Homework 01 (Due: Friday, February 17, 2017, 11 : 59 : 00PM Central Time)

CSCE 322

1 Instructions

In this assignment, you will be required to scan, parse, and check the semantics of a file that encodes the state of a variation of Battleflood. The definition of a properly formatted input file is given in Section 1.1.

You will be submitting one `.java` file and two `.g4` (ANTLR) files via web hand-in.

1.1 File Specification

- The file contains two (2) labeled sections: **Moves** and **Game** . Each section is enclosed by start and end tags (`{` and `}`, respectively).
- **Moves** is a color-separated (`:`) list of numbered, contiguous areas that appear between `(` and `)` tokens. Valid **Moves** are numerical symbols.
- **Game** contains a two-dimensional array of colon-separated entries that uses numeric symbols to encode the state of the game. Rows will be ended with a `;` and the **Game** will be begun with a `[` and ended with a `]`. The two-dimensonal array contains the number of the player who currently occupies that given location.

An example of a properly formatted file is shown in Figure 1.

```
@game {
[
2 : 1 : 1 : 1 : 2 : 2 : 3 : 3 : 1 : 3 ;
3 : 3 : 1 : 1 : 3 : 2 : 3 : 2 : 1 : 1 ;
1 : 1 : 1 : 3 : 2 : 1 : 2 : 3 : 2 : 1 ;
2 : 2 : 2 : 3 : 1 : 1 : 3 : 2 : 3 : 3 ;
1 : 3 : 1 : 2 : 3 : 3 : 3 : 1 : 3 : 2 ;
2 : 3 : 1 : 2 : 2 : 2 : 3 : 2 : 3 : 3 ;
3 : 2 : 1 : 2 : 3 : 1 : 3 : 1 : 1 : 1
]
}
@moves {
( 12 : 11 : 8 : 3 : 2 : 1 )
}
```

Figure 1: A properly formatted BattleFlood Format (bff) encoding

The assignment is made up of two parts: scanning the text of the input file and parsing the information contained in the input file.

1.2 Scanning

Construct a combined grammar in a `.g4` file that ANTLR can use to scan a supplied BattleFlood Format encoding. The logic in this file should be robust enough to identify tokens in the encoding and accurately process any correctly formatted encoding. The rules in your `.g4` file will be augmented with actions that display information about the input file. An example of that output is specified in Section 2.

The purpose of the scanner is to extract tokens from the input and pass those along to the parser. For the BattleFlood Format encoding, the types of tokens that you will need to consider are given in Table 1.

Type	Form
Section Beginning	`{`
Section Ending	`}`
Section Title	`@game` and `@moves`
Move/Game Symbol	One or more Numerical Symbols
Numerical Symbol	`0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, or `9`
Row Ending	`;`
Game Beginning	`[`
Game Ending	`]`
Moves Beginning	`(`
Moves Ending	`)`
White Space (to be ignored)	spaces, tabs, newlines

Table 1: Tokens to Consider

1.2.1 Invalid Encodings

For invalid BattleFlood encodings, the output `Abomination: T in line L.` should display. `T` would be the symbol read and `L` would be the line of input where the symbol was read. Your scanner should stop scanning the file after an unrecognized token is found.

1.3 Parsing

Construct a combined grammar in a `.g4` file that ANTLR can use to parse a supplied BattleFlood encoding. In addition to the rules for scanning, there are several parsing rules:

- Each section appears once and only once. The sections may appear in either **Moves** /**Game** or **Game** /**Moves** order.
- There must be more than four (4) rows in a valid **Game** .
- There must be more than four (4) columns in a valid **Game** .

  You may assume that each row has the same number of columns, and each column has the same number of rows.
- There must be more than four (4) locations in the **Moves** section.

The semantics of a properly formatted Greater Than Sudoku encoding are:

1. The **Game** must have 2 to 4 players
2. The number of **Moves** must be more than three (3) times as large as the number of players in the **Game**
3. The **Moves** must be in descending order
4. Player 1 may not occupy more than 25% of the spaces in the **Game** .
5. **Extra Credit (**10 **points or Honors contract)**: No player in the **Game** may occupy a block of just one space.

2 Output

2.1 Scanner

Your `.g4` file should produce output for both correctly formatted files and incorrectly formatted files. For the correctly formatted file in Figure 1, the output would have the form of the output presented in Figure 2

```
Section: @game
Section Initiation: {
Game Initiation: [
Number: 2
...
Number: 3
Row Termination: ;
Number: 3
...
Number: 1
Number: 3
Number: 1
Number: 1
Number: 1
Game Termination: ]
Section Termination: }
Section: @moves
Section Initiation: {
List Initiation: (
Number: 12
Number: 11
...
Number: 1
List Termination: )
Section Termination: }
File Termination
```

Figure 2: Truncated Output of Scanner for File in Figure 1

For a correctly formatted file in Part 2, the output would be: `This game has p players.` where `p` is the number of players in the **Game** . For the file in Figure 1, the output would be
`This game has 3 players.`.

2.1.1 Invalid Syntax & Semantics in Parsing

For invalid encodings in Part 2, a message describing the error should be displayed. For a syntax error (violation of the syntax rules), the output
`Abomination: T in line L.` should be displayed, where `L` is the line number where the unrecognized token `T` was found. For that error, the parser should stop processing the file. For a semantic rule violation, the output
`Semantic Abomination R` should be displayed, where `R` is the number of the rule (from List 1.3) that was violated, but parsing should continue.

Syntax errors in Part 2 should be reported in the `syntaxError` method of `csce322h0mework01partt02error.java`.

3 Naming Conventions

The ANTLR file for the first part of the assignment should be named `csce322h0mework01part01.g4`. The ANTLR file for the second part of the assignment should be named `csce322h0mework01part02.g4`. Both

grammars should contain a start rule named `flood`. The Java file for the second part of the assignment should be named `csce322h0mework01part02error.java`.

4 webgrader

The webgrader is available for this assignment. You can test your submitted files before the deadline by submitting them on webhandin and going to http://cse.unl.edu/~cse322/grade, choosing the correct assignment and entering your `cse.unl.edu` credentials

The script should take approximately 2 minutes to run and produce a PDF.

4.1 The Use of `diff`

Because Part 1 of this assignment only depends on the symbols in the file, the order in which they are displayed should not be submission dependent. Therefore, `diff` will be used to compare the output of a particular submission against the output of the solution implementation.

5 Point Allocation

Component	Points
Part 1	35
Part 2	65
Total	100

6 External Resources

ANTLR
Getting Started with ANTLR v4
ANTLR 4 Documentation
Overview (ANTLR 4 Runtime 4.6 API)

7 Commands of Interest

```
alias antlr4='java -jar /path/to/antlr-4.6-complete.jar'
alias grun='java org.antlr.v4.gui.TestRig'
export CLASSPATH="/path/to/antlr-4.6-complete.jar:$CLASSPATH"
antlr4 /path/to/csce322h0mework01part0#.g4
javac -d /path/for/.classfiles /path/to/csce322h0mework01part0#*.java
java /path/of/.classfiles csce322h0mework01part02driver /path/to/inputfile
grun csce322h0mework01partt0# flood -gui
grun csce322h0mework01part0# flood -gui /path/to/inputfile
grun csce322h0mework01partt0# flood
grun csce322h0mework01part0# flood /path/to/inputfile
```