CS 180 Project 5 -- Robots, Part 1

Project Assigned: Friday, 24 February 2012 (02/24/12)
Team Formation: Tuesday, 28 February 2012 (02/28/12)
Milestone Due: Friday, 2 March 2012 (02/02/12)
Project Due: Thursday, 8 March 2012 (02/08/12)

Overview

The first part of the Robots project is to create a simplified version of the game Robots.

Robots is an old text-based game traditionally associated with BSD UNIX. Although there are many variations, for each the gameplay revolves around a player character in a two-dimensional grid containing deadly robots. The game is turn-based, and on each turn, you select a direction in which to move (including diagonals). You can also teleport yourself to a random location (which is dangerous, because you may be teleported directly next to a robot). Each time you move/teleport, the robots move directly toward your new location. Occupying the same grid square as a robot kills you. The goal of the game is to destroy all the robots without dying, in order to move on to the next level, which contains more robots. However, you have no weapons, so to win, you must instead trick the robots into colliding with each other. If two robots collide, they produce a debris pile, which does not move and is deadly both to you and to the remaining robots.

For the first part of the project, the Robots game is simplified. Instead of a field of robots, you find yourself in a field of debris piles. There will be only one robot, which you must destroy, by tricking it into moving onto a debris pile. Also, the game will contain only one level, so if you win or lose, the game ends. Although this version is less challenging and interesting than the full version of the game, it's a good starting point, and if written carefully, almost everything from this stage can be reused in the next project, when the full game logic will be implemented.

One of the major goals for this (and the following) project is to give you experience designing and implementing a piece of software from the ground up. Accordingly, no sample code will be provided for this project; you must (get to!) write it from scratch.

Team Formation

You will be working in a team for this project. (This is mandatory. You cannot work individually.) You can form teams composed of either two or three members. If your team has only two members, your instructor may assign an additional third member (depending on how many students are without teams at the end of the team formation period). Every team should have one member (a leader) who will register the team and turn in the final project. Note that there is to be only be a single submission from a team. Make sure that the names of all team members are added as @authors when you add class-level comments to your code. Go to uft to register your team as early as possible. If you do not register a team by February 28th, you will be auto assigned to a team by your instructor. The team you form for this project will work together on all future team projects for this course.

As a final note, issues like ‘I am doing all the work’, ‘Bob is not listening to my ideas’, ‘I can’t start until Bob finishes methodA()’, and how to resolve these are part of the learning experience when working in a team.

Use the URL below for team registration:

https://spreadsheets.google.com/spreadsheet/viewform?hl=en&key=0AuXmpSQHwtbEdGhpaC1sVWF3MWpzWHJKTT1ZMURzQXc (or http://tinyurl.com/7ncdz5k)

Collaborating in a Team

Working in a team means all of you will be coding and fixing bugs. Emailing code back and forth to your teammates is often unreliable and causes unnecessary confusion. There are a few alternatives that allow you
to share code better. If you are already familiar with source control software like mercurial or svn, you are free to use them. Atlassian Bitbucket (https://bitbucket.org/) allows private sharing of both mercurial and git repositories. But if you are hearing these names for the first time, learning these tools for a two-week project may be too heavy an investment on your time.

The suggested alternative is to use Dropbox. Dropbox allows multiple users to share files. Changes made by userA is automatically sync’ed to a centralized server and everyone sharing that folder will be able to view those changes. If you decide to use Dropbox, all team members will have to install the software on their computers and designate one folder as a ‘Base’ folder. All your project-related code and documents should be placed inside this folder. Once you select ‘share this folder’, Dropbox will ask for a set of email addresses. Give the addresses of your team-members and you should be all set.

You may also decide to meet up together and work on one computer—Pair Programming is encouraged.

**Project Plan**

Your final submission should also contain a project_plan.txt file. This file should be a very small writeup with the following details for each team-member--Name, Component (method or logical work unit) worked upon, Time taken for completion.

Your instructor may talk to you about your contribution to the project.

**Project Setup**

Create a project5 folder in your cs180 folder. Save all your Java source files in this project5 folder. You will be turning in the contents of this folder when the project is completed.

**Note: Use the commands below, if needed.**

% cd ~/cs180
% mkdir project5
% cd project5
% pwd
/u/u9x/yourlogin/cs180/project5
% drjava

**Detailed Discussion**

**Board Setup**

Before each round, the positions of the debris, the board, and the player must be randomly initialized. The game board's dimensions should be 10x10 for this part of the project, **but this will change in the next project.** Additionally, 10 debris items should be added at random locations on the map. **Make sure** that each debris item is added to an empty square, i.e., don’t add debris to the same square twice. Also, after placing debris, you will need to place the player and robot in locations not occupied by anything else.

Every time the player/robot moves, the game needs to be able to check whether or not the player/robot has just moved onto debris, and accordingly died. This check will be performed often, and later on in the project, performance may become a concern, when there are many robots. So, at this stage, make sure whatever means you use to store the debris allows you to efficiently determine from a set of grid coordinates whether or not that location contains debris. This must be done without iteration, or else the operation would be too expensive when very many robots need to check for collision with debris or each other.

A multidimensional array of booleans may work well for this part of the project, as it allows easy access to the contents of a grid square given the coordinates of that square.

You might also try creating a class and instances of that class to store the Locations of the player and robot.
**Game Logic**

Each time the player enters a move, the game will need to calculate the player’s new location based on the move requested, and move the player to that new location. You need to ensure two conditions:

1. That the player cannot move off the map, and
2. That teleporting never places the player directly on the robot (though it can place him beside it).

After the player has moved, the robot will move one square towards the player. The game then needs to determine how this has changed the state of the game, as follows:

1. If the player’s location is on the robot, the game is lost.
2. If the player is on debris, the game is lost.
3. If the robot is on debris, the game is won.

**User Interface**

Instead of a GUI, the game will be displayed to the user using the System console, and the user will be prompted for input, as shown below.

```
+-------+
| *     *
|       |
|       |
| @     |
| *     *
|       *
|       *
|-------+
```

Please input a direction on the numeric keypad (0 for teleport):

In the display above, you, the player, are represented by the @ character. (This is the traditional character for text-based games like the original BSD robots. It was chosen because it sort of looks like a cowboy hat from above.) Piles of deadly debris are represented by *, and the enemy robots (just one for this project) are represented by +.

The player has a variety of moves available to him. He can move one square in any direction, and also teleport to a random location on the map, one not occupied by debris or the robot.

The controls are as follows (laid out like a numeric keypad on a computer keyboard).

**Move in a direction (5 does not move, but waits)**

```
    7  8  9
\  /  \  /  \  /
\ /   /   /   /   /
4----5----6  0-Teleport
\ / \   \   \   \
\ /  \   \   \   \
\ /\   \   \   \
1 2 3
```

After the user enters a move and presses “Enter,” the board is reprinted. If the game was won or lost, either “You
win!” or “You lose!” is displayed, and the game ends.

**Example games:**

**A loss:**

```
+--------+
|   *    |
|   @    |
| * * *   |
|   *    |
|   +    |
|   *    |
| * * *   |
+--------+
```

Please input a direction on the numeric keypad (0 for teleport): 2

```
+--------+
|   *    |
|   * @   |
|   *    |
|   +    |
|   *    |
| * * *   |
+--------+
```

You lose!

**A win:**

```
+--------+
|  +* * * |
|   * *    |
|   * @    |
|   *     |
|   *     |
+--------+
```

Please input a direction on the numeric keypad (0 for teleport): 0

```
+--------+
|  +* * * |
|   * *    |
|   * @    |
|   *     |
|   *     |
|   *     |
+--------+
```
Please input a direction on the numeric keypad (0 for teleport): 6

* * * *
* + * 
  @
* *
* *
* *
* *

Please input a direction on the numeric keypad (0 for teleport): 6

* * * *
* * *
* + @
* *
* *
* *

Please input a direction on the numeric keypad (0 for teleport): 9

* * * *
* * @
* *
* *
* *
* *
You win!

**Milestone Requirements**

For this project, you will be turning in a milestone, which has two requirements. First, you must complete the "Board Setup" section above. Second, you must display the board on the console, using the format detailed above. Game logic, win or loss conditions, player input, and robot movement are **not** required for the milestone. The milestone will be worth ten (10) percent of the project grade.

**Organization**

Try to look for a logical way to split this project up into classes and methods. Although this first phase is simple enough to do all in one class, this will change as the game logic becomes more complex. You might consider separating code which handles user input and winning/losing from the Game's logic, for instance. Also, try creating a separate class to output the view of the board to the console. This might make it easier to create a GUI later on, since you could reuse some of the same methods.

**Coding Standards**
Make sure you follow the coding standards specified in the course website at http://web.ics.purdue.edu/~cs180/Spring2012Web/java_programming_standards.html

For this project, you will be required to use Javadoc style comments on all class, method, and variable definitions. Previous projects have used Javadoc style comments in the provided sample code; however, since this project provides no sample code, you are required to create these comments yourself. You can use the above programming standards as a reference, or http://en.wikipedia.org/wiki/Javadoc.

The purpose of using Javadoc is easily creating documentation for a piece of software. Once you have Javadoc comments in your code, try running "javac *.java" on your project. You should get an HTML directory that details the layout of your project. **Successful generation of this file by the grader will count towards the coding style portion of your grade, so make sure this command completes successfully before you turnin.**

**Hints**
- Teleporting the player is very similar to placing the player initially.
- For regular moves (not teleports), there’s a sneaky way to mathematically use the number inputted by the user to produce the player's new location, without using conditionals. See if you can find it.

**Grading**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Board initialization</td>
<td>16%</td>
</tr>
<tr>
<td>Board display</td>
<td>16%</td>
</tr>
<tr>
<td>Input / Player movement</td>
<td>16%</td>
</tr>
<tr>
<td>Robot behavior</td>
<td>16%</td>
</tr>
<tr>
<td>Win and lose conditions</td>
<td>16%</td>
</tr>
<tr>
<td>Coding Standards</td>
<td>10%</td>
</tr>
<tr>
<td>Milestone</td>
<td>10%</td>
</tr>
</tbody>
</table>

**Milestone Turnin**
Make sure you have class level comments that clearly specify your name, cs login, recitation section number, and date.

`/*`
* name:
* cs login:
* recitation section
* date
`*/`

To turn in your project, first remove all the .class files and other backup files (e.g., files with names ending in ~) from your project5 folder.

`% pwd`
`/u/u9x/yourlogin/cs180/project5`
`% rm -i *.class *~`

To turn in the milestone, create a copy of your project5 directory named “project5m”: (notice the “m”)

`% cd ..`
`% pwd`
`/u/u9x/yourlogin/cs180`
`% cp -r project5 project5m`

Now run the turnin command:

`% turnin -v -c cs180=XXX -p project5m project5m`

In the table below, find the recitation section in which you are enrolled. Substitute the XXX above with the corresponding value in column XXX.

<table>
<thead>
<tr>
<th>Section Day</th>
<th>Time</th>
<th>Rec. TA</th>
<th>XXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>R01 F</td>
<td>11:30:00 AM</td>
<td>Gnanasekaran, Bala</td>
<td>REC1</td>
</tr>
<tr>
<td>R02 F</td>
<td>02:30:00 PM</td>
<td>Zage, John</td>
<td>REC2</td>
</tr>
<tr>
<td>R03 F</td>
<td>09:30:00 AM</td>
<td>Stephen, Julian</td>
<td>REC3</td>
</tr>
<tr>
<td>R04 F</td>
<td>08:30:00 AM</td>
<td>Sriran, Dinesh</td>
<td>REC4</td>
</tr>
<tr>
<td>RM1 F</td>
<td>03:30:00 PM</td>
<td>Zage, John</td>
<td>REC2</td>
</tr>
<tr>
<td>RM2 F</td>
<td>10:30:00 AM</td>
<td>Stephen, Julian</td>
<td>REC3</td>
</tr>
<tr>
<td>RM3 F</td>
<td>04:30:00 PM</td>
<td>Sriram, Dinesh</td>
<td>REC4</td>
</tr>
</tbody>
</table>

**Final Turnin**

Make sure you have class level comments that clearly specify your name, cs login, recitation section number, and date.

`/*`
To turn in your project, first remove all the .class files and other backup files (e.g., files with names ending in ~) from your project5 folder.

% pwd
/u/u9x/yourlogin/cs180/project5
% rm -i *.class *~

Change your current folder to your cs180 folder and only then run the turnin command.

% cd ..
% pwd
/u/u9x/yourlogin/cs180
% turnin -v -c cs180=XXX -p project5 project5

In the table below, find the recitation section in which you are enrolled. Substitute the XXX above with the corresponding value in column XXX.

<table>
<thead>
<tr>
<th>Section Day</th>
<th>Time</th>
<th>Rec. TA</th>
<th>XXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>R01 F</td>
<td>11:30:00 AM</td>
<td>Gnanasekaran, Bala</td>
<td>REC1</td>
</tr>
<tr>
<td>R02 F</td>
<td>02:30:00 PM</td>
<td>Zage, John</td>
<td>REC2</td>
</tr>
<tr>
<td>R03 F</td>
<td>09:30:00 AM</td>
<td>Stephen, Julian</td>
<td>REC3</td>
</tr>
<tr>
<td>R04 F</td>
<td>08:30:00 AM</td>
<td>Sriran, Dinesh</td>
<td>REC4</td>
</tr>
<tr>
<td>RM1 F</td>
<td>03:30:00 PM</td>
<td>Zage, John</td>
<td>REC2</td>
</tr>
<tr>
<td>RM2 F</td>
<td>10:30:00 AM</td>
<td>Stephen, Julian</td>
<td>REC3</td>
</tr>
<tr>
<td>RM3 F</td>
<td>04:30:00 PM</td>
<td>Sriram, Dinesh</td>
<td>REC4</td>
</tr>
</tbody>
</table>