + "
                    already exists");}
82              validProgram=false;
83              errorList.add("Error at Line " + line + ": " +
                    functionName+" has already been defined.");
84              return;
85          }
86          functionTable.put(functionName, new Function(
                functionName, returnType, paramList, line));
87          this.varList = new HashMap<String, String>();
88          if (debugging){System.out.println("Reinitializing
                variable list");}
89      }
90
91      /* Checks whether a given value is boolean true or false (
            note: case specific) */
92      public void isBooleanValue(String bool, int line){
93          if (bool.equals("true") || bool.equals("false")){
94              return;
95          }
96          if (debugging){System.out.println(bool + " is not a
                boolean value");}
97          validProgram=false;
98          errorList.add("Error at Line " + line + ": " +bool+" is
                not of type Bool.");
99      }
100
101     /* Adds a variable to the function's variable list. Checks
            if it exists first */
102     public void addVar(String varName, String varType, int line
            ){
103         if (varExists(varName)){
104             if (debugging){System.out.println(varName + " already
                    exists");}
105             validProgram=false;
106             errorList.add("Error at Line " + line + ": " +varName
                    +" has already been defined.");
107             return;
108         }
109         addVarToTable(varName, varType);
110     }
111
112     /* Check whether a variable used exists and is the same as
            its declared type */
```

```
113     public boolean varExists(String varName){
114        if(varList.containsKey(varName)){
115           return true;
116        }
117        else{
118           return false;
119        }
120     }
121
122     /* AddtoCurrentVarList */
123     public void addVarToTable(String varName, String varType){
124        varList.put(varName, varType);
125        if (debugging){System.out.println("Added varName: " +
              varName + ", type: " + varType);}
126     }
127
128     /* Checks the type of a variable against the left side of
           an expression */
129     public void assignmentCheckVar(String right, int line){
130        if (varExists(right)){
131           if (varList.get(right).equals(leftSide)){
132              if (debugging){System.out.println("Both are of
                    type " + leftSide);}
133              return;
134           }
135           if (debugging){System.out.println(right + " isn't of
                 type " + leftSide);}
136           validProgram=false;
137           errorList.add("Error at Line " + line + ": " +right+"
                 is not of type "+leftSide+".");
138           return;
139        }
140        if (debugging){System.out.println(right + " doesn't
              exist");}
141        validProgram=false;
142           errorList.add("Error at Line " + line + ": " +right+"
                 has not been defined.");
143           return;
144     }
145
146     /* Preserves the type of the left side of an expression */
147     public void assignmentCheckLeft(String left, int line){
148        if (varExists(left)){
149           leftSide = varList.get(left);
150           if (debugging){System.out.println("Added " + left);}
```

```
151             return;
152         }
153         validProgram=false;
154         errorList.add("Error at Line " + line + ": " +left+" has
                not been defined.");
155     }
156 
157     /* Check whether an arithmetic expression can be used with
           the kind of ID */
158     public void checkForBadAdditionType(int line){
159         if (leftSide.equals("float") || leftSide.equals("int")
               || leftSide.equals("String")){
160             if (debugging){System.out.println(leftSide + " can be
                    used with add");}
161             return;
162         }
163         if (debugging){System.out.println(leftSide + " cannot be
                used in addition");}
164         validProgram=false;
165         errorList.add("Error at Line " + line + ": " +leftSide+"
                cannot be used in addition.");
166         return;
167     }
168 
169     /* Check whether an arithmetic expression can be used with
           the kind of ID */
170     public void checkForBadArithmeticType(int line){
171         if (leftSide.equals("float") || leftSide.equals("int")){
172             if (debugging){System.out.println(leftSide + " can be
                    used with add/sub/mul/div/mod");}
173             return;
174         }
175         if (debugging){System.out.println(leftSide + " cannot be
                used in an arithmetic expression");}
176         validProgram=false;
177         errorList.add("Error at Line " + line + ": " +leftSide+"
                cannot be used with arithmetic expressions.");
178         return;
179     }
180 
181     /* Checks whether the left side of the expression is a TYPE
            */
182     public void assignmentCheckLeftIsOfType(String type, int
           line){
183         if(funcReturnFlag){
```

```
184             if (leftSide.equals(type)){
185                 if (debugging){System.out.println(leftSide + " IS
                        of type " + type);}
186                 return;
187             }
188             if (debugging){System.out.println(leftSide + " is not
                    of type " + type);}
189             validProgram=false;
190                 errorList.add("Error at Line " + line + ": " +
                        leftSide + " is not of type " + type + ".");
191                 return;
192         }
193         else{
194             if (debugging){System.out.println("Function with no
                        return assignment");}
195             return;
196         }
197     }
198
199     /* Checks the Return Type of the Function Against the Left
           Side of an expression */
200     public void assignmentCheckFunction(String functionName,
           int line){
201         if (functionTable.containsKey(functionName)){
202             if(funcReturnFlag){
203                 if (functionTable.get(functionName).getReturnType
                        ().equals(leftSide)){
204                     if (debugging){System.out.println("Both are of
                            type " + leftSide);}
205                     return;
206                 }
207                 if (debugging){System.out.println(functionName + "
                        doesn't return type " + leftSide);}
208                 validProgram=false;
209                 errorList.add("Error at Line " + line + ": " +
                        functionName + " does not return a value of type
                        " + leftSide + ".");
210                 return;
211             }
212             else{
213                 if (debugging){System.out.println("Function with
                        no return assignment");}
214                 return;
215             }
216         }
```

```
217         if (debugging){System.out.println(functionName + " has
               not been defined.");}
218         validProgram=false;
219         errorList.add("Error at Line " + line + ": " +
               functionName + " has not been defined.");
220         return;
221     }
222
223     public void checkArrayType(String varName, int line){
224         if(varExists(varName)){
225             if(varList.get(varName).equals("User[]") || varList.
                   get(varName).equals("Player[]")){
226                 if (debugging){System.out.println(varName+" is of
                       type "+varList.get(varName));}
227                 return;
228             }
229             if (debugging){System.out.println(varList.get(varName
                   )+" is not a valid array type.");}
230             validProgram=false;
231             errorList.add("Error at Line " + line + ": " +varList
                   .get(varName)+" is not a valid array type.");
232             return;
233         }
234         if (debugging){System.out.println(varName+" has not been
                defined.");}
235         validProgram=false;
236         errorList.add("Error at Line " + line + ": " +varName+"
               has not been defined.");
237         return;
238     }
239     /* Check Divide by zero */
240     public void checkDivideByZero(String var, int line){
241         if (Double.parseDouble(var) == 0){
242             return;
243         }
244         validProgram=false;
245         errorList.add("Error at Line " + line + ": Cannot divide
                by zero.");
246         return;
247     }
248
249     /* Checks against an int */
250     public void checkRelExp(String left, String right, int line
           ){
251         if(right.matches("^-?\\d.*$")){
```

```
252             if(right.contains(".")){
253                 checkRelationalExpAgainstType(left, "float", line)
                        ;
254             }
255             else{
256                 checkRelationalExpAgainstType(left, "int", line);
257             }
258         }
259         else if(right.matches("^\".*\"$")){
260             checkRelationalExpAgainstType(left, "String", line);
261         }
262         else{
263             checkRelationalExp(left, right, line);
264         }
265     }
266 
267     /* Checks against a declared variable */
268     private void checkRelationalExp(String left, String right,
            int line){
269         if (varExists(right)){
270             checkRelationalExpAgainstType(left, varList.get(right
                    ), line);
271             return;
272         }
273         if (debugging){System.out.println(right + " doesn't
                exist");}
274         validProgram=false;
275         errorList.add("Error at Line " + line + ": " +right+"
                has not been defined.");
276         return;
277     }
278 
279     /* Checks to make sure that relational expression don't
            compare an invalid type */
280     public void checkRelationNumber(String left, int line){
281         if (varExists(left)){
282             String leftType = varList.get(left);
283             if (!leftType.equals("int") && !leftType.equals("
                    float")){
284                 if (debugging){System.out.println(left + " cannot
                        be used because it is of type " + leftType);}
285                 validProgram=false;
286                 errorList.add("Error at Line " + line + ": " +left
                         + " cannot be used because it is of type " +
                        leftType + ".");
```

```
287             return;
288          }
289       }
290       else{
291          validProgram=false;
292          errorList.add("Error at Line " + line + ": " +left+"
                has not been defined.");
293          return;
294       }
295    }
296
297    /* Checks to make sure that relational expression don't
          compare an invalid type */
298    public void checkRelationNotString(String left, int line){
299       if (varExists(left)){
300          String leftType = varList.get(left);
301          if (leftType.equals("String")){
302             if (debugging){System.out.println(left + " cannot
                   be used because it is of type " + leftType);}
303             validProgram=false;
304             errorList.add("Error at Line " + line + ": " +left
                    + " cannot be used because it is of type " +
                   leftType + ".");
305             return;
306          }
307       }
308       else{
309          validProgram=false;
310          errorList.add("Error at Line " + line + ": " +left+"
                has not been defined.");
311          return;
312       }
313    }
314
315    /* Checking known type of string */
316    public void checkRelationString(String left, int line){
317       if (varExists(left)){
318          String leftType = varList.get(left);
319          if (!leftType.equals("String")){
320             if (debugging){System.out.println(left + " cannot
                   be used because it is of type " + leftType);}
321             validProgram=false;
322             errorList.add("Error at Line " + line + ": " +left
                    + " cannot be used because it is of type " +
                   leftType + ".");
```

```
323             return;
324          }
325       }
326       else{
327          validProgram=false;
328          errorList.add("Error at Line " + line + ": " +left+"
               has not been defined.");
329          return;
330       }
331    }
332
333    public boolean isString(String left, int line){
334       if (varExists(left)){
335          String leftType = varList.get(left);
336          if (leftType.equals("String")){
337             return true;
338          }
339          return false;
340       }
341       else{
342          validProgram=false;
343          errorList.add("Error at Line " + line + ": " +left+"
               has not been defined.");
344          return false;
345       }
346    }
347
348    /* Private method for checking a variable against an
          unknown type */
349    private void checkRelationalExpAgainstType(String left,
          String rightType, int line){
350       if (varExists(left)){
351          String leftType = varList.get(left);
352          if (leftType.equals(rightType)){
353             if (debugging){System.out.println("Both are of
                  type " + rightType);}
354             return;
355          }
356          if (debugging){System.out.println(rightType + " is
               not " + leftType);}
357          validProgram=false;
358          errorList.add("Error at Line " + line + ": " +
               rightType + " is not of type " + leftType + ".");
359          return;
360       }
```

```
361         else{
362             if (debugging){System.out.println(left + " doesn't
                exist");}
363             validProgram=false;
364             errorList.add("Error at Line " + line + ": " +left +
                " has not been defined.");
365             return;
366         }
367     }
368
369     //check that array index is an int
370     public void checkIndex(String arrayIndex, int line){
371         //check if ID is a variable of type int
372         arrayIndex = arrayIndex.replaceAll(" ", "");
373         if(varList.containsKey(arrayIndex) && varList.get(
              arrayIndex).equals("int")){
374             if(debugging){System.out.println(arrayIndex +"\n"+
                varList.get(arrayIndex));}
375             return;
376         }
377         else{
378             if (debugging){System.out.println("Fail, invalid type
                ");}
379             validProgram=false;
380             errorList.add("Error at Line " + line + ": " +
                arrayIndex+" is an invalid type for an array index.
                ");
381             return;
382         }
383     }
384
385     public String getType(String id){
386         if(varExists(id))
387             return varList.get(id);
388         if(debugging){System.out.println(id+" is a valid
              variable!");}
389         return null;
390     }
391     public void setReturnProdType(String type){
392         returnProductionType=type;
393     }
394     public void checkReturnTypeMatch(String returnType, int
           line){
395         if(returnProductionType!=null){
396             if(returnProductionType.equals(returnType)){
```

```
397             if (debugging){System.out.println(
                   returnProductionType + " return type matches "+
                   returnType);}
398             return;
399          }
400          if (debugging){System.out.println(
                returnProductionType + " return type doesn't match
                "+returnType);}
401          validProgram=false;
402          errorList.add("Error at Line " + line + ": " +
                returnProductionType + " return type does not match
                 "+returnType+".");
403          return;
404       }
405       if (debugging){System.out.println(returnProductionType +
              " is not a valid type");}
406       validProgram=false;
407       errorList.add("Error at Line " + line + ": " +
             returnProductionType + " is not a valid type.");
408       return;
409    }
410
411    /* Check that an ID is of a certain type */
412    public void checkIDagainstType(String id, String type, int
          line){
413       if (varExists(id)){
414          String idType = varList.get(id);
415          if (idType.equals(type)){
416             if (debugging){System.out.println(id + " IS of
                   type " + type);}
417             return;
418          }
419          if (debugging){System.out.println(id + " is not " +
                type);}
420          validProgram=false;
421          errorList.add("Error at Line " + line + ": " +id + "
                is not of type " + type + ".");
422          return;
423       }
424       if (debugging){System.out.println(id + " doesn't exist")
             ;}
425       validProgram=false;
426       errorList.add("Error at Line " + line + ": " +id + " has
              not been defined.");
427       return;
```

```
428     }
429
430     //set flags for required functions
431     public String setFlags(){
432        //flag for draftFunction
433        Function fun;
434        if(functionTable.containsKey("draftFunction")){
435           fun = functionTable.get("draftFunction");
436           if(fun.returnType.equals("int")){
437              if(fun.paramTypeList.length == 1 && fun.
                    paramTypeList[0].equals("int")){
438                 if(debugging)System.out.print("Draft function
                       found");
439                 draftFunFlag = true;
440              }
441           }
442        }
443        if(functionTable.containsKey("draftPlayer")){
444           fun = functionTable.get("draftPlayer");
445           if(fun.returnType.equals("boolean")){
446              if(fun.paramTypeList.length==2 && fun.
                    paramTypeList[0].equals("User")  && fun.
                    paramTypeList[1].equals("Player")){
447                 if(debugging)System.out.print("Draft player
                       found");
448                 draftPlayFlag = true;
449              }
450           }
451        }
452        if(functionTable.containsKey("trade")){
453           fun = functionTable.get("trade");
454           if(fun.returnType.equals("boolean")){
455              if(fun.paramTypeList.length==4 && fun.
                    paramTypeList[0].equals("User")
456                       && fun.paramTypeList[1].equals("Player[]")
                          && fun.paramTypeList[2].equals("User")
457                       && fun.paramTypeList[3].equals("Player[]")){
458                 if(debugging)System.out.print("Trade found");
459                 tradeFlag = true;
460              }
461           }
462        }
463        if(functionTable.containsKey("dropPlayer")){
464           fun = functionTable.get("dropPlayer");
465           if(fun.returnType.equals("boolean")){
```

```
466             if(fun.paramTypeList.length==2 && fun.
                   paramTypeList[0].equals("User")
467                       && fun.paramTypeList[1].equals("Player")){
468                 if(debugging)System.out.print("Drop player
                       found");
469                 dropPlayFlag = true;
470             }
471         }
472     }
473     return writeDefaultFuns();
474 }
475
476 //write default functions if needed
477 public String writeDefaultFuns(){
478     String functions = "";
479     if(!draftFunFlag){
480         //write draft function
481         functions += "public static int draftFunction(int
                turn){\nreturn turn%myLeague.getCurrentNumUsers();\
                n}\n";
482     }
483     if(!draftPlayFlag){
484         //write draft player
485         functions += "public static boolean draftPlayer(User
                u, Player p){\nu.addPlayer(p);\nreturn true;\n}\n";
486     }
487     if(!tradeFlag){
488         //write trade
489         functions += "public static boolean trade(User u1,
                Player[] p1, User u2, Player[] p2){\n"+
490                 "int i,j;\n"+
491                 "boolean flag2=true;\n"+
492                 "i=0;\n"+
493                 "while(i<p1.length){\n"+
494                     "flag2=dropPlayer(u1,p1[i]);\n"+
495                     "if(!flag2){    //If the drop was
                            unsuccessful\n"+
496                         "j=i;\n"+
497                         "while(j>=0){    //Add p1 back to u1\n
                                "+
498                             "draftPlayer(u1,p1[j]);\n"+
499                             "j--;\n"+
500                         "}\n"+
501                         "return false;\n"+
502                     "}\n"+
```

```
503                    "i++;\n"+
504                 "}\n"+
505                 "i=0;\n"+
506                 "while(i<p2.length){\n"+
507                    "flag2=dropPlayer(u2,p2[i]);\n"+
508                    "if(!flag2){    //If the drop was
                          unsuccessful\n"+
509                       "j=i;\n"+
510                       "while(j>=0){    //Add p2 back to u2\n
                             "+
511                          "draftPlayer(u2,p2[j]);\n"+
512                          "j--;\n"+
513                       "}\n"+
514                       "j=0;\n"+
515                       "while(j<p1.length){    //Add p1 to u1
                             \n"+
516                          "draftPlayer(u1,p1[j]);\n"+
517                          "j++;\n"+
518                       "}\n"+
519                       "return false;\n"+
520                    "}\n"+
521                    "i++;\n"+
522                 "}\n"+
523                 "i=0;\n"+
524                 "while(i<p1.length){\n"+
525                    "flag2=draftPlayer(u2,p1[i]);\n"+
526                    "if(!flag2){    //If draft was
                          unsuccessful\n"+
527                       "j=i;\n"+
528                       "while(j>=0){    //Remove p1 from u2\n
                             "+
529                          "dropPlayer(u2,p1[j]);\n"+
530                          "j--;\n"+
531                       "}\n"+
532                       "j=0;\n"+
533                       "while(j<p1.length){    //Add p1 to u1
                             \n"+
534                          "draftPlayer(u1,p1[j]);\n"+
535                          "j++;\n"+
536                       "}\n"+
537                       "j=0;\n"+
538                       "while(j<p2.length){    //Add p2 to u2
                             \n"+
539                          "draftPlayer(u2,p2[j]);\n"+
540                          "j++;\n"+
```

```
                        "}\n"+
                        "return false;\n"+
                     "}\n"+
                     "i++;\n"+
                  "}\n"+
                  "i=0;\n"+
                  "while(i<p2.length){\n"+
                     "flag2=draftPlayer(u1,p2[i]);\n"+
                     "if(!flag2){    //If the drop was
                        unsuccessful\n"+
                        "j=i;\n"+
                        "while(j>=0){    //Remove p2 from u1\n
                           "+
                            "dropPlayer(u1,p2[j]);\n"+
                            "j--;\n"+
                        "}\n"+
                        "j=0;\n"+
                        "while(j<p1.length){    //Remove p1
                           from u2\n"+
                            "dropPlayer(u2,p1[j]);\n"+
                            "j++;\n"+
                        "}\n"+
                        "j=0;\n"+
                        "while(j<p1.length){    //Add p1 to u1
                           \n"+
                            "draftPlayer(u1,p1[j]);\n"+
                            "j++;\n"+
                        "}\n"+
                        "j=0;\n"+
                        "while(j<p2.length){    //Add p2 to u2
                           \n"+
                            "draftPlayer(u2,p2[j]);\n"+
                            "j++;\n"+
                        "}\n"+
                        "return false;\n"+
                     "}\n"+
                     "i++;\n"+
                  "}\n"+
                  "return true;\n"+
               "}\n";
      }
      if(!dropPlayFlag){
         //write drop player
         functions += "public static boolean dropPlayer(User u
            , Player p){\nu.removePlayer(p);\nreturn true;\n}\n
```

```
                ";
580         }
581         return functions;
582     }
583 }
```

Listing A.5: Function.java

```
1  /*
2   * Function.java
3   * This class supports the FLOOD semantics on functions
4   */
5
6  import java.util.HashMap;
7  import java.util.Arrays;
8
9  public class Function {
10
11    String functionName;
12    String[] paramTypeList;
13    HashMap<String, String> argsList = new HashMap<String,
         String>(); //name type
14    String returnType;
15    int lineNumber;
16    static boolean debugging = false;
17
18    /* Constructor sets a function's name, returnType and
         parameters in the instance variables of the function */
19    public Function(String functionName, String returnType,
         String paramList, int lineNumber){
20       this.functionName = functionName;
21       if (debugging){System.out.println("**Initializing " +
            functionName + " function**");}
22       this.returnType = returnType;
23       if (debugging){System.out.println(functionName + "=
            returnType: " + returnType);}
24       this.lineNumber = lineNumber;
25       if (!paramList.trim().isEmpty()){
26          String[] params = paramList.trim().split("\\s*,\\s*")
               ;
27          paramTypeList = new String[params.length];
28          for(int i=0; i<params.length; i++){
29             //split
30             String[] temp = params[i].split("\\s+");
31             argsList.put(temp[1], temp[0]); //This is reversed
```

```
                    in the argument list so reversing it back here
32              paramTypeList[i]=temp[0];
33              if (debugging) {System.out.println(functionName +
                   "= argName: " + temp[1] + ", type: " + temp[0])
                   ;}
34           }
35        }
36        else{
37           paramTypeList = new String[0];
38        }
39     }
40
41     public String getReturnType() {
42        return returnType;
43     }
44  }
```

Listing A.6: GUI.java

```
1  /*
2   * GUI.java
3   * This class the GUI for the fantasy league
4   */
5
6  import java.awt.BorderLayout;
7  import java.awt.Color;
8  import java.awt.Dimension;
9  import java.awt.event.ActionListener;
10 import java.awt.event.ActionEvent;
11 import java.io.File;
12 import java.text.DecimalFormat;
13
14 import javax.swing.JFrame;
15 import javax.swing.JOptionPane;
16 import javax.swing.JToolBar;
17 import javax.swing.JButton;
18 import javax.swing.JFileChooser;
19 import javax.swing.JLabel;
20 import javax.swing.JScrollPane;
21 import javax.swing.JTable;
22 import javax.swing.ListSelectionModel;
23 import javax.swing.JTabbedPane;
24 import javax.swing.JComboBox;
25 import javax.swing.JSplitPane;
26 import javax.swing.event.ChangeEvent;
```

```
27 import javax.swing.event.ChangeListener;
28 import javax.swing.table.DefaultTableModel;
29
30
31 public class GUI {
32    private League theLeague;
33    private static JFrame frmFloodFantasyLeague;
34    private MyTableModel homeTable,draftTable,tradeTable_1,
         tradeTable_2,dropTable,ruleTable;
35    private DefaultTableModel homeModel,draftModel,tradeModel_1
         ,tradeModel_2,dropModel,ruleModel;
36    private JLabel draftLabel;
37    private int currentTurn,pick;
38    private DecimalFormat twoDForm;
39    private final String[] homeHeader={"Rank","Team","Points"},
40       playerInfoHeader={"Player","Position","Points All Season
            "},
41       ruleHeader={"Action","Point Value"};
42
43    /**Constructor
44     *
45     * @param League game
46     */
47    public GUI(League game){
48       this.theLeague=game;
49       twoDForm = new DecimalFormat("0.00");
50       currentTurn=0;
51    }
52
53    /**Populates the home table assuming it has already been
         initialized.
54     *
55     */
56    private void populateHome(){
57       User[] rankedTeams= theLeague.getRankedUsers(); //Get
            all the teams in reverse ranked order
58       while(homeModel.getRowCount()>0) //Remove all the rows
            from the home table
59          homeModel.removeRow(0);
60       String[] tempHome=new String[3]; //Initialize a
            temporary row
61       for(int i=rankedTeams.length-1,j=1;i>=0;i--,j++){  //
            Iterate through the teams
62          tempHome[0]=Integer.toString(j); //Set the rank
63          tempHome[1]=rankedTeams[i].getName();  //Set the name
```

```
64          tempHome[2]=twoDForm.format(rankedTeams[i].getPoints
               ()); //Set the points
65          homeModel.addRow(tempHome);   //Add the row
66       }
67    }
68
69    /**Populates the add table assuming it has already been
         initialized.
70     *
71     */
72    private void populateDraft(){
73       pick=FloodProgram.draftFunction(currentTurn);   //
            Determine who is picking next
74       draftLabel.setText(theLeague.getUser(pick).getName()+"'s
             turn!"); //Figure out which user is picking next
75       Player[] rankedPlayers=theLeague.
            getRankedAvailablePlayers();  //Get all the players in
             reverse ranked order
76       while(draftModel.getRowCount()>0)   //Remove all the
            rows from the add table
77          draftModel.removeRow(0);
78       String[] tempDraft=new String[3];   //Initialize a
            temporary row
79       for(int i=rankedPlayers.length-1;i>=0;i--){  //Iterate
            through the players
80          tempDraft[0]=rankedPlayers[i].getName();  //Set the
               name
81          tempDraft[1]=rankedPlayers[i].getPosition(); //Set
               the position
82          tempDraft[2]=twoDForm.format(rankedPlayers[i].
               getPoints()); //Set the points scored all season
83          draftModel.addRow(tempDraft); //Add the row
84       }
85    }
86
87    /**Populates the add table assuming it has already been
         initialized.
88     *
89     */
90    private void populateRules(){
91       Action[] rules=theLeague.getActions(); //Get all the
            players in reverse ranked order
92       while(ruleModel.getRowCount()>0) //Remove all the rows
            from the add table
93          ruleModel.removeRow(0);
```

```
94          String[] tempRule=new String[2]; //Initialize a
               temporary row
95          for(int i=rules.length-1;i>=0;i--){ //Iterate through
               the players
96             tempRule[0]=rules[i].getAction();   //Set the name
97             tempRule[1]=twoDForm.format(rules[i].getPoints()); //
                  Set the points scored all season
98             ruleModel.addRow(tempRule);   //Add the row
99          }
100      }
101
102      /**Display an error window with the title and message given
103       * as parameters.
104       *
105       * @param String title
106       * @param String message
107       */
108      public static void error(String title,String message){
109         JOptionPane.showMessageDialog(frmFloodFantasyLeague,
               message,title,JOptionPane.ERROR_MESSAGE);
110      }
111
112      /**Show a warning window with the title and message given
113       * as parameters
114       *
115       * @param String title
116       * @param String message
117       */
118      public static void alert(String title, String message) {
119         JOptionPane.showMessageDialog(frmFloodFantasyLeague,
               message,title,JOptionPane.WARNING_MESSAGE);
120      }
121
122      /**
123       * Initialize the contents of the frame.
124       */
125      public void drawBoard() {
126         //Set up the frame
127         frmFloodFantasyLeague = new JFrame();
128         frmFloodFantasyLeague.setBackground(new Color(0, 0, 205)
               );
129         frmFloodFantasyLeague.setTitle("FLOOD Fantasy League: "+
               theLeague.getName());
130         frmFloodFantasyLeague.setBounds(100, 100, 450, 300);
131         frmFloodFantasyLeague.setDefaultCloseOperation(JFrame.
```

```
            EXIT_ON_CLOSE);
132         frmFloodFantasyLeague.setSize(new Dimension(700, 400));
133         frmFloodFantasyLeague.setVisible(true);
134
135         //Initialize and add the tabbed pane
136         final JTabbedPane tabbedPane = new JTabbedPane(
            JTabbedPane.TOP);
137         frmFloodFantasyLeague.getContentPane().add(tabbedPane,
            BorderLayout.CENTER);
138
139         //Initialize the home tab
140         JSplitPane homeSplitPane = new JSplitPane();
141         homeSplitPane.setResizeWeight(0.99);
142         homeSplitPane.setOrientation(JSplitPane.VERTICAL_SPLIT);
143
144         //Initialize the home tab toolbar
145         JToolBar homeToolBar = new JToolBar();
146         homeToolBar.setFloatable(false);
147         homeSplitPane.setRightComponent(homeToolBar);
148
149         //Add the upload stats button to the toolbar
150         JButton uploadStatsButton = new JButton("Upload Stat
            File");
151         uploadStatsButton.setMaximumSize(new Dimension(32767,
            32767));
152         homeToolBar.add(uploadStatsButton);
153
154         //Add the create dump button to the toolbar
155         JButton createDumpButton = new JButton("Create Dump File
            ");
156         createDumpButton.setMaximumSize(new Dimension(32767,
            32767));
157         homeToolBar.add(createDumpButton);
158
159         //Add the import dump button to the toolbar
160         JButton importDumpButton = new JButton("Import Dump File
            ");
161         importDumpButton.setMaximumSize(new Dimension(32767,
            32767));
162         homeToolBar.add(importDumpButton);
163
164         //Initialize the home tab scrollpane
165         JScrollPane homePane = new JScrollPane();
166         homeSplitPane.setLeftComponent(homePane);
167
```

```
168         //Add the home tab to the tabbed pane
169         tabbedPane.addTab("Home", null, homeSplitPane, null);
170 
171         //Initialize, format and add the home table
172         homeModel = new DefaultTableModel(homeHeader,0);   //Add
                the header but no rows
173         homeTable = new MyTableModel(homeModel);
174         homeTable.setEnabled(false);  //Make the rows
               unselectable
175         homeTable.setAutoCreateRowSorter(true);   //Allow
               sorting
176         homePane.setViewportView(homeTable);   //Put the table
               into the scroll pane
177 
178         populateHome();   //Populate the home table
179 
180         JSplitPane draftSplitPane = new JSplitPane();   //Add
               the draft split pane
181         draftSplitPane.setOrientation(JSplitPane.VERTICAL_SPLIT)
               ;   //Split vertically
182         tabbedPane.addTab("Draft", null, draftSplitPane, null);
                 //Add the split pane to the tabbed pane
183 
184         //Initialize and add the draft scroll pane
185         JScrollPane draftScrollPane = new JScrollPane();
186         draftSplitPane.setRightComponent(draftScrollPane);
187 
188         //Initialize, format and add the draft table
189         draftModel = new DefaultTableModel(playerInfoHeader,0);
                 //Add the header but no rows
190         draftTable = new MyTableModel(draftModel);
191         draftTable.setRowSelectionAllowed(true);  //Allow row
               selection
192         draftTable.setSelectionMode(ListSelectionModel.
               SINGLE_SELECTION); //Allow only one row selection at a
                time
193         draftTable.setAutoCreateRowSorter(true);  //Allow
               sorting
194         draftScrollPane.setViewportView(draftTable); //Put the
               table into the scroll pane
195 
196         //Initialize, format and add the draft tool bar
197         JToolBar draftToolBar = new JToolBar();
198         draftToolBar.setFloatable(false);
199         draftSplitPane.setLeftComponent(draftToolBar);
```

```
200
201         //Initialize, format and add the draft label
202         draftLabel = new JLabel();
203         draftLabel.setMinimumSize(new Dimension(600, 15));
204         draftLabel.setMaximumSize(new Dimension(32767, 15));
205         draftToolBar.add(draftLabel);
206
207         //Initialize,format and add the draft button
208         JButton btnDraft = new JButton("Draft");
209         btnDraft.setPreferredSize(new Dimension(100, 25));
210         btnDraft.setMaximumSize(new Dimension(100, 25));
211         btnDraft.setMinimumSize(new Dimension(100, 25));
212         draftToolBar.add(btnDraft);
213
214         //Initialize, format and add the trade split pane
215         JSplitPane tradeSplitPane = new JSplitPane();
216         tradeSplitPane.setOrientation(JSplitPane.VERTICAL_SPLIT)
               ;   //Split vertically
217         tabbedPane.addTab("Trade", null, tradeSplitPane, null);
218
219         //Initialize, format and add the trade tool bar
220         JToolBar tradeToolBar = new JToolBar();
221         tradeToolBar.setFloatable(false);
222         tradeSplitPane.setLeftComponent(tradeToolBar);
223
224         //Initialize, format and add the first trade combo box
225         final JComboBox tradeComboBox_1 = new JComboBox();
226         tradeToolBar.add(tradeComboBox_1);
227
228         //Initialize, format and add the second trade combo bos
229         final JComboBox tradeComboBox_2 = new JComboBox();
230         tradeToolBar.add(tradeComboBox_2);
231
232         //Initialize and add the trade button
233         JButton btnTrade = new JButton("Trade");
234         tradeToolBar.add(btnTrade);
235
236         //Initialize, format and add the second trade split pane
237         JSplitPane tradeSplitPane_2 = new JSplitPane();
238         tradeSplitPane_2.setResizeWeight(0.5);
239         tradeSplitPane.setRightComponent(tradeSplitPane_2);
240
241         //Initialize and add the left trade scroll pane
242         JScrollPane tradeScrollPane_1 = new JScrollPane();
243         tradeSplitPane_2.setLeftComponent(tradeScrollPane_1);
```

```
244
245         //Initialize, format and add the left trade table
246         tradeModel_1 = new DefaultTableModel(playerInfoHeader,0)
               ;   //Add the header but no rows
247         tradeTable_1 = new MyTableModel(tradeModel_1);
248         tradeTable_1.setRowSelectionAllowed(true);   //Set
               selection of entire rows
249         tradeTable_1.setSelectionMode(ListSelectionModel.
               MULTIPLE_INTERVAL_SELECTION);   //Allow multiple row
               selection
250         tradeTable_1.setAutoCreateRowSorter(true);   //Allow
               sorting
251         tradeScrollPane_1.setViewportView(tradeTable_1);   //Add
                table to scroll pane
252
253         //Initialize and add the right trade scroll pane
254         JScrollPane tradeScrollPane_2 = new JScrollPane();
255         tradeSplitPane_2.setRightComponent(tradeScrollPane_2);
256
257         //Initialize, format and add the right trade table
258         tradeModel_2 = new DefaultTableModel(playerInfoHeader,0)
               ;   //Add the header but no rows
259         tradeTable_2 = new MyTableModel(tradeModel_2);
260         tradeTable_2.setRowSelectionAllowed(true);   //Set
               selection of entire rows
261         tradeTable_2.setSelectionMode(ListSelectionModel.
               MULTIPLE_INTERVAL_SELECTION);   //Allow multiple row
               selection
262         tradeTable_2.setAutoCreateRowSorter(true);   //Allow
               sorting
263         tradeScrollPane_2.setViewportView(tradeTable_2);   //Add
                table to scroll pane
264
265         //Initialize, format and add the drop split pane
266         JSplitPane dropSplitPane = new JSplitPane();
267         dropSplitPane.setOrientation(JSplitPane.VERTICAL_SPLIT);
                //Split vertically
268         tabbedPane.addTab("Drop", null, dropSplitPane, null);
269
270         //Initialize, format and add the drop toolbar
271         JToolBar dropToolBar = new JToolBar();
272         dropToolBar.setFloatable(false);
273         dropSplitPane.setLeftComponent(dropToolBar);
274
275         //Initialize and add the drop combo box
```

```
276         final JComboBox dropComboBox = new JComboBox();
277         dropToolBar.add(dropComboBox);
278 
279         //Initialize and add the drop button
280         JButton btnDrop = new JButton("Drop");
281         dropToolBar.add(btnDrop);
282 
283         //Initialize and add the drop scroll pane
284         JScrollPane dropScrollPane = new JScrollPane();
285         dropSplitPane.setRightComponent(dropScrollPane);
286 
287         //Initialize, format and add the drop table
288         dropModel = new DefaultTableModel(playerInfoHeader,0);
                //Add the header but no rows
289         dropTable = new MyTableModel(dropModel);
290         dropTable.setSelectionMode(ListSelectionModel.
              SINGLE_SELECTION);  //Alow single selection
291         dropTable.setAutoCreateRowSorter(true);   //Allow
              sorting
292         dropScrollPane.setViewportView(dropTable);   //Add the
              table to the scroll pane
293 
294         //Initialize the home tab scrollpane
295         JScrollPane rulePane = new JScrollPane();
296 
297         //Add the home tab to the tabbed pane
298         tabbedPane.addTab("Rules", null, rulePane, null);
299 
300         //Initialize, format and add the home table
301         ruleModel = new DefaultTableModel(ruleHeader,0);   //Add
               the header but no rows
302         ruleTable = new MyTableModel(ruleModel);
303         ruleTable.setEnabled(false);  //Make the rows
              unselectable
304         ruleTable.setAutoCreateRowSorter(true);   //Allow
              sorting
305         rulePane.setViewportView(ruleTable);   //Put the table
              into the scroll pane
306 
307         populateRules();  //Populate the home table
308 
309 
310         //Initialize the file chooser
311         final JFileChooser chooser=new JFileChooser();
312 
```

```
313         //Populate both trade and the drop combo boxes
314         User[] rankedTeams= theLeague.getRankedUsers();
315         for(int i=0;i<rankedTeams.length;i++){
316            tradeComboBox_1.insertItemAt(rankedTeams[i].getName()
                  ,i);
317            tradeComboBox_2.insertItemAt(rankedTeams[i].getName()
                  ,i);
318            dropComboBox.insertItemAt(rankedTeams[i].getName(),i)
                  ;
319         }
320
321         //*****************************************************
322         //*****************Action Listeners********************
323         //*****************************************************
324
325         //Action listener for changing tabs
326         tabbedPane.addChangeListener(new ChangeListener() {
327            public void stateChanged(ChangeEvent e) {
328               int selection = tabbedPane.getSelectedIndex();  //
                     Get selected tab
329               switch(selection){
330               case 0:  //Populate home table
331                  populateHome();
332                  break;
333               case 1:  //Populate draft table
334                  populateDraft();
335                  break;
336               case 4:
337                  populateRules();
338                  break;
339               }
340            }
341         });
342
343         //Stats upload action listener
344         uploadStatsButton.addActionListener(new ActionListener()
               {
345            public void actionPerformed(ActionEvent arg0) {
346               int result = chooser.showOpenDialog(null);   //
                     Determine what the user pressed
347               switch (result) {
348               case JFileChooser.APPROVE_OPTION:   //Opened file
349                  File file=chooser.getSelectedFile();   //Get
                        the chosen file
350                  IOManager.uploadStats(theLeague,file.
```

```
                    getAbsolutePath()); //Pass the file path to
                    the parser method
351                 populateHome();
352                 break;
353             case JFileChooser.CANCEL_OPTION: //Canceled
354                 break;
355             case JFileChooser.ERROR_OPTION:  //Generated an
                    error
356                 GUI.error("Upload Error!","Sorry, there was an
                        error opening the stat file.");
357                 break;
358             }
359         }
360     });
361
362     //Dump generator action listener
363     createDumpButton.addActionListener(new ActionListener()
            {
364         public void actionPerformed(ActionEvent e) {
365             chooser.setSelectedFile(new File("flooddmp.txt"));
366             int result = chooser.showSaveDialog(null);
367             switch (result) {
368             case JFileChooser.APPROVE_OPTION:   //Opened file
369                 File file=chooser.getSelectedFile();   //Get
                        the chosen file
370                 if(file.exists()) {
371                         int overwrite = JOptionPane.
                            showConfirmDialog(
                            frmFloodFantasyLeague, "Do you want to
                             overwrite " + file.getName());
372                         if(overwrite == JOptionPane.YES_OPTION)
                            {
373                           IOManager.writeState(theLeague,file.
                              getAbsolutePath(),currentTurn);  //
                              Pass the file path to the parser
                              method
374                           tabbedPane.setSelectedIndex(0);
375                         }
376                 }
377                 else{
378                     IOManager.writeState(theLeague,file.
                        getAbsolutePath(),currentTurn);  //Pass
                        the file path to the parser method
379                     tabbedPane.setSelectedIndex(0);
380                 }
```

```
381                 break;
382             case JFileChooser.CANCEL_OPTION: //Canceled
383                 break;
384             case JFileChooser.ERROR_OPTION:  //Generated an
                    error
385                 GUI.error("Upload Error!","Sorry, error
                        creating the dump file.");
386                 break;
387             }
388         }
389     });
390
391     //Dump importer action listener
392     importDumpButton.addActionListener(new ActionListener()
            {
393         public void actionPerformed(ActionEvent arg0) {
394             int result = chooser.showOpenDialog(null);   //
                    Determine what the user pressed
395             switch (result) {
396             case JFileChooser.APPROVE_OPTION:   //Opened file
397                 File file=chooser.getSelectedFile();   //Get
                        the chosen file
398                 int temp=IOManager.importState(theLeague,file.
                        getAbsolutePath()); //Pass the file path to
                        the parser method
399                 if(temp!=-1){
400                     currentTurn=temp;
401                     populateDraft();
402                 }
403                 populateHome();
404                 break;
405             case JFileChooser.CANCEL_OPTION: //Canceled
406                 break;
407             case JFileChooser.ERROR_OPTION:  //Generated an
                    error
408                 GUI.error("Upload Error!","Sorry, there was an
                        error opening the stat file.");
409                 break;
410             }
411         }
412     });
413
414     //Determine which user is picking first
415     pick=FloodProgram.draftFunction(currentTurn);   //Gets
            the number representing the user's turn
```

```
416         draftLabel.setText(theLeague.getUser(pick).getName()+"'s
                turn!"); //Puts the user's name in the label
417
418         //Draft action listener
419         btnDraft.addActionListener(new ActionListener() {
420            public void actionPerformed(ActionEvent arg0) {
421               for(int i=0;i<draftModel.getRowCount();i++){ //
                     Iterate through table entries
422                  if(draftTable.isCellSelected(i,0)){ //If it's
                        selected
423                     if(!FloodProgram.draftPlayer(theLeague.
                        getUser(pick),League.athletes.get(
                        draftModel.getValueAt(i,0)))){  //If the
                        draft isn't successful
424                        GUI.error("Invalid draft!","Sorry, your
                           draft violates rules of the league.");
425                        return;
426                     }
427                     int overwrite = JOptionPane.
                        showConfirmDialog(frmFloodFantasyLeague, "
                        Are you sure you want to draft: " + League
                        .athletes.get(draftModel.getValueAt(i,0)).
                        getName());
428                      if(overwrite == JOptionPane.YES_OPTION)
                           {
429                        currentTurn++; //Increment the turn
430                        draftModel.removeRow(i);   //Remove that
                           row from the draft table
431                        populateDraft();
432                        //Make all the combo boxes not select
                           anything
433                        tradeComboBox_1.setSelectedIndex(-1);
434                        tradeComboBox_2.setSelectedIndex(-1);
435                        dropComboBox.setSelectedIndex(-1);
436                        //Remove the current tables in all the
                           other tabs so that they are up to date
437                        while(tradeModel_1.getRowCount()>0){
438                           tradeModel_1.removeRow(0);
439                        }
440                        while(tradeModel_2.getRowCount()>0){
441                           tradeModel_2.removeRow(0);
442                        }
443                        while(dropModel.getRowCount()>0){
444                           dropModel.removeRow(0);
445                        }
```

```
446                     return;
447                      }
448                      return;
449                 }
450               }
451               //If no selection is found
452               GUI.error("Add error!","Sorry, no player was
                     selected!");
453            }
454         });
455
456         //Action listener for left trade combo box
457         tradeComboBox_1.addActionListener (new ActionListener ()
              {
458               public void actionPerformed(ActionEvent e) {
459                 if(tradeComboBox_1.getSelectedIndex()!=-1){  //If
                     a user is selected
460                    Player[] teamPlayers=League.teams.get(
                       tradeComboBox_1.getSelectedItem()).getPlayers
                       (); //Get the players the user has
461                    while(tradeModel_1.getRowCount()>0){   //Clear
                       the current table
462                       tradeModel_1.removeRow(0);
463                    }
464                    for(int i=0;i<teamPlayers.length;i++){ //
                       Populate the table with the new data
465                       String[] temp={teamPlayers[i].getName(),
                          teamPlayers[i].getPosition(),Float.
                          toString(teamPlayers[i].getPoints())}; //
                          Initialize the row
466                       tradeModel_1.addRow(temp);
467                    }
468                 }
469               }
470         });
471
472         //Action listener for the right trade combo box
473         tradeComboBox_2.addActionListener (new ActionListener ()
              {
474               public void actionPerformed(ActionEvent e) {
475                 if(tradeComboBox_2.getSelectedIndex()!=-1){  //If
                     a user is selected
476                    Player[] teamPlayers=League.teams.get(
                       tradeComboBox_2.getSelectedItem()).getPlayers
                       (); //Get the players the user has
```

```
477                 while(tradeModel_2.getRowCount()>0){    //Clear
                       the current table
478                    tradeModel_2.removeRow(0);
479                 }
480                 for(int i=0;i<teamPlayers.length;i++){ //
                       Populate the table with the new data
481                    String[] temp={teamPlayers[i].getName(),
                          teamPlayers[i].getPosition(),Float.
                          toString(teamPlayers[i].getPoints())}; //
                          Initialize the row
482                    tradeModel_2.addRow(temp);
483                 }
484              }
485           }
486        });
487
488        //Action listener for the trade button
489        btnTrade.addActionListener(new ActionListener() {
490           public void actionPerformed(ActionEvent arg0) {
491              if (tradeComboBox_1.getSelectedIndex()==-1 ||
                    tradeComboBox_2.getSelectedIndex()==-1) {   //If
                     either combo box doesn't have a user selected
492                 GUI.error("Trade error!","Must select two teams
                       to trade between.");
493                 return;
494              }
495              else if(tradeComboBox_1.getSelectedIndex()==
                    tradeComboBox_2.getSelectedIndex()){ //If the
                    same user is selected in each combo box
496                 GUI.error("Trade error!","Must select two
                       different teams to trade between.");
497                 return;
498              }
499              int rows1[] = tradeTable_1.getSelectedRows();   //
                    Get selected rows in the left table
500              int rows2[] = tradeTable_2.getSelectedRows();   //
                    Get selected rows in the right table
501              if(rows1.length==0 && rows2.length==0){   //If no
                    players are selected
502                 GUI.error("Trade error!","Must select at least
                       one player to trade.");
503                 return;
504              }
505              int overwrite = JOptionPane.showConfirmDialog(
                    frmFloodFantasyLeague, "Are you sure you want to
```

```
                trade?");
506              if(overwrite == JOptionPane.YES_OPTION) {
507             Player[] p1 = new Player[rows1.length];   //
                   Initialize player array for the left table
                   selection
508             Player[] p2 = new Player[rows2.length];   //
                   Initialize player array for the right table
                   selection
509             //Populate the player arrays
510             for (int i = 0; i < p1.length; i++) {
511                p1[i] = League.athletes.get(tradeModel_1.
                      getValueAt(rows1[i],0));
512             }
513             for (int i = 0; i < p2.length; i++) {
514                p2[i] = League.athletes.get(tradeModel_2.
                      getValueAt(rows2[i],0));
515             }
516             boolean success=FloodProgram.trade(League.teams
                   .get(tradeComboBox_1  //Determine if it's a
                   successful trade
517                    .getSelectedItem()), p1, League.teams
518                    .get(tradeComboBox_2.getSelectedItem()),
                         p2);
519             if(!success){  //If unsuccessful
520                GUI.error("Invalid trade!","Sorry, your
                      trade violates rules of the league.");
521                return;
522             }
523             //Repopulate the tables
524             Player[] teamPlayers = League.teams.get(
                   tradeComboBox_1.getSelectedItem()).getPlayers
                   ();
525             while (tradeModel_1.getRowCount() > 0) {  //
                   Clear the left trade table
526                tradeModel_1.removeRow(0);
527             }
528             for (int i = 0; i < teamPlayers.length; i++) {
                     //Repopulate the left trade table
529                String[] temp = { teamPlayers[i].getName(),
530                      teamPlayers[i].getPosition(),
531                      Float.toString(teamPlayers[i].
                         getPoints()) };
532                tradeModel_1.addRow(temp); //Add the row
533             }
534             teamPlayers = League.teams.get(tradeComboBox_2.
```

```
                    getSelectedItem()).getPlayers();
535                 while (tradeModel_2.getRowCount() > 0) {  //
                    Clear the left trade table
536                    tradeModel_2.removeRow(0);
537                 }
538                 for (int i = 0; i < teamPlayers.length; i++) {
                         //Repopulate the left trade table
539                    String[] temp = { teamPlayers[i].getName(),
540                          teamPlayers[i].getPosition(),
541                          Float.toString(teamPlayers[i].
                             getPoints()) };
542                    tradeModel_2.addRow(temp); //Add the row
543                 }
544                  }
545            }
546         });
547
548         //Action listener for drop combo box
549         dropComboBox.addActionListener (new ActionListener () {
550             public void actionPerformed(ActionEvent e) {
551               if(dropComboBox.getSelectedIndex()!=-1){  //If a
                    user is selected
552                  Player[] teamPlayers=League.teams.get(
                       dropComboBox.getSelectedItem()).getPlayers();
                        //Get the user's players
553                  while(dropModel.getRowCount()>0){   //Clear the
                        table
554                     dropModel.removeRow(0);
555                  }
556                  for(int i=0;i<teamPlayers.length;i++){ //
                       Repopulate the table
557                     String[] temp={teamPlayers[i].getName(),
                          teamPlayers[i].getPosition(),Float.
                          toString(teamPlayers[i].getPoints())};
558                     dropModel.addRow(temp); //Add the row
559                  }
560               }
561             }
562         });
563
564         //Action listener for drop button
565         btnDrop.addActionListener(new ActionListener() {
566            public void actionPerformed(ActionEvent arg0) {
567               int index=dropTable.getSelectedRow();
568               if(index==-1){ //If no player is selected to drop
```

```
569                 GUI.error("Drop error!","Must select a team to
                       drop a player from.");
570                 return;
571              }
572              int overwrite = JOptionPane.showConfirmDialog(
                    frmFloodFantasyLeague, "Are you sure you want to
                     drop: " + League.athletes.get(dropModel.
                    getValueAt(index,0)).getName());
573                  if(overwrite == JOptionPane.YES_OPTION) {
574                 Player drop=League.athletes.get(dropModel.
                       getValueAt(index,0));   //Get player
575                 boolean success=FloodProgram.dropPlayer(League.
                       teams.get(dropComboBox.getSelectedItem()),
                       drop); //Determine if drop is successful
576                 if(!success){
577                    GUI.error("Invalid drop!","Sorry, your drop
                          violates rules of the league.");
578                    return;
579                 }
580                 dropModel.removeRow(index);   //Delete that row
581                 //Make all the combo boxes not select anything
582                 tradeComboBox_1.setSelectedIndex(-1);
583                 tradeComboBox_2.setSelectedIndex(-1);
584                 while(tradeModel_1.getRowCount()>0){
585                    tradeModel_1.removeRow(0);
586                 }
587                 while(tradeModel_2.getRowCount()>0){
588                    tradeModel_2.removeRow(0);
589                 }
590                  }
591           }
592        });
593
594        tabbedPane.setSelectedIndex(0);
595     }
596  }
597  class MyTableModel extends JTable {
598     /**
599      *
600      */
601     private static final long serialVersionUID = 1L;
602     public MyTableModel(){
603        super();
604     }
605     public MyTableModel(DefaultTableModel model){
```

```
606         super(model);
607     }
608     public MyTableModel(Object[][] data,Object[] columnNames){
609         super(data,columnNames);
610     }
611      public boolean isCellEditable(int row, int col) {
612         return false;
613      }
614 }
```

Listing A.7: IOManager.java

```
1  /*
2   * IOManager.java
3   * This class handles the input and output of the FLOOD GUI
4   */
5
6  import java.io.BufferedReader;
7  import java.io.BufferedWriter;
8  import java.io.DataInputStream;
9  import java.io.FileInputStream;
10 import java.io.FileNotFoundException;
11 import java.io.FileWriter;
12 import java.io.IOException;
13 import java.io.InputStreamReader;
14 import java.util.ArrayList;
15
16 /*
17  * Stats Parser
18  */
19 public class IOManager {
20
21     /**Write the current state of the league to a text file so
           that the program
22      * may be exited and resumes from the place it left off.
23      *
24      * @param League myLeague
25      * @param String filePath
26      * @param int turn
27      */
28     public static void writeState(League myLeague,String
           filePath,int turn){
29         try {
30             FileWriter fstream = new FileWriter(filePath);  //
                   Create the file
```

```
31          BufferedWriter out = new BufferedWriter(fstream);  //
               Initialize the output stream
32          out.write(turn+","+myLeague.getMaxTeamSize()+","+
               myLeague.getMinTeamSize()+","+myLeague.getMaxUser()
               +","+myLeague.getMinUser()+"\n");  //Write the
               first line of the file
33          //Write the actions
34          out.write("ACTIONS:\n");
35          Action[] actions=myLeague.getActions();   //Get the
               actions
36          for(int i=0;i<actions.length;i++){  //Iterate through
                the actions
37             out.write(actions[i].getAction()+","+actions[i].
                  getPoints()+"\n");   //Write each action
38          }
39          //Write the players
40          out.write("PLAYERS:\n");
41          Player[] players=myLeague.getPlayers();   //Get the
               players
42          for(int i=0;i<players.length;i++){  //Iterate through
                the players
43             out.write(players[i].getName()+","+players[i].
                  getPosition()+","+players[i].getPoints()+"\n");
                     //Write each players
44          }
45          //Write the teams
46          out.write("TEAMS:\n");
47          User[] teams=myLeague.getUsers();   //Get the teams
48          for(int i=0;i<teams.length;i++){ //Iterate through
               the teams
49             Player[] teamPlayers=teams[i].getPlayers();  //Get
                   the players of each team
50             out.write(","+teams[i].getName()+","+teams[i].
                  getPoints()+"\n");  //Write each team
51             for(int j=0;j<teamPlayers.length;j++){ //Iterate
                  through the players
52                out.write(teamPlayers[j].getName()+"\n"); //
                     Write the player's name as a reference to the
                      above players
53             }
54          }
55          //Write the free agent
56          out.write("FREE AGENT:\n");
57          User free=League.freeAgent;   //Get the free agen
58          Player[] freePlayers=free.getPlayers();   //Get the
```

```
            players of the free agent
59          for(int i=0;i<freePlayers.length;i++){ //Iterate
               through the players
60             out.write(freePlayers[i].getName()+"\n"); //Write
                  the player's name as a reference to the above
                  players
61          }
62          out.close();   //Close the output stream
63       } catch (IOException e) {
64          GUI.alert("Stat Dump Error!", "Error creating the
               dump file! Please try again.");
65       }
66    }
67
68    /**Read a file representing the state of a program and
         restore that state
69     * to the current program.
70     *
71     * @param League myLeague
72     * @param String filePath
73     * @return int turn
74     */
75    public static int importState(League myLeague,String
         filePath){
76       try {
77          myLeague.clear(); //Clear the current league
78          FileInputStream fstream = new FileInputStream(
               filePath); //Open the file
79          DataInputStream in = new DataInputStream(fstream);//
               Get the object of DataInputStream
80          BufferedReader br = new BufferedReader(new
               InputStreamReader(in));   //Initialize the buffered
                reader
81          //Initialize variables
82          String str="",team="";
83          int turn=-1;
84          str=br.readLine();   //Read the first line
85          String[] data=str.split(",\\s*");   //Split on commas
86          if(data.length!=5){
87             in.close();
88             myLeague.clear();
89             GUI.alert("Dump Import Error!", "Invalid dump file
                  !");
90             return -1;
91          }
```

```
92          //Store the data
93          turn=Integer.parseInt(data[0]);
94          System.out.println("Parsed: "+data[0]+" to "+turn);
95          myLeague.setMaxTeamSize(Integer.parseInt(data[1]));
96          myLeague.setMinTeamSize(Integer.parseInt(data[2]));
97          myLeague.setMaxUser(Integer.parseInt(data[3]));
98          myLeague.setMinUser(Integer.parseInt(data[4]));
99
100         boolean teamFlag=false,playerFlag=false,actionFlag=
               false,freeFlag=false;
101         while((str = br.readLine()) != null){  //Read File
               Line By Line
102            if(str.equalsIgnoreCase("ACTIONS:")){  //Actions
                  reached
103               actionFlag=true;
104               playerFlag=false;
105               teamFlag=false;
106               freeFlag=false;
107            }
108            else if(str.equalsIgnoreCase("PLAYERS:")){   //
                  Players reached
109               playerFlag=true;
110               actionFlag=false;
111               teamFlag=false;
112               freeFlag=false;
113            }
114            else if(str.equalsIgnoreCase("TEAMS:")){  //Teams
                  reached
115               teamFlag=true;
116               actionFlag=false;
117               playerFlag=false;
118               freeFlag=false;
119            }
120            else if(str.equals("FREE AGENT:")){ //Free agent
                  reached
121               teamFlag=false;
122               actionFlag=false;
123               playerFlag=false;
124               freeFlag=true;
125            }
126            else{ //Data
127               if(actionFlag){   //If currently looking at
                     actions
128                  String[] parts=str.split(",\\s*");  //Split
                        on commas
```

```
129                     if(parts.length!=2){ //Validate that it's an
                            action
130                         in.close();
131                         myLeague.clear();
132                         GUI.alert("Dump Import Error!", "Invalid
                               dump file!");
133                         return -1;
134                     }
135                     myLeague.addAction(new Action(parts[0].trim
                           (),Float.parseFloat(parts[1].trim())));
                           //Add action to league
136                 }
137                 else if(playerFlag){ //If currently looking at
                       players
138                     String[] parts=str.split(",\\s*");  //Split
                           on commas
139                     if(parts.length!=3){ //Validate that it's a
                           player
140                         in.close();
141                         myLeague.clear();
142                         GUI.alert("Dump Import Error!", "Invalid
                               dump file!");
143                         return -1;
144                     }
145                     myLeague.addPlayer(new Player(parts[0].trim
                           (),parts[1].trim(),Float.parseFloat(parts
                           [2].trim()))); //Add player to league
146                 }
147                 else if(teamFlag){   //If currently looking at
                       teams
148                     String[] parts=str.split(",\\s*");  //Split
                           on commas
149                     if(str.charAt(0)==','){ //If it's a team
                           name
150                         if(parts.length!=3){ //Validate that it's
                                a team name
151                             in.close();
152                             myLeague.clear();
153                             GUI.alert("Dump Import Error!", "
                                   Invalid dump file!");
154                             return -1;
155                         }
156                         team=parts[1].trim();   //Trim white
                               space
157                         myLeague.addUser(new User(team,Float.
```

```
                        parseFloat(parts[2].trim())));  //Add
                        team to league
158                 }
159                 else{ //If it's a team player
160                    if(parts.length!=1){ //Validate that it's
                        a player name
161                       in.close();
162                       myLeague.clear();
163                       GUI.alert("Dump Import Error!", "
                           Invalid dump file!");
164                       return -1;
165                    }
166                    myLeague.getTeam(team).addPlayer(myLeague
                       .getPlayer(parts[0].trim()));  //Get
                       reference to player and add to team
167                 }
168              }
169              else if(freeFlag){   //If currently looking at
                    free agent
170                 String[] parts=str.split(",\\s*");  //Split
                    on commas
171                 if(parts.length!=1){ //Validate that it's a
                    player name
172                    in.close();
173                    myLeague.clear();
174                    GUI.alert("Dump Import Error!", "Invalid
                       dump file!");
175                    return -1;
176                 }
177                 myLeague.getFreeAgent().addPlayer(myLeague.
                    getPlayer(parts[0].trim())); //Get
                    reference to player and add player to free
                     agent
178              }
179           }
180        }
181        in.close(); //Close input stream
182        if(!freeFlag){ //If the free agent was never reached
183           in.close();
184           myLeague.clear();
185           GUI.alert("Dump Import Error!", "Invalid dump file
                !");
186           return -1;
187        }
188        return turn;   //Return current turn
```

```
189         } catch (FileNotFoundException e) {
190            myLeague.clear();
191            GUI.alert("Dump Import Error!", "File not found!");
192            return -1;
193         } catch (IOException e) {
194            myLeague.clear();
195            GUI.alert("Dump Import Error!", "Error writing file!"
                  );
196            return -1;
197         } catch (IndexOutOfBoundsException e){
198            myLeague.clear();
199            GUI.alert("Dump Import Error!", "Invalid dump file!")
                  ;
200            return -1;
201         }
202      }
203
204      /**Upload the statistics from a file.
205       *
206       * @param League myLeague
207       * @param String fileName
208       */
209      public static void uploadStats(League myLeague,String
            filePath){
210         try {
211            ArrayList<String[]> statsAL = new ArrayList<String
                  []>();
212            FileInputStream fstream = new FileInputStream(
                  filePath); //Open the file
213            DataInputStream in = new DataInputStream(fstream); //
                  Get the object of DataInputStream
214            BufferedReader br = new BufferedReader(new
                  InputStreamReader(in));
215            String str;
216            String[] stats;
217            boolean valid=true;
218            while((str = br.readLine()) != null){  //Read File
                  Line By Line
219               stats=str.split(",\\s*");
220               valid=valid && myLeague.getPlayer(stats[0])!=null;
                      //Check if the athlete exists
221               if(!valid){ //If the athlete doesn't exist
222                  in.close();
223                  GUI.error("Athelete doesn't exist! ",stats[0]+"
                         is not a valid athlete.");
```

```
224                 return;
225             }
226             valid=valid && myLeague.getAction(stats[1])!=null;
                    //Check if the action exists
227             if(!valid){ //If the athlete doesn't exist
228                 in.close();
229                 GUI.error("Action doesn't exist! ",stats[1]+"
                        is not a valid action.");
230                 return;
231             }
232             valid=valid && Integer.parseInt(stats[2])>0; //
                    Check if the quantity is greater than zero
233             if(!valid){ //If the quantity is less than or
                    equal to zero
234                 in.close();
235                 GUI.error("Quantity must be positive! ",stats
                        [2]+" is not a positive number greater than
                        zero.");
236                 return;
237             }
238             statsAL.add(stats);  //Split on commas
239         }
240         in.close(); //Close
241         for(int i=0;i<statsAL.size();i++){  //Iterate through
                the stats
242             float pts=myLeague.getAction(statsAL.get(i)[1]).
                    getPoints() * Integer.parseInt(statsAL.get(i)
                    [2]); //Compute the points
243             Player temp=myLeague.getPlayer(statsAL.get(i)[0]);
                    //Get the player
244             temp.addPoints(pts); //Add the points to the
                    player and thereby the team they're one
245         }
246     } catch (FileNotFoundException e) {
247         GUI.alert("Stat Parsing Error!","File not found!");
248     } catch (IOException e) {
249         GUI.alert("Stat Parsing Error!","Error opening the
                file!");
250     } catch (IndexOutOfBoundsException e){
251         GUI.alert("Stat Parsing Error!","Invalid stat file!")
                ;
252     }
253   }
254 }
```

Listing A.8: League.java

```
1  /*
2   * League.java
3   * This class handles the the League elements in the fantasy
       league
4   */
5
6  import java.util.ArrayList;
7  import java.util.Arrays;
8  import java.util.HashMap;
9
10
11 public class League {
12    private String name;
13    private int maxTeamSize,minTeamSize,maxUsers,minUsers;
14    public static HashMap<String,User> teams;
15    public static HashMap<Player,User> playerToTeam;
16    public static HashMap<String,Player> athletes;
17    public static HashMap<String,Action> ptsDist;
18    public static ArrayList<User> indexedTeams;
19    public static User freeAgent;
20
21    /**Constructor for League.
22     *
23     * @param String name
24     */
25    public League(String name){
26       this.name=name;
27       //Initialize data structures
28       teams=new HashMap<String,User>();
29       indexedTeams=new ArrayList<User>();
30       athletes=new HashMap<String,Player>();
31       ptsDist=new HashMap<String,Action>();
32       playerToTeam= new HashMap<Player,User>();
33       freeAgent=new User("Free Agent");
34    }
35
36    /**Add an action to the league.
37     *
38     * @param Action a
39     */
40    public void addAction(Action a){
41       ptsDist.put(a.getAction(),a);
42    }
```

```
43
44      /**Add a user to the league.
45       *
46       * @param User u
47       */
48      public void addUser(User u){
49         teams.put(u.getName(),u);
50         indexedTeams.add(u); //Store the indexed user
51      }
52
53      /**Add a player to the league.
54       *
55       * @param p
56       */
57      public void addPlayer(Player p){
58         athletes.put(p.getName(),p);  //Add to the list of
              players
59         freeAgent.addPlayer(p); //Add to the free agent
60      }
61
62      /**Get a user based on an index.
63       *
64       * @param int index
65       * @return User u
66       */
67      public User getUser(int index){
68         return indexedTeams.get(index);
69      }
70
71      /**Get the max team size.
72       *
73       * @return int maxTeamSize
74       */
75      public int getMaxTeamSize() {
76         return maxTeamSize;
77      }
78
79      /**Set the max team size.
80       *
81       * @param int maxTeamSize
82       */
83      public void setMaxTeamSize(int maxTeamSize) {
84         this.maxTeamSize = maxTeamSize;
85      }
86
```

```
87      /**Get the min team size.
88       *
89       * @return int minTeamSize
90       */
91      public int getMinTeamSize() {
92          return minTeamSize;
93      }
94
95      /**Set the min team size.
96       *
97       * @param int minTeamSize
98       */
99      public void setMinTeamSize(int minTeamSize) {
100         this.minTeamSize = minTeamSize;
101     }
102
103     /**Get the max number of users
104      *
105      * @return int maxUsers
106      */
107     public int getMaxUser() {
108         return maxUsers;
109     }
110
111     /**Set the max number of users
112      *
113      * @param int maxUsers
114      */
115     public void setMaxUser(int maxUsers) {
116         this.maxUsers = maxUsers;
117     }
118
119     /**Get the min number of users
120      *
121      * @return int minUsers
122      */
123     public int getMinUser() {
124         return minUsers;
125     }
126
127     /**Set the min number of users.
128      *
129      * @param int minUsers
130      */
131     public void setMinUser(int minUsers) {
```

```
132         this.minUsers = minUsers;
133     }
134 
135     /**Get the name of the league.
136      *
137      * @return String name
138      */
139     public String getName() {
140         return name;
141     }
142 
143     /**Get the current number of users in the league.
144      *
145      * @return int numUsers
146      */
147     public int getCurrentNumUsers(){
148         return teams.size();
149     }
150 
151     /**Get the current number of players in the league.
152      *
153      * @return int numPlayers
154      */
155     public int getCurrentNumPlayers(){
156         return teams.size();
157     }
158 
159     /**Get the current number of actions in the league.
160      *
161      * @return int numActions
162      */
163     public int getCurrentNumActions(){
164         return athletes.size();
165     }
166 
167     /**Get a user from their name.
168      *
169      * @param String name
170      * @return User u
171      */
172     public User getTeam(String name){
173         return teams.get(name);
174     }
175 
176     /**Get a player from their name.
```

```
177      *
178      * @param String name
179      * @return Player p
180      */
181     public Player getPlayer(String name){
182        return athletes.get(name);
183     }
184
185     /**Get an action from the rule.
186      *
187      * @param String action
188      * @return Action a
189      */
190     public Action getAction(String action){
191        return ptsDist.get(action);
192     }
193
194     /**Get the free agent.
195      *
196      * @return User freeAgent
197      */
198     public User getFreeAgent(){
199        return freeAgent;
200     }
201
202     /**Get the users in reverse ranked order based on
203      * the number of points they have.
204      *
205      * @return Users[] rankedTeams
206      */
207     public User[] getRankedUsers(){
208        User[] ranked=teams.values().toArray(new User[teams.size
              ()]);
209        Arrays.sort(ranked);
210        return ranked;
211     }
212
213     /**Get the players that are still in the draft in reverse
214      * ranked order.
215      *
216      * @return Player[] availablePlayers
217      */
218     public Player[] getRankedAvailablePlayers(){
219        return freeAgent.getPlayers();
220     }
```

```
221
222     /**Get all the players in the league.
223      *
224      * @return Player[] allPlayers
225      */
226     public Player[] getPlayers(){
227        return athletes.values().toArray(new Player[athletes.
              size()]);
228     }
229
230     /**Get all the actions in the league.
231      *
232      * @return Action[] allActions
233      */
234     public Action[] getActions(){
235        return ptsDist.values().toArray(new Action[ptsDist.size
              ()]);
236     }
237
238     /**Get all the users in the correct indexed order.
239      *
240      * @return User[] users
241      */
242     public User[] getUsers(){
243        return indexedTeams.toArray(new User[indexedTeams.size()
              ]);
244     }
245     /**Clear all the data structures.
246      *
247      */
248     public void clear(){
249        teams.clear();
250        playerToTeam.clear();
251        athletes.clear();
252        ptsDist.clear();
253        indexedTeams.clear();
254        freeAgent.clear();
255     }
256
257     /**Get a string representation of the league's statistics.
258      *
259      * @return String league
260      */
261     public String toString() {
262        return "League [name=" + name + ", maxTeamSize=" +
```

```
            maxTeamSize
263             + ", minTeamSize=" + minTeamSize + ", maxUser=" +
                   maxUsers
264             + ", minUser=" + minUsers + "]";
265     }
266
267 }
```

Listing A.9: makefile

```
1  # makefile to generate the FLOOD compiler frontend
2
3  JFLEX      = jflex
4  JAVAC      = javac
5  JAR        = jar -cf
6  DELETE = rm -rf
7  BYACCJ = byaccj -J
8
9  # targets:
10
11 all: flood_frontend.jar
12
13 run: flood_frontend.jar
14    java Parser
15
16 build: clean flood_frontend.jar
17
18 clean:
19    $(DELETE) *.*~ *.class
20    $(DELETE) ParserVal.java
21    $(DELETE) Parser.java
22    $(DELETE) Yylex.java
23    $(DELETE) flood_front.jar
24    $(DELETE) FloodProgram.java
25
26 flood_frontend.jar: Parser.class
27    $(JAR) flood_front.jar *.class
28
29 Parser.class: Yylex.java Parser.java
30    $(JAVAC) Parser.java
31
32 Yylex.java: flood_lex.flex
33    $(JFLEX) flood_lex.flex
34
35 Parser.java: flood_grammar.y
```

```
36    $(BYACCJ) flood_grammar.y
```

Listing A.10: Player.java

```
1  /*
2   * Player.java
3   * This class handles an individual player in the league
4   */
5
6  public class Player implements Comparable<Player>{
7     private String name, position;
8     private float totalPoints;
9
10    /**Constructor with just the name and position.
11      *
12      * @param String name
13      * @param String position
14      */
15    public Player(String name, String position){
16       this.name=name;
17       this.position=position;
18       totalPoints=0;
19    }
20
21    /**Constructor with the name and points. Used for importing
22      * dump files.
23      *
24      * @param String name
25      * @param String position
26      * @param float totalPoints
27      */
28    public Player(String name, String position,float
         totalPoints){
29       this.name=name;
30       this.position=position;
31       this.totalPoints=totalPoints;
32    }
33
34    /**Get the player's name.
35     *
36     * @return
37     */
38    public String getName() {
39       return name;
40    }
```

```
41
42      /**Return a string representation of the player.
43       *
44       * @return String player
45       */
46      public String toString() {
47         return "Player [name=" + name + ", position=" + position
48               + ", totalPoints=" + totalPoints + "]";
49      }
50
51      /**Get the player's position.
52       *
53       * @return String position
54       */
55      public String getPosition() {
56         return position;
57      }
58
59      /**Add points to the player and the team they are on.
60       *
61       * @param float pts
62       */
63      public void addPoints(float pts){
64         totalPoints+=pts; //Add pts to players points
65         User temp=League.playerToTeam.get(this);  //Get team
              they are on
66         if(temp!=null) //Free agent
67            temp.addPoints(pts); //Add the points to the team
68      }
69
70      /**Get the player's total points.
71       *
72       * @return float totalPoints
73       */
74      public float getPoints(){
75         return totalPoints;
76      }
77
78      /**Determine if two players are equal.
79       *
80       * @return boolean areEqual
81       */
82      public boolean equals(Object obj){
83         if (this == obj)  //Same reference
84            return true;
```

```
85          if (obj == null)  //Other is null
86              return false;
87          if (getClass() != obj.getClass())   //Not the same class
88              return false;
89          final Player other = (Player) obj;  //Cast other object
90          if (name.equals(other.name) && position.equals(other.
               position))   //Same name and position
91              return true;
92          return false;
93      }
94
95      /**Compare two players based on their total points scored.
96       *
97       * @return int compared
98       */
99      public int compareTo(Player o) {
100         if(totalPoints>o.getPoints())
101             return 1;
102         else if(totalPoints==o.getPoints())
103             return 0;
104         return -1;
105     }
106
107 }
```

Listing A.11: run.sh

```
1 #!/bin/bash
2 java Parser $1
3 javac FloodProgram.java
4 java FloodProgram
```

Listing A.12: User.java

```
1  /*
2   * User.java
3   * This class handles the users in the league
4   */
5
6  import java.util.Arrays;
7  import java.util.HashMap;
8
9
10 public class User implements Comparable<User>{
```

```
11      private float totalPoints;
12      private String name;
13      private HashMap<String,Player> teamAthletes;
14
15      /**Constructor with just the name.
16       *
17       * @param String name
18       */
19      public User(String name){
20        this.name=name;
21        totalPoints=0;
22        teamAthletes=new HashMap<String,Player>();
23      }
24
25      /**Constructor with the name and points. Used for
           importing
26       * dump files.
27       *
28       * @param String name
29       * @param float totalPoints
30       */
31      public User(String name,float totalPoints){
32        this.name=name;
33        this.totalPoints=totalPoints;
34        teamAthletes=new HashMap<String,Player>();
35      }
36
37      /**Add a player to the team and if the team is not the
           free agent,
38       * remove the player from the free agent.
39       *
40       * @param Player athlete
41       */
42      public void addPlayer(Player athlete){
43        teamAthletes.put(athlete.getName(),athlete); //Add
             player to this team
44        if(!name.equals("Free Agent")){  //If it's not the free
             agent team
45          League.playerToTeam.put(athlete,this); //Add the
               association of player to team mapping
46          League.freeAgent.removePlayer(athlete);   //Remove
               this player from the free agent
47        }
48      }
49
```

```
50      /**Remove a player from a team and if the team is not the
           free agent,
51       * add the player to the free agent.
52       *
53       * @param Player athlete
54       */
55      public void removePlayer(Player athlete){
56        teamAthletes.remove(athlete.getName());   //Remove
             athlete from this team
57        if(!name.equals("Free Agent")){  //If it's not the free
             agent
58           League.playerToTeam.remove(athlete);   //Remove the
                association of player to team
59           League.freeAgent.addPlayer(athlete);   //Add the
                player to the free agent
60        }
61      }
62
63      /**Add points to the team.
64       *
65       * @param float points
66       */
67      public void addPoints(float points){
68        totalPoints+=points;
69      }
70
71      /**Get the user's points.
72       *
73       * @return float totalPoints
74       */
75     public float getPoints() {
76        return totalPoints;
77     }
78
79     /**Get the name of the user.
80      *
81      * @return String name
82      */
83     public String getName() {
84        return name;
85     }
86
87     /**Get the number of players on the user's team.
88      *
89      * @return int numPlayers
```

```
90       */
91      public int getNumPlayers(){
92         return teamAthletes.size();
93      }
94
95      /**Get the player on the user's team in reverse ranked
96       * order.
97       *
98       * @return Player[] rankedPlayers
99       */
100     public Player[] getPlayers(){
101        Player[] ranked=teamAthletes.values().toArray(new Player
              [teamAthletes.size()]);
102        Arrays.sort(ranked);
103        return ranked;
104     }
105
106     /**Determine if two users are the same.
107      *
108      * @return boolean areEqual
109      */
110     public boolean equals(Object obj){
111        if (this == obj)  //Same reference
112           return true;
113        if (obj == null)  //Other is null
114           return false;
115        if (getClass() != obj.getClass())   //Not the same class
116           return false;
117        final User other = (User) obj;   //Cast object
118        if (name.equals(other.name))  //If the names are equal
119           return true;
120        return false;
121     }
122
123     /**Compare one user to another based on their number of
           points.
124      *
125      * @return int compared
126      */
127     public int compareTo(User o) {
128        if(totalPoints>o.getPoints())
129           return 1;
130        else if(totalPoints==o.getPoints())
131           return 0;
132        return -1;
```

```
133     }
134
135     /**Return a string representation of the user.
136      *
137      * @return String user
138      */
139     public String toString() {
140        return "User [name=" + name + ", points=" + totalPoints
             + "]";
141     }
142
143     /**Clear the data structures.
144      *
145      */
146     public void clear(){
147        teamAthletes.clear();
148     }
149 }
```