Grading

The grade for this assignment will be determined by functionality (60%), design (25%), and style (15%).

FUNCTIONALITY

For this assignment, you'll navigate a **`CS15Robot`** called AndyBot through a maze—surpassing daunting obstacles such as walls and an especially trifling roadblock. Your task is to call move methods on the AndyBot to move it out of the maze (off-screen) so the “Winner!” message appears.

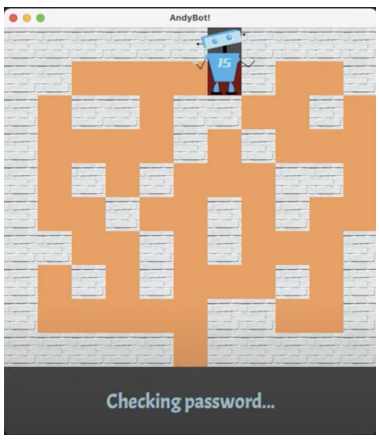
1. Don't try to move your bot into a wall because it will cause AndyBot to get stuck.

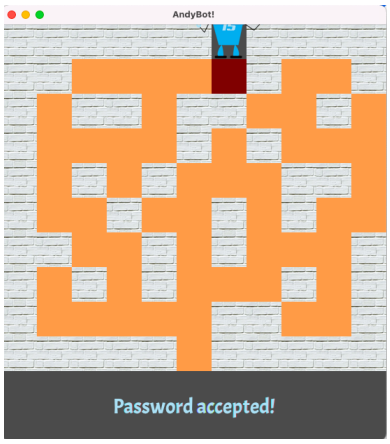
2. The red at the end of the maze represents the roadblock. To pass it, AndyBot will have to submit a secret password (it will be a number). Unfortunately, this password will be different every time you run the program. Luckily, the maze will help you if you call the right methods (**hint:** think about who has this information when calling the method)!

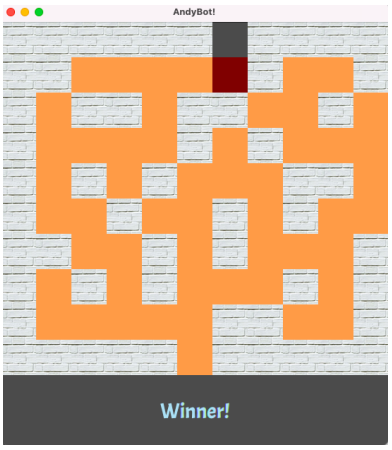
3. Once AndyBot is on the maroon square (the square directly underneath the gray square) it may enter the password. If AndyBot tries to enter the password before it reaches the square, the password will not be accepted as that is too early to input it.

4. Once AndyBot submits the correct password, make sure to move your AndyBot upwards 2 more steps and off-screen to victory. The bottom screen should read “WINNER!”

5. A successful program will match the pattern shown below:







We're providing you with a support class, **`CS15Robot`**, which is code we've already written for you (and you can use). We've also provided a stencil class, **`MazeSolver`**, for you to fill in with all your code. Before you start programming, look over the slides from the first three lectures. Make sure you understand objects and classes, and how they look in code.

Coding Incrementally

After you've watched the demo, *thoroughly read this handout and the Javadocs in full* to make sure you understand the project. Once you're ready, start coding! It is important to code **incrementally**, meaning you completely accomplish one logical task before moving on to the next one.

Compiling and Running Your Code

Refer to the Andy's Kitchen handout for more detailed instructions on how to compile and run your code. Here is a summary:

- In the IntelliJ terminal, navigate to the folder where this project is stored with the **`cd`** command.
- To compile your code, type **`javac *.java`** in your terminal
- After you compile your code successfully, move back to the **`src`** directory (**`cd ..`**). Then run your code by typing **`java andybot.App`**
- Each time you want to recompile and run your code, move back to the **`andybot`** folder with **`cd andybot`**, then repeat the 2 steps above.
    - Hint: If you're not sure which folder you're in, use the **`pwd`** (Mac) / **`cd`** (Windows) command.

Suggested Order for Incremental Coding

Step 1. Navigate to the maroon square

Step 2. Get past the roadblock

Step 3. Move up 2 more spaces to get off the board

Saving Your Work on GitHub

Refer to the CS15 GitHub guide for more detailed instructions on how to save snapshots of your work to GitHub. We recommend doing this around once an hour to make sure you're maintaining a copy of your code. Here is a summary:

1. Move into the **`andybot`** directory
2. **`git add -A`**
3. **`git commit -m "<some descriptive message>"`**
4. **`git push`**

5. Repeat!

Minimum Functionality Requirements

MF Policy Summary: *In order to pass CS15, you will have to meet minimum functionality requirements for all projects. If you don't meet them the first time around, you may hand the project in again until you succeed, but you will keep your original grade. MF requirements are* ***not*** *the same as the requirements for full credit on the project. You should attempt the full requirements on every project to keep pace with the course material. An 'A' project would meet all of the requirements on the handout and have good design and code style.*

In order to meet MF for AndyBot:

1. AndyBot must move to the roadblock.

Full Functionality Requirements

Beyond the minimum functionality requirements, the rest of the functionality grade will depend on the following criterion:

- AndyBot passes the roadblock and moves off screen, with the "Winner!" message appearing
- No other minor bugs

IMPLEMENTATION

This section of the handout has valuable details to help with the actual implementation of the assignment. Read it thoroughly for lots of helpful hints!

Support Code and Javadocs

In many early CS15 projects, you will be using support code. In a nutshell, this means that we have predefined some classes for you, and you can call methods on instances of those classes. In this project, for example, there is a support code class called **`CS15Robot`**. You don't have to edit or even see the definition for this class. However, you can tell an instance of the **`CS15Robot`** class, which is passed into **`MazeSolver`** as a parameter called **`andyBot`**, to do things by calling its methods.

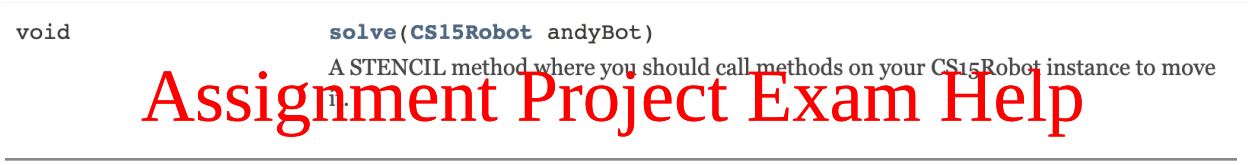
In order to see which methods are contained in the **`CS15Robot`** class, please refer to the Javadocs. Javadocs is a website that contains documentation of existing Java methods. For CS15, we have our own Javadocs that give information about the classes and methods that you'll need to use for the first few projects of the course.

The Javadocs have crucial information to understand this assignment - in fact, you will not be able to complete the assignment without some of that information, so be sure to read them carefully.

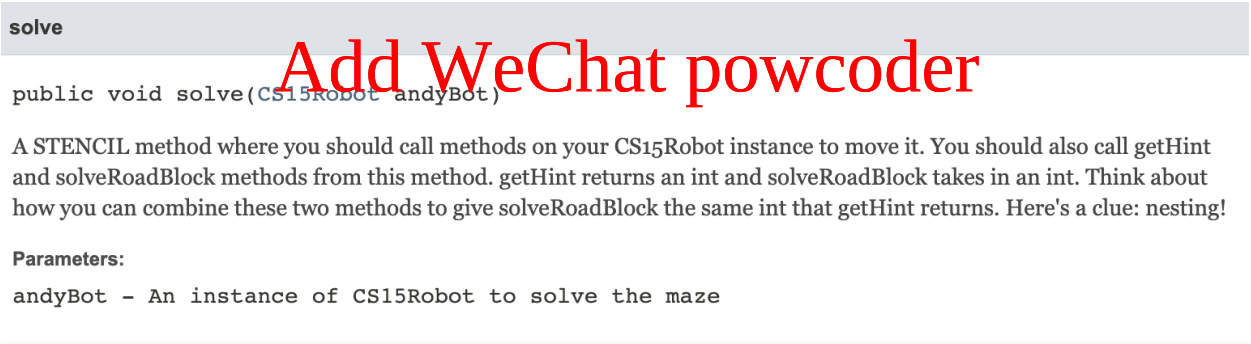
CS15's philosophy is to give you some “magic” (i.e. support code) in the beginning so that you can create rich, graphical applications in your very first assignments. This makes the projects fun and rewarding from the get-go. As you progress through the semester, however, we'll gradually peel away the magic. By the time you do the Cartoon project, you'll be using no CS15 support code!

How to Read the Javadocs

First, spend 5 minutes clicking through the andyBotSupport in the JavaDocs to get a sense of what it all means - it’s a lot to take in! Notice that there is one page for each class. Also be sure to scroll the whole way down each page. There are summaries of the method at the top, like this:



But below there is more detail, some of which will be crucial to solving AndyBot! So just be sure you read the whole thing.



Stencil Code vs. Support Code

We provide you with both stencil code and support code.

“Stencil” code or “skeleton” code refers to the classes that are the shell of the program. **You should never delete code that was given to you as stencil code — it is there to help you!** You will be adding your own code to the stencil code that we provide. For this project, the stencil class that you should fill out is **`MazeSolver`**.

On the other hand, **support code** is code that you must use, but will not be able to see or add to. In other words, we've written methods for you that you should call! Some classes (like **`MazeSolver`**) have support code methods that you should call from within those classes (using the **`this`** keyword!). There are also entire support classes that are entirely invisible to you, but you will have to use those objects (like **`CS15Robot`**)! For this project, **`MazeSolver`** has some support code methods, but **`CS15Robot`** is an invisible support class.

DESIGN

For this assignment, your "design" grade will be based on your code implementation, such as proper use of nesting method calls. This section has some helpful reminders to guide your implementation.

Calling Methods

When calling methods, we must always specify a receiver, or the object that should carry out a command. For example, if we've defined a **`Waiter`** class with an **`addPepper`** method, perhaps a **`Customer`** wants to ask their waiter to add pepper to their salad like this:


```
// this method is in the Customer class
public void eatSalad(Waiter myWaiter) {
      myWaiter.addPepper();
      // code to eat the salad
}
```


This example is like saying "Hey waiter! Add pepper please!"

Or maybe the waiter knows to always add pepper to salads before serving them, so the waiter may have code like this:

```
// this method is in the Waiter class
public void serveSalad() {
      this.addPepper();
      // code to serve the salad
}
```

This example is like saying "Hey me! Add pepper please!"

Notice that we've called the same method **`addPepper`** in two different contexts, but when we call the method from the *same* class where the method is defined (in this case **`Waiter`**), we use the **`this`** keyword.

Nesting Methods

Nesting methods is putting one method call inside of another. This is useful when we want to pass the return value of a method into another method as an argument. For example, let's say a waiter must set a table at a restaurant. First, they will need to know how many guests to set the table for!

```
//this method is in the Waiter class
public void setTable(int numberOfCustomers) {
      this.addForks(numberOfCustomers);
      this.addKnives(numberOfCustomers);
      this.addPlates(numberOfCustomers);
}
```

The waiter does not know how many people are going to be sitting at the table, but the hostess does!


```
//this method is in the Hostess class
public int getNumberOfCustomers() {
      return 5;
}
```


In the **`Restaurant`** class, we need to call the **`setTable`** method to make sure the customers have everything they need at their table. We can use nesting to pass the return value of **`getNumberOfCustomers`** into **`setTable`** as an argument!


```
//this code is in the Restaurant class
waiter.setTable(hostess.getNumberOfCustomers());
```

`hostess.getNumberOfCustomers()` will return 5, so 5 will be passed as the argument, just like if we wrote **`waiter.setTable(5)`**. Then the waiter will set the table with 5 forks, 5 knives, and 5 plates!

STYLE

Refer to the CS15 Style Guide for the specific style guidelines along which your code will be graded for the "style" portion. For this assignment specifically, be sure to use (when appropriate) the **`this`** keyword, remove **`TODO`** comments, and have proper indentation.
