CS 416 Program 3I/3P – Targets v.1

February 14, 2015

Intermediate code submission (3I) 40% of the grade:

* Due: Friday, 2/20 at 23:59 with **two late "day"**: Sa/Su (-5). Mo (-10)

Complete code submission (3P) 60% of the grade:

* Due:Thursday, February 26 at 23:59 with late days: Fr (-3), Sa/Su (-7), Mo(-12)

Goals: Use Swing buttons, labels, and sliders in a "real" application. Develop a serious test environment.

Scenario

This program simulates a simple arcade shooting game. There should be an air gun on the left of the display about half way down. A very thick *JLine* makes a good gun since it is easy to rotate it (about its left point). The angle it makes with the left side of the display is controlled by a slider located in the West panel. Angles should range from -45 to +45 degrees. The East panel should have a slider that controls the speed of the pellets shot by the air gun (between 0 and 30 pixels per frame). There should be a column of rectangular targets along the right side of the screen. These should move up and down, but the targets should pretty much stay on the screen.

The north panel should have a *JButton* that fires a pellet from the air gun and another that restarts the game. It should also have a label that shows the number of pellets remaining in the air gun. Whenever a pellet hits a target, both should be removed from the display. When a pellet leaves the panel, it is also removed. The game should end when all targets have been hit (a win for the user) or when all pellets have been fired and have left the window or hit a target. If both of those are zero, it is a *tie* and should be reported as such using a predefined *Reporter* class *endGame* method:

```
Reporter.endGame( boolean win, int pelletsLeft, int targetsLeft )
```

Application Classes

There are 4 "application" classes that you will need to implement:

AirGun: Manages the display and motion of the AirGun.

Pellet: Manages the display and motion of the pellets.

TargetStream: Manages the stream of targets that move up and down on the right of the window.

GUI: Is responsible for the user interface, **including properly handling resize requests of the user**.

There are 2 application classes that require no changes: *Targets* is the main application and *Reporter* is a utility class designed to isolate certain kinds of interactions. This version only has 2 methods: *gameOver* and *testOver*. It should be pretty clear how you use them by reading the comments in the code.

Target Layout

The targets are managed by *TargetStream.java*, which has the following package scope class variables. The targets should all be drawn so their right sides are just inside the right side of the draw panel (about 20 pixels of space). The range of the target stream should be from the top of the draw panel to the bottom. That is, when the targets are first laid out, the top of the top target should not move past the top of the draw panel and the bottom of the bottom target should not move past the bottom of the bottom space. After initial layout, the rule still applies to the positions the top and bottom targets would be at if they are still in the game.

Window resizing

The skeleton starter code is organized and has some code (but not all) in such a way that the user of your program can resize the window and you should be able to adapt your display to handle the new window – as best you can, of course. Check the video. See the code comments in *GUI.constructor()* and *GUI.resized()*

Preliminary submission / Test frameworks

The initial submission is composed entirely of test components, which, of course, require some portion of the application functionality to be working. Each test will focus on a specific set of functionalities usually for a single class, but sometimes two classes.

Note: Early next week I'll provide more details about the 3P submission point distribution.

Pellet.main: should show pellets displayed (without moving) at a variety of locations in the panel. There is "boilerplate" code in *main* that sets up a frame and panel. You need to add code to show an effective test of the *Pellet* with respect to display.

AirGun.main: has the same "boilerplate" code in its *main*. You need to add code to demonstrate that air gun objects can be generated correctly across a range of valid angles (-45 to +45 degrees).

StreamTarget.main: has the same "boilerplate" code in its *main* along with some extra code for handling resize events. You need to add code that displays a *StreamTarget* along the right side of the window and allows the window to be resized.

TestPellet: is a separate class whose job is to test the *Pellet* motion. It has the basic "boilerplate" along with a *Timer* event handler that invokes *Pellet.newFrame* for a bunch of pellets that you need to start up at the same location, but with different dx,dy parameters. This program should terminate after a fixed number of timer events have occurred. This value is defined in a class variable, *maxFrames*. Although this is a constant, you must use the variable since we will test your code with different values by changing those values in our version of the main code. This is true of most of the defined constants in the starter code.

TestGunPellet: is also a separate class, but it tests the *AirGun* and *Pellet* classes together. It's boilerplate also has a timer framework and a *maxFrames* limit. In this program you need to add code to initialize an *AirGun* with an angle somewhere near +45 and start a Pellet from the version of the AirGun. After a certain number of frames have gone by (defined by the *pelletLifeTime* variable), the *AirGun*'s angle should be updated by some *dAngle*, a new *Pellet* should be created based on the new angle and start moving. The previous pellet should just freeze. See more comments in the code and the video.

Final submission: the GUI (the game) and one more Test

TestTargetStream: should be a separate class that shows the motion of the *TargetStream*. There is no starter code, but you can build on *TestPellet.java,* which has the *Mover* framework and *StreamTarget.main*, which has the *resize* framework. Both should be include in *TestTargetStream*. However, the most important part is the animation.

GUI: is the actual application with user interaction. It is the game!

Notes

1. The starter code includes *LabeledSlider.java*, which is a basic "wrapper" around a *JLabel* and *JSlider*. All features of *JSlider* can be accessed via the method *getJSlider()* if there is something you want to do that is not provided by the *LabeledSlider* interface. **This class is in *cs416.jar*, but I'm distributing it so you can see better what it does. DO NOT CHANGE IT; we'll use ours and your code won't work.**
2. To test if a pellet has exited the window, you can use the *contains(Point)* method of *Component* (which means it is a method of *JPanel*). The best argument to use is the center of a pellet object.
3. In order to test for a "collision" of a pellet with a target, you can use the *intersects(java.awt.Rectangle)* method of *java.awt.Rectangle.* This method returns *true* if there is a non-empty intersection of the invoking object with the object passed as the parameter. The *getBounds()* method of *Component* returns a *java.awt.Rectangle*. Since *JComponent* (and *JRectangle* and *JEllipse*, etc.) inherit from *Component*, they all have a *getBounds()* method as well. Watch out for fast pellets, however. See comments in the starter code.
4. Make sure your code is well-modularized. Define appropriate classes to simplify your design; define appropriate *private* methods so that code is not replicated and to keep methods short.
5. Pay attention to the style conventions.

Intermediate submission point allocation (3I)

10 *Pellet.main*
15 *AirGun.main*
20 *TargetStream.main*
25 *TestPellets*
30 *TestGunPellets*

Submission

Submit your assignment as 3I. Do not include any J classes in your submission or *LabeledSlider*. **Don't forget your collaboration statement if submitting from terminal.** Style will **NOT** be graded for 3I.

Note: Early next week I'll provide more details about the 3P submission point distribution.