CS180 Project 6 -- Robots Project – Part 2

Project Assigned: Saturday, 10 March 2012 (03/10/12)
Milestone Due: Monday, 26 March 2012 (03/26/12) 11:59 PM
Project Due: Thursday, 5 April 2012 (04/05/12) 11:59 PM

Team Project

Overview
The second part of the Robots project builds upon the first by completing the game logic and giving the game a GUI. The requirements of this project will be very similar to the requirements of Part 1, with a few major differences.

For the milestone, instead of a single robot chasing the player through a field of debris, the player will find himself in a field of many vicious robots, of two different kinds. His only recourse will be to trick the robots into crashing into one another and into the debris piles left from previous collisions. Once the player succeeds in destroying all the robots, he will be rewarded with a new game with even more robots (making his defeat inevitable, as in all old-school time-waster games). The major focus for the milestone will be adding to the game logic a means to handle storing the different kinds of robots, their differing behaviors, and what happens when they collide.

After having completed the game logic in the milestone, the project focuses on building a GUI for the game using Java's Swing API. The GUI will display the contents of the board, the game's status, and buttons and menus to handle player commands. The addition of the GUI should not alter the way the game is played. Accordingly, you should be able to reuse the vast majority of your code from the milestone.

Project 5 Solution
This project will depend heavily on Project 5. Accordingly, a reference solution to Project 5 is provided below. You are encouraged to continue using your implementation of Project 5, perhaps after incorporating code or ideas from the reference implementation. If you find you cannot complete this project with your implementation of Project 5, you can base Project 6 entirely off the reference implementation, as a last resort.


Project Plan
This project is a team project, so 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 project6 folder in your cs180 folder. Save all your Java source files in this project6 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 project6
% cd project6
% pwd
/u/u9x/yourlogin/cs180/project6
% drjava

Detailed Discussion (Milestone)

Board Setup
For this phase of the project, the board size will be increased to 30 rows by 45 columns, to accommodate the more complex gameplay. At the beginning of the first round, the board will be randomly filled with 10 robots. There is a 15% chance that any given robot will be a broken robot. Unlike the previous project, initially there will be no debris on the board. The player is then placed at a random unoccupied location on the board. This is level 1.

If the player wins, the level number is incremented. The board is then set up exactly as before, except the new number of robots is $10^L$, where $L$ is number of the current level. The player is then given a new random unoccupied location, as before.

If the player loses, the board is reinitialized at level 1.

Game Logic
When the player makes a move, the game will handle it exactly as in Part 1.

After the player's move, each robot needs to update its position. Make sure that no robot can make more than one move for any given turn, that is, ensure that each robot's position is changed exactly once.

The board can contain three different kinds of objects: robots, broken robots, and debris. Robots move towards the player, as before. Broken robots also move towards the player, but every fourth move, the robot will instead move one square in a random direction. You should give each broken robot a counter indicating the number of moves left until a random move. At board setup, this counter should be set to a random number between 1 and 4. Debris does not move.

The game now also needs to handle collisions much more generally. A collision occurs whenever two objects occupy the same grid space after having both of their positions updated. If either kind of robot collides with either kind of robot, both are destroyed and a debris is created. If either kind of robot collides with a debris, the robot is destroyed, and the debris remains. If the player collides with anything else, the game is lost, just as before. Note that for the last display of the board, the player's character should be displayed on top of whatever it collided with.

The game also needs to keep track of the number of robots currently on the board. The robot count should not distinguish between broken or regular robots. When the robot count is zero and the game is not lost, the game is won. The game is lost exactly as in Part 1.

User Interface
The user interface is the same as in Part 1, except below each map, print the level number and the current
number of robots. Also, since winning and losing no longer end the program, interpret a negative movement
code as a request to quit. There is no need to check whether the user’s input is valid. Also, the broken robot is
represented by a lowercase ‘x’.

**Organization**

For this part of the project, the organization of your code and data has the potential to make the solution much
easier or much more difficult, especially in a few key areas.

As in the first part, there needs to be an efficient way to check if any given tile is occupied, in order to detect
collisions. Keeping a grid lends itself to this. However, there are several kinds of objects that can move and
collide. Additionally, at least one of the objects (the broken robot) needs to keep internal state (its counter).
Although you will learn a (much) better solution to this once polymorphism is covered, for now a good way to
accomplish this is to create a class that represents a grid cell, and give it:

- a) all the instance variables that any grid object may need (i.e., the broken robot counter), and
- b) an id variable indicating what kind of object it is (e.g., robot, broken robot, debris).

You can then create a method in this class that performs the appropriate behavior based on the value of the id
field.

Also, if a grid is kept, the order of movement becomes tricky. Consider if you move objects by considering each
in a loop from top-left to bottom-right. A robot that moves down or to the right would be then considered more
than once (once to move it down/right, then again when its new position gets considered). Handling collision can
also be tricky, since collisions must be checked after all involved objects have moved.

There are many ways to solve the above problems, but a particularly clean solution is to keep two grids, one
for the way the board is now, and one for the way the board will be in the future. Moving an object in the world
would then just be copying it onto the “future board,” and collisions could be easily detected on the future board.
Then the boards could be swapped, and the old current board cleared, to become the new future board.

**Example Game**

Below is a screencast of the completed milestone being played on a terminal.
http://youtu.be/BuZM9ullbUY

**Detailed Discussion (Final)**

**Images**

The images for this part of the project can be found here:


Note: these images will need to be in the same directory as your project.

**User Interface**

The GUI will be divided into four major sections: The game board panel, the button panel, the menu, and the
status bar.

The game board panel will consist of a set of labels laid out in a grid, with each label measuring 24 pixels by 24
pixels. The background colors of the labels will alternate in a checkerboard pattern, to make it easier to precisely
determine the motion of the robots and player. Every time the board changes, the labels will update themselves
to display the icon for the corresponding entity on the corresponding grid location. The icons have been provided
in a zip file accompanying this documentation; their names should be self-explanatory (except that “robot.png”
is the regular robot, and “robot2.png” is the broken robot). To attach these icons to the labels, you may want to
look up the `<imageicon>` class. Note that the player will display a different icon depending on whether or not the
game is currently lost.
To the right of the game board will be the button panel, which will display 10 buttons arranged in a grid. The buttons will be labeled “Up Left”, “Up”, “Up Right”, “Left”, “Wait”, “Right”, “Down Left”, “Down”, “Down Right”, and “Teleport”. (See the demonstration video below).

There will be only one menu, titled “Game”. Inside this menu will be a single item, “New Game”, which reset the game back to level 1, and update the board accordingly.

Finally, the status bar should display two labels: the first indicating the current level number, and the second indicating the number of robots remaining on the board.

When the game is lost, after giving the display a final update, present a dialog box saying “Game Over. Would you like to play again?”, with the options Yes and No. If the user selects Yes, start a new game at level 1. If the user selects no, terminate the program.

When the game is one, after giving the display a final update, present a dialog box saying “You win!”, with only an OK option. When the user presses OK, start the game at the next level.

**Example GUI**

**Below is a link to a screencast of a completed version of this project being played.** Please note that a bit of leniency will be allowed for creation of the GUI, that is, it doesn’t have to look exactly like the one below. However, everything given an explicit description in the above section must be implemented precisely as specified for you to receive full credit.


**Making Real Time (Extra Credit)**

If you have extra time, you can make the game play in real time for ten percent (10%) extra credit. The key to doing this is the concept of a Timer in Swing. Swing Timers allow a snippet of code to be executed at regular intervals. For this game, every 200 milliseconds, the game will update its position based on the movement last requested by the player. This updating is referred to as a “tick”. If no movement was requested since the previous tick, the player will not move. Whenever a tick happens, the robots will calculate their moves, whether the player moves or not. However, each will only move after every five (5) ticks. Note that the player should have the opportunity to move every tick, so the robots will be five times slower than the player.

One final note about ticks: At the beginning of every level, the game should wait 500 ms (instead of 200ms) before the first tick. This gives the player a chance to examine the board before the robots begin chasing him. Trying to play the game without this additional initial delay is nearly impossible.

There is also a tricky problem that must be addressed. Holding down the buttons should result in a continuous stream of player movements, one per tick. This can be easily achieved by checking which button is currently down each tick. (You can accomplish this by attaching a MouseListener to the buttons). However, since a button click (a button-down and button-up pair) can occur in much less time than 200 milliseconds, it is possible that clicking a movement button will have no effect. This makes it difficult to precisely maneuver the player. Thus, make sure that every movement button down/up pair results in player movement, even if no update occurs while the button is down.

**Example GUI (Extra Credit)**

**Below is a link to a screencast of the extra credit version of this project being played.**

http://youtu.be/T9xZ-RWHBRQ

**Coding Standards**

Make sure you follow the coding standards specified in the course website at
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 "javadoc *.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 turn it.

Hints

- A border layout provides a convenient means to organize the three panels.
- You could try creating separate subclasses of JPanel for each of the three panels.
- There's no need to scrap the movement code you used in the milestone; the same number code could be attached to the buttons.

Grading

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Board initialization</td>
<td>10%</td>
</tr>
<tr>
<td>GUI board display</td>
<td>20%</td>
</tr>
<tr>
<td>GUI control functionality</td>
<td>20%</td>
</tr>
<tr>
<td>Collision handling</td>
<td>15%</td>
</tr>
<tr>
<td>Robot and broken robot behavior</td>
<td>15%</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 project6 folder.

% pwd
/u/u9x/yourlogin/cs180/project6
% rm -i *.class *~

To turn in the milestone, create a copy of your project6 directory named “project6m”: (notice the “m”)

% cd ..
% pwd
/u/u9x/yourlogin/cs180
% cp -r project6 project6m

Now run the turnin command:

% turnin -v -c cs180=XXX -p project6m project6m

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.

/*
 * 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 project6 folder.

% pwd
/u/u9x/yourlogin/cs180/project6
% rm -i *.class *~

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

% cd ..
% pwd
/u/u9x/yourlogin/cs180
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>