Assignment 01 (Due: Friday, October 4, 2019, 11 : 59 : 00PM)

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 Slippery Crossings. 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: **Maze** and **Moves** . Each section is enclosed by start and end tags (`{` and `}`, respectively). The value of the section is set by the `@` assignment operator.
- **Moves** is an underscore-separated (`_`) list of move values that appear between `[` and `]`. Valid moves are `u`, `d`, `l`, and `r`.
- **Maze** is a two-dimensional array of space-separated entries that uses alphanumeric symbols to encode the state of the maze. Rows will be ended with a `$` and the **Maze** will be begun with an `(` and ended with a `)`.

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

```
{ Moves    @ [   u _   r _   l _   u _   u _   d _   r _   u _   d _   u _   u _   u _
   l _   u _   l _   u _   l _   d _   d _   r]     }
{ Maze    @     (
  x     x     x     x     x     x     x     x     x     x$
  x     -     -     -     -     -     -     -     -     x$
  x     -     -     -     -     -     1     -     -     x$
  x     -     -     -     -     -     -     -     -     x$
  x     -     -     -     -     -     -     -     x     x$
  x     -     -     -     -     -     -     -     -     x$
  x     -     -     -     -     x     -     -     -     x$
  x     -     -     -     -     -     -     x     4     x$
  x     -     -     -     -     -     2     -     -     x$
  x     g     -     -     -     -     -     -     -     x$
  x     -     -     -     -     -     -     -     -     x$
  x     -     -     -     -     -     -     -     -     x$
  x     -     -     -     -     -     -     x     3     x$
  x     x     x     x     x     x     x     x     x     x
)
}
```

Figure 1: A properly formatted Maze Game (M) 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 Maze Game 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 Maze Game encoding, the types of tokens that you will need to consider are given in Table 1.

Type	Form
Section Beginning	`{`
Section Ending	`}`
Section Title	`Moves` and `Maze`
Assign Value	`@`
Move Symbol	`u`, `d`, `l`, or `r`
Maze Symbol	`-`, `g`, `x`, or one (1) or more numerical symbols
Numerical Symbol	`0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`
Row Ending	`$`
Maze Beginning	`(`
Maze Ending	`)`
List Beginning	`[`
List Ending	`]`
White Space (to be ignored)	spaces, tabs, newlines

Table 1: Tokens to Consider

1.2.1 Invalid Encodings

For invalid Maze Game encodings, the output `Something on Line L is Unrecognized.` should display. `L` would be the line of input where the problem occurred. 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 Maze Game 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** /**Maze** or **Maze** /**Moves** order.
- There must be more than six (6) rows in a valid **Maze** .
- There must be more than six (6) columns in a valid **Maze** .

  You may assume that each row has the same number of columns, and each column has the same number of rows.
- The first row, last row, first column, and last column of the **Maze** must contain only `x` symbols.
- There must be more than four (4) moves in the **Moves** section.

The semantics of a properly formatted Maze Game encoding are:

1. Between one (1) and four (4) (inclusive) players must appear in the **Maze** .
2. The **Maze** must contain exactly one (1) `g` symbol.
3. The number of `x` symbols in the **Maze** must not exceed 50% of all symbols in the **Maze** .
4. `u`, `d`, `l`, and `r` must each appear at least once in the **Moves** .
5. **Extra Credit (10 points or Honors contract)**: Every player must have at least one (1) valid move available to them

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 Beginning: {
Section Title: Moves
Assign Value: @
List Beginning: [
Move Symbol: u
Move Symbol: r
...
Move Symbol: r
List Ending: ]
Section Ending: }
Section Beginning: {
Section Title: Maze
Assign Value: @
Maze Beginning: (
Maze Symbol: x
Maze Symbol: x
..
Maze Symbol: x
Row Ending: $
Maze Symbol: x
Maze Symbol: -
...
Maze Symbol: x
Row Ending: $
Maze Symbol: x
...
Maze Symbol: x
Maze Ending: )
Section Ending: }
File Ending
```

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

For a correctly formatted file in Part 2, the output would be: `There are p players.` where `p` is the number of players in the **Maze** . For the file in Figure 1, the output would be `There are 4 players.`.

2.1.1 Invalid Syntax & Semantics in Parsing

For invalid Maze Game encodings in Part 2, a message describing the parsing error should be displayed. For an unrecognized token (not in the alphabet of tokens), the output `Something on Line L is Problematic.` should be displayed, where `L` is the line number where the problem occurred. For a semantic violation, the output `Semantic Violation: Rule 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 `csce322a01part02error.java`.

3 Naming Conventions

The ANTLR file for the first part of the assignment should be named `csce322a01part01.g4`. The ANTLR file for the second part of the assignment should be named `csce322a01part02.g4`. Both grammars should contain a start rule named `mazeGame`. The Java file for the second part of the assignment should be named `csce322a01part02error.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. In Part 2, the output is sorted and the unique lines extracted, so the order and number of times a semantic error is reported will not affect the diff.

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 4.7.2 API)

7 Commands of Interest

```
alias antlr4='java -jar /path/to/antlr-4.7.2-complete.jar'
alias grun='java org.antlr.v4.gui.TestRig'
export CLASSPATH="/path/to/antlr-4.7.2-complete.jar:$CLASSPATH"
antlr4 /path/to/csce322a01part0#.g4
javac -d /path/for/.classfiles /path/to/csce322a01part0#*.java
java /path/of/.classfiles csce322a01part02driver /path/to/inputfile
grun csce322a01part0# mazeGame -gui
grun csce322a01part0# mazeGame -gui /path/to/inputfile
grun csce322a01part0# mazeGame
grun csce322a01part0# mazeGame /path/to/inputfile
```