Mehran Sahami
CS 106A December 7, 2012

Additional Final Exam Review Problems

Target Seeker

Write a graphics program that allows the user to move a **filled red** "target" square on the screen by clicking the mouse anywhere on the canvas. The user can change the position of the target using subsequent clicks of the mouse. The program should also have an animated outlined (black) square representing the "seeker" that continually moves to try to reach the same position (center x and y coordinates) of the target. For example, the **Figure 1** below shows the target and the seeker. **Figure 2** shows what the screen looks like when the seeker has reached the target.

Figure 1

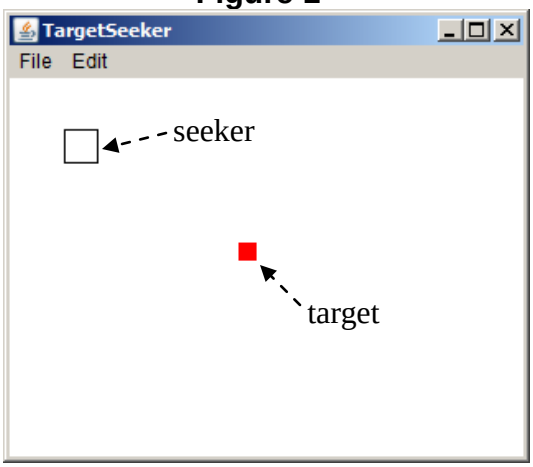
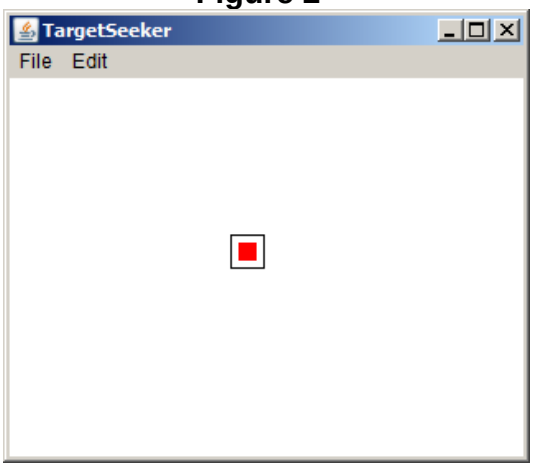


Figure 2



The program should start with the target **centered** in the graphics window and the seeker's upper left-hand corner at position (0, 0). The height and width of the target are both given by the constant **`TARGET_SIZE`**. The height and width of the seeker are both given by the constant **`SEEKER_SIZE`**. In each time step of the animation, the seeker should try to move to the same (x, y) position as the target by at most 1 pixel in the x direction and 1 pixel in the y direction. For example, if the seeker is at the same (center) x coordinate as the target, then the seeker would not update its x coordinate in each step of the animation. Your program should pause **`PAUSE_TIME`** milliseconds between each step in the animation of the seeker.

Note that the program never actually ends. If the seeker's center reaches the same location as the target's center, it just remains there until the user clicks to reposition the target somewhere else, at which point the seeker would then again start moving toward the target.

The solution to this problem is given on the following pages.

Target Seeker solution

```
/* File: TargetSeeker.java */

import acm.graphics.*;
import acm.program.*;
import java.awt.*;
import java.awt.event.*;

public class TargetSeeker extends GraphicsProgram {

	/* Contants */
	private static final int TARGET_SIZE = 10;
	private static final int SEEKER_SIZE = 20;
	private static final int PAUSE_TIME = 10;


	public void run() {
		initTarget();
		initSeeker();
		addMouseListeners();

		// Always keep seeking the target
		while (true) {
			seek();
		}
	}


	// Target is filled red square that starts in center of screen
	private void initTarget() {
		targetSquare = new GRect(TARGET_SIZE, TARGET_SIZE);
		targetSquare.setColor(Color.RED);
		targetSquare.setFilled(true);
		targetX = getWidth() / 2;
		targetY = getHeight() / 2;
		add(targetSquare,
			targetX - TARGET_SIZE/2, targetY - TARGET_SIZE/2);
	}

	// Seeker is unfilled black square that starts at origin
	private void initSeeker() {
		seeker = new GRect(SEEKER_SIZE, SEEKER_SIZE);
		add(seeker, 0, 0);
	}


	// Determine direction for seekerPos to move to get
	// closer to targetPos
	private int moveAmount(double seekerPos, double targetPos) {
		int amount = 0;
		if (targetPos > seekerPos) {
			amount = 1;
		} else if (targetPos < seekerPos) {
			amount = -1;
		}
		return amount;
	}
```

```
      // Seek target by taking a step toward its direction
      private void seek() {
            pause(PAUSE_TIME);

            // See if target is to left or right
            double seekerMidX = seeker.getX() + SEEKER_SIZE / 2;
            int dx = moveAmount(seekerMidX, targetX);

            // See if target is above or beloe
            double seekerMidY = seeker.getY() + SEEKER_SIZE / 2;
            int dy = moveAmount(seekerMidY, targetY);

            // move seeker toward target
            seeker.move(dx, dy);
      }

      // move center of target to position of mouse click
      public void mouseClicked(MouseEvent e) {
            targetX = e.getX();
            targetY = e.getY();
            remove(targetSquare);
            add(targetSquare, targetX - TARGET_SIZE / 2,
                              targetY - TARGET_SIZE / 2);
      }

      /* Private instance variables */
      private int targetX;
      private int targetY;
      private GRect targetSquare;
      private GRect seeker;
}
```

Using HashMaps

Write a method **`PrintMatchingKeys`** that is passed a HashMap containing keys and values that are both Strings (i.e., **`HashMap<String,String>`**). The header for the method is:

```
public void PrintMatchingKeys(HashMap<String,String> map)
```

Your method should print to the console all the key/value pairs in the HashMap, where the value in a key/value pair matches *any* key in the HashMap.

For example, say your method is called with a HashMap that contains the following key/value pairs:

key	value
Alice	Bob
Bob	Jeff
Mary	Pat
Pat	Ethel

It should print the following to the console:

```
Alice: Bob
Mary: Pat
```

In the example above, the value "Bob" in the key/value pair "Alice: Bob" matches one of the keys in the HashMap, so the key/value pair "Alice: Bob" is printed on the console. This is also the case for the key/value pair "Mary: Pat".

The solution to this problem is given on the next page.

PrintMatchingKeys Solution

```
public void PrintMatchingKeys(HashMap<String,String> map) {

        // Store all keys from keySet of map in an array list
        ArrayList<String> keys = new ArrayList<String>();
        Iterator<String> it = map.keySet().iterator();
        while (it.hasNext()) {
            keys.add(it.next());
        }

        // Reset "it" iterator to allow us to iterate over keys again
        it = map.keySet().iterator();

        // Check all key/values to see if value is
        // contained in ArrayList of keys (created previously)
        while (it.hasNext()) {
            String key = it.next();
            String value = map.get(key);

            // Print out the key/value pair if value is in list of keys
            if (keys.contains(value)) {
                println(key + ": " + value);
            }
        }
 }
```