Programming Structures and Abstractions (CS 2334)
Project 5

November 19, 2009

Introduction

This project is an extension of project 4 in which we will define a new type of FinchAction, **FinchMeta**, that consists of sequences of FinchActions. This meta action will allow us to draw fractal structures with our Finch and to define action sequences that are conditionally executed and repeated.

Objectives

The objectives of this project are to:

1. implement recursive data structures and programs, and
2. use the recursive and looping nature of our FinchActions to create interesting Finch programs.

Milestones

1. Implement the **FinchTurn** action that will turn the Finch by a specified angle (10 pts)
2. Implement the **FinchJogScaled** action that will jog the Finch by a distance proportional to a "scale" parameter (10 pts)
3. Implement the **FinchMeta** action that will contain a sequence of FinchActions (25 pts)
4. Update the FinchActionDialog class and the FinchData.readText() method to accommodate the new actions (20 pts)
5. Write your own FinchAction lists (10 pts)

Other components:

- Describe the details of your new classes using a detailed UML diagram (related classes may be abbreviated) (5 pts)
- Design the "look" of your new FinchActinDialog box. Be sure to capture the layout and types of the key components (5 pts)
- Use proper documentation and formatting (javadoc and in-line documentation) (15 pts)

Note: of the design and documentation components, a total of 15 points are available during the design phase. The remaining 85 points are obtainable for the final submission of the project.

Other notes:

- You must extend the provided abstract classes. These abstract classes may not be edited. (note: exceptions may be granted by the instructor if appropriate reasons are given)
- You must write your own code for the GUI component layout (no tools may be used to automatically generate this code).
- Implement and test incrementally. Doing the implementation in the order of the milestones will help you do this.
- The example executable may be used as a guideline for the look-and-feel of your GUI. However, you may make different choices.

This lab is due in two phases:

1. Thursday, December 3^{rd} at 5:00pm: design
2. Thursday, December 10^{th} at 5:00pm: completed program and short demonstration.

More details for what to hand-in and when may be found below.

Resources

Main web page: see section on project 5

Main web page / projects / project5

- project5.pdf: this project description
- cs2334_project5.pdf: a copy of the lab section slides
- Abstract class definition: FinchMetaAbstract.java
- *.txt: example input files
- project5.jar: executable implementation of the assigned program

Example Executable

project5.jar contains an example implementation of the assigned program. To execute it, move it, finch.jar, finch_core.jar to the same folder. At the command line (unix), "cd" to this folder and execute:

```
java -cp project5.jar:finch_core.jar:finch.jar FinchDriver
```

In windows: you will also need local copies of the **dll** files. Then: "cd" to this folder and execute:

```
java -cp project5.jar;finch_core.jar;finch.jar FinchDriver
```

Note: there is also a project5_noFinch.jar file that you can use instead that does not attempt to connect to the Finch.

Input Files

We will keep the same input text file format as with the previous projects. The text files now have three additional options:

```
<name> <priority> TURN <degrees> <velocity>
<name> <priority> SJOG <duration> <left> <right>
<name> <priority> META <filter> <terminal> <condition> <loop> <scaleFactor> <minScale>
```

where:

- *filter* and *terminal* are strings,
- *condition* is one of: All, NoObstacle, LeftObstacle, RightObstacle, BothObstacle, Level, BeakUp, BeakDown, UpsideDown (more information below),
- *loop* is one of: If, IfNot, While, WhileNot, and
- *scaleFactor* and *minScale* are doubles in the range of 0 to 1.

Milestones

Milestone 1: FinchTurn

Implement class **FinchTurn** that extends **FinchAction**. This class has two properties that represent the angle to turn and the wheel velocities.

The Finches are not very accurate at turning. You may have to add a constant correction factor to your FinchAction class to achieve better turns. Add this as a class constant.

Milestone 2: Scaled Execution

Implement for all actions (existing and new):

```
public void execute(Finch myFinch, double scale)
```

The default behavior for this method is to simply call:

```
public void execute(Finch myFinch)
```

Note: think carefully about how to simply implement this default behavior.

Create a new class, **FinchJogScaled** that extends **FinchJog**. For this class only, the method:

```
public void execute(Finch myFinch, double scale)
```

will scale the specified duration by the parameter **scale** (there are no other differences).

Modify your **FinchActionDialog** to allow the user to also select this action and its parameters (and those of **FinchTurn**. Note: an instance of FinchJogScaled is also an instanceof FinchJog.

Modify **FinchActionList**: modify the execute() method to accept a **scale** parameter that is passed to all actions in the list.

Modify the **FinchData** execute() method. It should now provide a scale of 1.0 to the FinchActionList.execute() method.

Milestone 3: FinchMeta

Implement a **FinchMeta** action class that extends **FinchMetaAbstract**. Unlike our other FinchActions, FinchMeta is a "meta action": executing it results in the execution of a sequence of FinchActions. The sequence of FinchActions is defined by a **filter** string that selects actions from the **master action list**. The conditions of execution are defined by several variables.

The **conditionType** property defines (in part) the conditions under which list of actions will be executed. The possible types are:

- FinchMeta.conditionAll: always true
- FinchMeta.conditionNoObstacle: true if there are no obstacles
- FinchMeta.conditionLeftObstacle: true if there is an obstacle to the left
- FinchMeta.conditionRightObstacle: true if there is an obstacle to the right
- FinchMeta.conditionBothObstacle: true if there is an obstacle to both the left and right

- FinchMeta.conditionLevel: true if the Finch is level
- FinchMeta.conditionBeakUp: true if the Finch beak is up
- FinchMeta.conditionBeakDown: true if the Finch beak is down
- FinchMeta.conditionUpsideDown: true if the Finch is upsidedown

The **loopType** property defines the conditional (or loop) structure that is used:

- FinchMeta.loopIf: the action sequence is executed if the condition is true
- FinchMeta.loopNotIf: the action sequence is executed if the condition is **not** true
- FinchMeta.loopWhile: the action sequence is executed repeatedly as long as the condition is true
- FinchMeta.loopNotWhile: the action sequence is executed repeatedly as long as the condition is **not** true

The specific rules of execution are:

- Terminal condition: the specified **scale** is less than **minScale**. If this is the case, then the actions that match **filterTerminal** are executed with a scale of **scale * scaleFactor**.
- Recursive condition: the specified **scale** is greater than or equal to **minScale**. If this is the case, then the actions that match **filter** are executed according to the **loopType** and **conditionType**, with a scale of **scale * scaleFactor**.

Milestone 4: Add new FinchActions to ActionDialog

Add the FinchMeta class to your action dialog box. This may require reorganization of your box (I had to go to 2 columns).

Add all three new classes to your FinchData.readText() method (the name of your method may be different). FinchMetaAbstract.string2Condition() and string2Loop() will be helpful in this step.

Milestone 5: Create Your Own FinchAction Lists

Using meta actions, create (at least) two interesting recursive programs. Be ready to demonstrate these.

Hand-In Procedures

Part 1: Design

Deadline: Thursday, December 3^{rd} at 5:00pm

Each group must hand in one copy of the following:

1. a printed cover page that lists the group members, work contributed by each, and any outside citations,
2. a detailed UML design of classes FinchTurn, FinchJogScaled and FinchMeta (also include high-level details of any related classes), and
3. a design for the new FinchActionDialog box (document the layout and type of the key components).

Part 2: Complete program and short demonstration.

Deadline: Thursday, December 10^{th} at 5:00pm

Each group must do the following:

1. Turn in a printed cover page that lists the group members, work contributed by each, and any outside citations.
2. Turn in project5.zip to D2L. This is the zip file produced by Eclipse that contains:
   - Each of the class implementations with documentation. The author(s) of each class should be documented at the top of the java file.
   - html description of your project produced by javadoc (including private components).
3. A short demonstration. We will reserve time during your laboratory section for you to demonstrate your working program. You may also attend office hours or make appointments to perform the demonstrations.

General Hints and Notes

- Your program must be your own work. Do not discuss or look at the solutions of other groups in the class. However, you may discuss general issues (i.e., not directly related to the project requirements) with your classmates, as well as use the book and the resources available on the net.
- Start your work early. This is not a trivial programming assignment.
- Ask for help early. If you are stuck on something, talk to the TA or the instructor sooner than later (this is what we are here for).