**OVERVIEW**

Welcome to CS32! This is an optional (but recommended) assignment for those who are unsure if they are prepared for CS32, want a quick refresher on their Java skills, or want to prove they are awesome. If you are well-prepared to take this course, you should be able to complete the majority of the requirements for this assignment in a few hours. A great Boggle implementation, particularly one with a complicated graphical user interface (GUI), would take considerably longer. But don't worry too much about that yet, as we'll be building your skills to be able to write great implementations throughout the semester in labs.

Don't know/remember how to play Boggle? Play a round or two on [this website](http://www.wordplays.com/boggle) for a quick introduction.

In this assignment, your main task is implementing a single method, Board.play(). The play() method examines the Board's state and returns a Set<String> containing all of the words that can be found there according to Boggle rules. But besides implementing play(), we encourage you to read all of the provided code. It contains a lot of comments and questions that you should at least be able to understand, even if you can't always answer the questions perfectly.

There are a few things about this project that will be new to you. Don't panic! First, we are using [Maven](http://maven.apache.org/guides/getting-started/index.html#What_is_Maven) to manage library dependencies and to build the project. You can certainly use an IDE like IntelliJ, but all of your submissions must also compile using maven so that we can automate testing. You'll also need to use Maven to run some tools against your code. These tools will analyze your code looking for common mistakes, and code that violates our [style guide](https://cs.brown.edu/courses/cs0320/docs/style-guide.html). You can avoid losing easy points by listening to what these tools tell you. Run these tools early and often, so that simple mistakes don't pile up.

Maven makes it easy to add needed libraries to your projects. In general, CS32 *encourages* the use of standard libraries. We've included several that we like in this build ([JUnit](http://junit.org/), [Guava](https://github.com/google/guava), [Gson](https://github.com/google/gson/blob/master/UserGuide.md), [JOptSimple](https://jopt-simple.github.io/jopt-simple/)). You'll be getting to know all of them better throughout the semester.

Another surprise is [Spark](http://sparkjava.com/). Spark is a small library that makes it easy to run a web server inside your program. This way, your program can communicate with the user by producing HTML pages (like Boggle boards) and take input from HTML forms (like word submissions). We'll be using Spark this year so that the skills you learn creating GUIs will be more directly transferrable to the real world. That means you'll be seeing [CSS](http://www.w3schools.com/css/css_intro.asp) as a way to make your programs look good, and you'll be using [JavaScript](http://javascript.crockford.com/) to implement parts of your user interfaces (we'll teach you those in labs). The link above will lead you to an introduction to Spark. For more specific details of the API, check [Spark's Javadoc](http://javadoc.io/doc/com.sparkjava/spark-core/2.6.0).

**SETTING UP**

Before we start you must configure your environment to use Maven 3 and Java 11, which are installed by default on department machines.

Confirm that you are using the proper versions by running mvn -version. The first line should mention Maven 3.X.X Next, run java -version. The first line should mention version 11.0.XX. Version 11.\* of Java is commonly referred to as "Java 11." If these versions aren't right, get help now.

If you want to develop on your personal computer rather than by remotely using a department machine, you'll need to install the JDK (version 11), Maven (version 3), and perhaps IntelliJ. But there are quite a few IDEs out there so feel free to work on whichever one works the best for you (e.g. Eclipse) or none at all (e.g. emacs or vim). Check out our [Getting Set Up](https://cs.brown.edu/courses/cs0320/labs/getting-started.html) lab for detailed information about working on your personal computer.

If you have questions, feel free to ask us on [Piazza](http://piazza.com/brown/spring2021/csci0320)!

**GETTING STARTED**

For this project (and only this one) we're providing most of the skeleton code.

First, we recommend creating a work directory for cs032 work. For example, run mkdir ~/cs0320 to make a cs032 directory in your home directory. Then cd ~/cs0320. Now that you're in your work directory, you can download the boggle zip file [here](http://cs.brown.edu/courses/cs0320/projects/boggle.zip). If you are completing Boggle from home, you can still download the boggle file.

Once you've downloaded and unzipped the file, dig around for the Java files and start exploring. You should rename the student directories to your username. On a department machine, run:

mv boggle/src/main/java/edu/brown/cs/student boggle/src/main/java/edu/brown/cs/$USER

mv boggle/src/test/java/edu/brown/cs/student boggle/src/test/java/edu/brown/cs/$USER

(on a home machine, substitute your login for $USER manually).

Then make the associated changes to package names as well. Here's a handy shell command (for Mac) that looks through all files contained in your current directory and replaces any instances of "student" with your username:

grep -rl 'student' . | LC\_CTYPE=C xargs sed -i '' "s/student/your-login/g"

Use this version of the command for Linux (department machines):

grep -rl 'student' . | xargs sed -i "s/student/`whoami`/g"

Make sure you only run this command once you're inside your boggle directory, otherwise you'll end up searching through your entire home directory. NOTE: This command only works for Linux and Mac (OSX).

Alternatively, you can use the following command to find all the files containing "student", and then edit them by hand.

find boggle -name "\*.java" | xargs grep student

Keep the basic directory structure, because maven is picky about that.

**BUILDING**

We supplied a maven build file pom.xml, containing our configuration for maven. Run

mvn package

from inside the boggle directory to compile the project and produce an output .jar file (in the target/ directory). The first time you run mvn package, you'll see maven downloading a lot of libraries. That takes a while, but don't worry, that will only need to happen once.

You can run mvn clean to start fresh (which you should do after you change class or package names, for example). For more information about maven, see the useful links at the bottom of this page.

**RUNNING**

In the stencil for this project, we have provided a sample executable (./run). For all of your projects in CS32, we will expect you to turn in a similar executable. Even if you use an IDE, you must ensure that your command-line script works, as it allows the staff to run and test your code. If you take a look at that file, you will notice it's pretty straight-forward, with each line explained. Use the provided shell script run to invoke your build. For example,

./run --generate

should output a random Boggle board. The first time you use ./run it will be slow. It is figuring out which libraries need to be available for your code. It'll be faster next time.

Try invoking ./run with no extra arguments. You should get a message printed that tells provides a URL like http://127.0.0.1:4567/ or http://localhost:4567/ (which are equivalent). Visit the *play* page of the printed URL with a web browser (i.e. http://localhost:4567/play). You can enter a list of words there, but they won't be scored properly until you implement Board.play().

**REQUIREMENTS**

Here's what we've left for you to do:

1. Implement Board.play() to return all legal words for the board. Words must appear in the dictionary created at the start of the Board class, and can be formed by connecting letters vertically, horizontally, or diagonally. You may not "backtrack" or "circle around" to use the same position twice. Once you're done, a command like ./run --solve=msap,tett,yate,zvoe should print all legal words for the board made up of the given four rows of letters.
2. Test your program. In most projects, we'll ask you to write tests that show your individual classes work well (those are *unit tests*) and that your entire program works properly (those are *system tests*). We've provided some unit tests here, but we'll cover both in more depth in the first lab. For now, run mvn test to see how unit tests work. They will all succeed, but that's only because we haven't implemented a good set of tests yet. Add some tests. Two good ones are a test to see if a word is properly found in a board, and a test to ensure that a non-existent word is *not* found in a board.
3. Enhance the GUI. In later projects, you'll be creating GUIs as well, but you don't need to worry about that for now. In later labs, we'll show you how to make the boggle boards look good, implement interactive scoring and timers, and so on. We won't be assuming any experience with HTML, CSS, or JavaScript before CS32 so don't worry if you have no idea how you would do that yet.

**CHECKING FOR STYLE ERRORS**

All projects have a style component based on Maven Project reports. [Here](https://cs.brown.edu/courses/cs0320/labs/getting-started.html#checking-your-code) are some tools that we've set up for you that will pick up some simple mistakes, both in formatting and in common error-prone code patterns (also known as "anti-patterns").