AndyBot

Due Date: Thursday, September 19th, 11:59pm

Run a demo using: `cs0150_runDemo AndyBot`
To install: `cs0150_install AndyBot`
To handin: `cs0150_handin AndyBot`

Note: All of the above commands should be run in a terminal

Table of contents:

Silly Premise

Run! Jim has decided to prank Dwight the Recyclops by leaving a maze of Dunder Mifflin paper around the office. Recyclops, who's committed to saving the planet, wants to quickly get through the maze and reach the recycling bin at the end. Wanting to make the challenge even harder, Jim added a secret passcode that Dwight must provide in order to access the bin. Your challenge is to help Dwight navigate through the paper maze and get the right passcode so that he can save the environment and get promoted to Assistant Regional Manager!

Instructions

For this assignment, you'll navigate an 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.

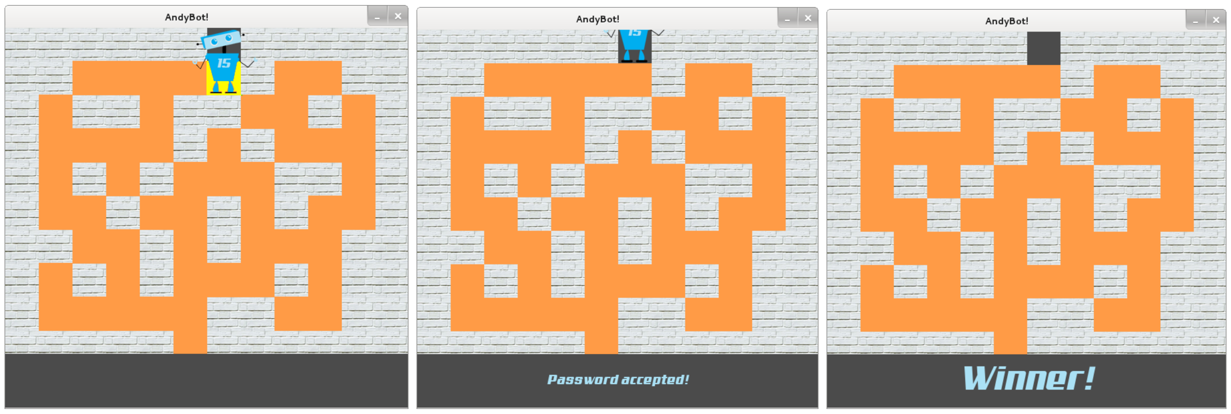
1. Don't try to move your bot into a wall because it will cause AndyBot to get stuck.

2. The grey block 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 reaches the square *before* the grey roadblock (highlighted in yellow in the image below), 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 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, `AndyBot`, 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 (more on this below). 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.

Editing your code

To edit your code, go to the terminal and type in `cd` (change directory) followed by the necessary directory you wish to go into. In this case, it is `course/cs0150/AndyBot`. Then type `atom *.java &` in your terminal. This command tells the terminal to open all the .java files in your current directory in the Atom text editor (and to run it in the background). Don't forget to save both files before you compile!

Running your code

Make sure you're in the `course/cs0150/AndyBot` directory (you can type `pwd` into your terminal to check). Compile your code by typing `javac *.java` in your terminal, and then run it by typing `java AndyBot.App`. The AndyBot program should open in another window.

README

In CS15, you're required to hand in a README file (must be named README) that documents any notable design choices or known bugs in your program. Remember that clear, detailed, and **concise** READMEs make your TAs happier when it counts (right before grading your project).

We will be providing a file for your README for AndyBot, but will expect you to create your own in subsequent projects. For this README, please explain the concept of nesting and how you used it in this project. In the future, please refer to the README guide for information on what information your README should contain and how you should format it.

Handing in your code

In order to hand in your code, run the handin script at the top of this handout. The script will list all of the files you are about to hand in, `App.java` and `MazeSolver.java`, and will prompt you to confirm. Once you've confirmed, you will receive an email stating that the handin was successful. **Note:** the email is your receipt or proof that you've handed in the assignment successfully, so please do not delete it! You can run this script as many times as you would like; however, once you run the script, all past handins for this assignment are overridden. This means that if you run the script after the deadline, the project will be marked late, even if you handed in an earlier version on time.

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 `AndyBot`. You don't have to edit or even see the definition for this class (in fact, you aren't allowed to see support code).

However, you can tell an instance of the `AndyBot` class, which is passed into `MazeSolver` as a parameter, to do things by calling its methods.

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

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!

Support Code Classes

This is a listing of the support code classes. We provide you with a description of their constructors. See the Support Code handout online for more information about what support code is and how it interacts with your stencil code.

Class: `AndyBot`

Methods:

> **`AndyBot()`**
> This is `AndyBot`'s constructor. You don't need to call this and you won't need to make a new `AndyBot`. You can and should use the instance passed into the MazeSolver's constructor.
>
> The AndyBot has four methods: **`shuffleLeft, shuffleRight, moveUp,`** and **`moveDown.`**
>
> Explanations for these methods can be found in the Javadocs. `AndyBot` will move in the specified direction for *n* steps. If `AndyBot` tries to step into a wall, it'll crash and stop moving completely, so make sure to be precise when deciding exactly how many steps to move. **Important**: these methods are slightly different than those in the lecture slides, so pay attention to the method signatures. The `AndyBot` cannot turn, it can only move up or down, and shuffle left or right.

Class: `MazeSolverSupport`

You'll notice that `MazeSolver "extends MazeSolverSupport"`. This is related to another important aspect of object-oriented programming that we'll cover in the Inheritance lecture next week. You don't need to worry about this line for the time being — just don't delete it!

Stencil Code Classes

This is a listing of the classes that you need to fill in for this assignment. **Note:** Stencil code is *not* the same as Support code. For support code, we provide you with classes that we have filled in, and you can call the pre-written methods. For stencil code, we give you a method header, but it's up to you to fill out the body of the method.

Class: `App`

Purpose: You do not need to worry about this class, and please do not modify it. This is a very important class and it is what triggers the start of the program. Every project that you create will have an `App` class, which you will learn more about as the semester progresses.

Class: `MazeSolver`

Purpose: This is the top-level class in your program. We have created an outline for this class, but you'll need to fill in the rest.

Methods:

`MazeSolver(AndyBot andyBot)`
This is the `MazeSolver`'s constructor. When a `MazeSolver` is instantiated, it is also passed an instance of `AndyBot`. We first call `super()`, which is necessary in this class for your code to interact with the support code (we'll cover what `super()` does in a few weeks!). As a general guideline, it is good to put minimal code directly in the constructor, so we call the `solve` method next, which is where the bulk of the work will happen, and pass it the instance of `AndyBot`.

`void solve(AndyBot andyBot)`
The `solve` method is where you should call methods on your instance of `AndyBot` to move it. See above in the handout for a description of the methods you can call on an instance of `AndyBot`.

In the `solve` method you should also call the `getHint` and `solveRoadBlock` methods. `getHint` returns an `int` and `solveRoadBlock` takes in an `int`. Think about how you can combine these two functions to give `solveRoadBlock` the same `int` that `getHint` returns. Here's a clue: *nesting!*

`int getHint()`
Calling this method will return a random `int`. The roadblock has a different password each time you run the program, but this `int` is an important clue that you'll need. (You can't see this method in `MazeSolver`'s file because it is part of the support code -- you do *not* need to write this method.)

`void solveRoadBlock(int x)`
This method takes in an `int`. When you call this method, pass in the `int` that is returned from the `getHint` method. In the method definition you should call `enterPassword`. Follow the directions defined in the comments in the stencil code. Comments in Atom will look like this:

```
// I am a comment
```

`void enterPassword(int password)`
Call this method to submit your password in the `solveRoadBlock` method. You'll want to pass in a mathematical expression, defined in the comment mentioned above, representing the password. This method, like `getHint`, is also defined elsewhere in the support code (again, you do *not* need to write this method). You don't need to worry about how it works, but you need to call it to get past the roadblock.

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.
2. `enterPassword(int x)` must be called from the helper method `solveRoadBlock(int x)`.