AndyBot

Due Date: Tuesday, September 22nd, 11:59pm

Run a demo using: cs015_runDemo AndyBot
To install: cs015_install AndyBot
To handin: cs015_handin AndyBot

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

Silly Premise

Gabriella and her chemistry group decided to abandon their careers in the scholastic decathlon to start East High School’s first Robotics team. They are extremely enthusiastic about this endeavor and they even got jackets! However, none of them have any programming experience, so they can’t even get it to move. The RoboCats could really use your help!

Note: You should complete Homework 1 before starting this project.

Instructions

For this assignment, you'll navigate an AndyBot through a maze—surpassing daunting obstacles such as walls and an especially trifling road block. 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 the program to terminate.

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 give you a hint if you call the right methods!

3. Once AndyBot reaches the square before the grey road block (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, you’ll be able to move your AndyBot upwards and off-screen to victory.

5. A successful program will match the pattern shown below.
We've given you the use of the AndyBot object, which is considered part of the support code for this project. We've also provided a "skeleton" class, Maze, where you'll write 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.

Editing your code

To edit your code, cd (change directory) into your course/cs015/AndyBot directory. Then type sublime *.java & in your terminal. This command tells the terminal to open all the .java files in your current directory in the Sublime text editor (and to run it in the background). Refer back to Lab 0 if you have questions on using Sublime. Don’t forget to save both files before you compile!

Running your code

Make sure you're in the course/cs015/AndyBot directory (you can run pwd 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. Again, refer back to Lab 0 if you have questions about compiling and running your code or navigating between directories.

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 Maze.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**

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 Maze as a parameter, to do things by calling its methods. Documentation below details all of our support code classes and their methods.

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!

---

**Skeleton Classes**

This is a listing of the classes that you need to fill in for this assignment.

**Class: App**

*Purpose:* You do not need to worry about this class, so do not modify it other than to add your login to the header comments. 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: Maze**

*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:*

```java
Maze(AndyBot andyBot);
```
This is the Maze's constructor. When a Maze is instantiated, it is also passed an instance of AndyBot. The constructor is where you should call methods on your instance of AndyBot to move it. See below for a description of the methods you can call on an instance of AndyBot.

In the constructor 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!

```plaintext
int getHint();
Calling this method will return a random int. The road block 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 Maze's file because it is part of the support code.)

void solveRoadBlock(int x);
This method takes in an int. When you call this method, pass in the int that you got 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 Sublime 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. You don't need to worry about how it works, but you need to call it to get past the road block.
```

---

**Support Code Classes:**

This is a listing of the support code classes. We provide you with a description of their constructors and the methods you can call on them. 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 Maze's constructor.

For the following move methods, AndyBot will move in the specified direction for \( n \) steps. If it 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 below. The AndyBot cannot turn, it can only move up or down, and shuffle left or right.

```java
void shuffleLeft(int n)
    Moves the AndyBot left \( n \) steps.

void shuffleRight(int n)
    Moves the AndyBot right \( n \) steps.

void moveUp(int n)
    Moves the AndyBot up \( n \) steps.

void moveDown(int n)
    Moves the AndyBot down \( n \) steps.
```

**Class:** MazeSupport

You'll notice that Maze "extends MazeSupport". 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!