P05 ESCAPE ROOM

Posted on February 8, 2019 by dahl

- **Submit your completed assignment by 9:59PM on Wednesday, March 6th 2019.** We will accept and grade submissions made through 9:59PM on Thursday, March 7th 2019. But no credit will be granted for any work that is submitted even one second after this hard deadline.
- **Pair Programming is allowed but not required for this assignment.** Register your partnership no later than Monday, March 4th to work with a partner. If you have problems accessing this form, try following the advice here.

This assignment involves developing an escape room style adventure game. This game will include graphical elements that the user can click on, and can drag onto others to solve puzzles and eventually win the game. The provided graphics and level file contain a game set in a computer center with a broken mainframe. In order to leave for the day, you must first fix the computer. Here's a screenshot of what this game may look like after you have completed this assignment.

OBJECTIVES AND GRADING CRITERIA

The primary goal of this assignment is to gain experience organizing code in an object oriented fashion that takes advantage of inheritance relationships between your classes. This will allow you to make use of the processing graphics library directly, rather than through a provided .jar wrapper.

20	Online Tests: these automated grading test results are visible upon uploading your submission. You are allowed multiple opportunities to correct the organization and functionality of your code (if necessary).
20 points	Offline Tests: these automated grading tests are run after the assignment's deadline has passed. They check for similar functionality and organizational correctness as the Online Tests. Since you will not have opportunities to make corrections after seeing

	the feedback from these tests, you should consider and test the correctness of your own code as thoroughly as possible.
10 points	Code Readability: human graders will review the commenting, style, and organization of your final submission. They will be checking whether it conforms to the requirements of the CS300 Course Style Guide. Since you will not have opportunities to make corrections after seeing the feedback from these graders, you should consider and review the readability of your own code as thoroughly as possible.

STEP 1. INTRODUCTION

Start by creating a new Java Project in eclipse and adding a new class called EscapeRoom to this project. Ensure that this new project uses Java 8, by setting the "Use an execution environment JRE:" drop down setting to "JavaSE-1.8" within the new Java Project dialog box. You will be submitting a total of seven java source files through gradescope.com for this assignment: EscapeRoom.java, Action.java, Thing.java, VisibleThing.java, ClickableThing.java, DraggableThing.java, and DragAndDroppableThing.java. You can make as many submissions as you would like prior to the deadline for this assignment, and the submission marked as "active" is the one that will be used for additional grading after the deadline.

Next download this P5Distributables.zip file, and extract the contents of this archive file directly into your P5 project folder. This should add a core.jar file, a folder of images, and a folder of rooms (containing only a single .room file) to your project folder. The core.jar file is part of the processing graphical library that you are welcome (but not required) to read more about here: https://processing.org/. If this .jar file does not immediately appearing in the Project Explorer, try right-clicking your project in the project folder and selecting "Refresh" to fix that. To make use of the code within this jar file, you'll need to right-click on it within the Project Explorer and choose "Add to Build Path" from the "Build Path" menu. Your EscapeRoom class should inherit from a class defined within this jar file called PApplet (You'll need to import processing.core.PApplet for this to work). Once this is done, you should be able to define a main method to include only the following single statement: PApplet.main("EscapeRoom"); This should result in a program that opens a small window when it is run. We won't be adding anything more to this main method for the rest of this assignment.

STEP 2. USING THE PROCESSING LIBRARY

There are JavaDocs for the processing library here, but we'll point out the specific methods that will be useful in this method throughout this write-up. To start with, there are three methods in the PApplet class that we'll override for this assignment. Note how the PApplet.setup() and PApplet.draw() methods are used in a similar fashion to the setup (called once at the beginning of our program) and update (called repeatedly while the program runs) methods from P2 Particle Fountain.

```
1 public void settings() { size(800,600); }
2 public void setup() { /* TODO: Implement this method */ }
3 public void draw() { /* TODO: Implement this method */ }
```

Add a private instance field of type PImage (imported from processing.core.PImage) named backgroundImage to your EscapeRoom class, initialize this field using PApplet.loadImage() method from setup() to load "images/computerCenter.png", and then draw it to position (0,0) at the beginning of the draw() method by using the PApplet.image() method. Running your program should now result in a window showing the gray-scale image of a computer center depicted above (without any of the fun interactive objects).

STEP 3. THE FOUNDATION: ACTIONS AND THINGS

Start by creating a new class called Action to represent the response to an object being clicked or dragged onto another. We'll expand the capabilities of this class later. But for now the only action that it will be capable of is printing out a message. Here are the instance fields and methods (including constructor) that you should implement for this class.

```
1 private String message; // message printed by this action (or null to do nothing)
2 public Action(String message) {} // initialize this new action
3 public void act() { } // when message is not null, message is printed to System.out
```

Next, create a new class called Thing to organize the capabilities that are common to all interactive things in our game. Here are the instance fields and methods (including constructor) that you should implement for this class.

```
1  private final String NAME; // the constant name identifying this object
2  private boolean isActive; // active means thing is visible and can be interacted with
3
4  public Thing(String name) {} // initialize name, and set isActive to true
5  public boolean hasName(String name) {} // returns true only when contents of name equal NAME
6  public boolean isActive() {} // returns true only when isActive is true
7  public void activate() {} // changes isActive to true
8  public void deactivate() {} // changes isActive to false
9  public Action update() { return null; } // this method returns null
10 // subclass types will override this update() method to do more interesting things
```

In addition to these instance methods and fields, we'll add the following static field and methods to the Thing class. These will make it easier for us to access PApplet capabilities, like loadImage() and image() from any Thing (or from any subclass of Thing in the future).

```
1 private static PApplet processing = null;
2 public static void setProcessing(PApplet processing) {} // initializes processing field
3 protected static PApplet getProcessing() {} // accessor method to retrieve this static field
```

In order to get our objects ready to use the PApplet's capabilities, be sure to call this setProcessing() once from the beginning of your EscapeRoom.setup() method's definition.

While we're in the EscapeRoom class, let's make one more addition to it. Let's add a private instance field of type ArrayList<Thing> called allThings to this class. We'll initialize it to an empty ArrayList object from setup. And then from the draw() method (after drawing the background image), we'll iterate through every Thing in this list, and call the update method on each one. If any of these Things that we call update on return a non-null reference to an Action, then we'll call the act() method on any such actions. We'll get a better sense of whether this is working in the next step, after adding visible things to our game.

STEP 4. VISIBLE AND CLICKABLE THINGS

Create a new class called VisibleThing which extends Thing, and represents a visible object with a graphical representation in our game. This class should include only the following instance fields and methods (including constructor).

```
1  private PImage image; // the graphical representation of this thing
2  private int x; // the horizontal position (in pixels of this thing's left side)
3  private int y; // the vertical position (in pixels of this thing's top side)
4
5  public VisibleThing(String name, int x, int y) {} // initialize this new thing
6  // the image for this visible thing should be loaded from :
7  // "images"+File.separator+ name +".png"
8  @Override
9  public Action update() {} // draws image at its position before returning null
10 public void move(int dx, int dy) {} // changes x by adding dx to it (and y by dy)
11 public boolean isOver(int x, int y) {} // return true only when point x,y is over image
12 public boolean isOver(VisibleThing other) {} // return true only when other's image overlaps t
```

For implementing the two isOver() methods, it will be helpful to make use of the width and height fields of the PImage class. If you would like help with the algorithm for implementing these isOver() methods, you should be able to find several resources for this online.

After implementing this class, test it out by adding a new VisibleThing to the allThings ArrayList at the end of your EscapeRoom.setup() method. To match the image above, you could try adding a

"koala" to position (350,65). When this is working, you should see your koala in the room.

Next, create a new class called ClickableThing which extends VisibleThing, and will represent objects that we want to interact with by clicking. This class should have only the following instance fields and methods:

```
1 private Action action; // action returned from update when this object is clicked
2 private boolean mouseWasPressed; // tracks whether the mouse was pressed during the last update
3
4 public ClickableThing(String name, int x, int y, Action action) {} // initializes this new obje
5 @Override
6 public Action update() {} // calls VisibleThing update, then returns action only when mouse is
```

The PApplet.mousePressed field will be helpful in determining whether the mouse button is currently pressed down. In order to return our action only when the mouse is first pressed on this clickable thing (not repeatedly for as long as the button is held down), we'll make sure that the mouse is currently pressed and was not pressed on the previous update. We'll also need to check whether the mouse is over this clickable thing. The position of the mouse can be accessed through PApplet.mouseX and PApplet.mouseY for this purpose.

Let's now test this new code by changing our koala from being a visible object to instead be a clickable object. This will require creating a new Action() that will print a message once each time our koala is clicked on. "What a cute stuffed koala!" seems like an appropriate message for this. In addition to clicking directly on the koala, be sure to experiment with clicking above, below, and on each side of the koala.

STEP 5. DRAGGABLE AND DROPPABLE THINGS

Create a new class called DraggableThing which extends VisibleThing, and will include code that allows the user to drag it around the screen. Here are the instance fields and methods (including the constructor) that should be included in this class:

```
1  private boolean mouseWasPressed; // similar to use in ClickableThing
2  private boolean isDragging; // true when this object is being dragged by the user
3  private int oldMouseX; // horizontal position of mouse during last update
4  private int oldMouseY; // vertical position of mouse during last update
5
6  public DraggableThing(String name, int x, int y) {} // initialize new thing
7  @Override
8  public Action update() {} // calls VisibleThing update(), then moves according to mouse drag
9  // each time isDragging changes from true to false, the drop() method below will be called onc
10 // and any action objects returned from that method should then be returned from update()
11 protected Action drop() { return null; } // this method returns null
12 // subclass types will override this drop() method to do more interesting things
```

It will again be helpful to use PApplet.mousePressed, PApplet.mouseX, and PApplet.mouseY to detect when the user presses down the mouse button is over this object. That action should begin the dragging of this object which will continue until the mouse button is released. While this thing is being dragged, it should be moved by the same amount that the mouse has moved between this and the previous call of the update() method (use oldMouseX and oldMouseY to track this). Each time this dragging ends, the drop method should be called as a result. Although this class' drop method doesn't do anything interesting (it simply returns null), sub classes will be able to override this method with more interesting behavior. Make sure that any action object returned from such drop() calls are returned from this class' update() method.

Test this out by adding a new DraggableThing to the allThings list in EscapeRoom.setup(), similar to how we were testing the koala. Creating a draggable thing with the name "key" at position (250,170) to accomplish what is shown in the image above. You should then try dragging this object around the room, while the program runs.

Now create a new class called DragAndDroppableThing which extends DraggableThing, and allows us to specify a target for this kind of thing to be dropped on along with an action that is produced when this happens. Here are the instance fields and methods that the DragAndDroppableThing class should define:

```
1 private VisibleThing target; // object over which this object can be dropped
2 private Action action; // action that results from dropping this object over target
3
4 public DragAndDroppableThing(String name, int x, int y, VisibleThing target, Action action) {}
5 @Override
6 protected Action drop() {} // returns action and deactivates objects in response to successful
7 // When this object is over its target and its target is active:
8 // deactivate both this object and the target object, and return action,
9 // otherwise return null
```

Remember that this drop() method we are overloading is called whenever this objects is done being dragged. We just need to check whether this object is over it's target and whether it's target is active at that time. When both of these conditions are true, this method should 1) deactivate both this and the target objects, and 2) return this object's action.

To test this out, let's add a new VisibleThing to allThings from setup (like our koala and key). The name of this new thing will be "chest" and it's position can be (590,310) as shown above. Then change your key from a DraggableThing to a DragAndDroppableThing with this chest as it's target. You'll also need an Action, one with the message "Open sesame!" will be helpful. Make sure that dragging this key onto this chest produces the expected output.

STEP 6. ADDING AND REMOVING OBJECTS

Let's expand our Action class, so that it can activate things. Add a private Thing instance field called thing to the Action class, and add two new constructors: Action(Thing thing) and Action(String message, Thing thing) so that new Actions can be constructed with either a String message, a Thing to activate, or both. Then update the act() method to take an ArrayList<Thing> as an input parameter. When act is called on an action that has a non-null thing field, three things should happen: 1) the activate method of that thing should be called, 2) that thing should be added to the array list passed in as a parameter, and 3) this action's thing field should be changed to null (so that each thing is only ever activated once). Since this change temporarily breaks our EscapeRoom's draw() method, we'll need to modify our call of act() there to pass its allThings array list as an argument.

Now test this out by creating a visible thing with the name "phone" within EscapeRoom.setup(). Instead of adding this thing to the allThings list, deactivate it before creating an action that will activate it when the key is dragged over the chest. Then make sure that this visible object appears when this happens.

Now that we have a way for things to be activated, let's remove deactivated objects from our allThings array. This will be the job of our EscapeRoom's draw() method. After calling update on every thing in allThings, our EscapeRoom's draw() method should search through and remove all deactivated objects from the allThings array list. After implementing this, you should see the key and chest disappear after the key is dragged onto the chest.

You now have code in place that will enable you to create a variety of different puzzles for a many different kinds of adventure games. Creating these puzzles and games will require fun ideas and corresponding graphics. Rather than hard-coding the things for a specific puzzle into our EscapeRoom's setup() method,let's load the background image filename, and introductory text message, and thing descriptions from a text file. Copy the following method definitions for loadRoom() and the accompanying helper methods into your EscapeRoom class. Then call loadRoom("rooms"+File.separator+"computerCenter.room") from setup(), instead of directly loading the background image or creating any things objects from there.

```
1  /**
2   * This method loads a background image, prints out some introductory text, and then reads
3   * a set of thing descriptions from a text file with the provided name.  The image is store
4   * in this.backgroundImage, and the activated things are added to the this.allThings list.
5   *
6   * @param filename - relative path of file to load, relative to current working directory
7   */
8  private void loadRoom(String filename) {
9    // start reading file contents
```

```
    Scanner fin = null;
    int lineNumber = 1; // report first line in file as lineNumber 1
    try {
      fin = new Scanner( new File(filename) );

      // read and store background image
      String backgroundImageFilename = fin.nextLine().trim();
      backgroundImageFilename = "images"+File.separator+backgroundImageFilename+".png";
      backgroundImage = loadImage(backgroundImageFilename);
      lineNumber++;

      // read and print out introductoy text
      String introductoryText = fin.nextLine().trim();
      System.out.println( introductoryText );
      lineNumber++;

      // then read and create new things, one line per thing
      while(fin.hasNextLine()) {
        String line = fin.nextLine().trim();
        if(line.length() < 1) continue;

        // fields are delimited by colons within a given line
        String[] parts = line.split(":");
        Thing newThing = null;

        // first letter in line determines the type of thing to create
        if(Character.toUpperCase( line.charAt(0) ) == 'C')
          newThing = loadNewClickableThing(parts);
        else if(Character.toUpperCase( line.charAt(0) ) == 'D')
          newThing = loadNewDragAndDroppableThing(parts);

        // even deactivated object references are being added to allThings, so they can be fo
        // these deactivated object references will be removed, when draw() is first called
        allThings.add(newThing);
        if(Character.isLowerCase( line.charAt(0) )) // lower case denotes non-active object
          newThing.deactivate();
        lineNumber++;
      }

    // catch and report warnings related to any problems experienced loading this file
    } catch(FileNotFoundException e) {
      System.out.println("WARNING: Unable to find or load file: "+filename);
    } catch(RuntimeException e) {
      System.out.println("WARNING: Problem loading file: "+filename+" line: "+lineNumber);
      e.printStackTrace();
    } finally {
      if(fin != null) fin.close();
    }
  }

  /**
   * Helper method to retrieve thing references from allThings, based on their names. If mult
   * things have that name, this method will return the first (lowest-index) reference found.
   *
   * @param name is the name of the object that is being found
   * @return a reference to a thing with the specified name, or null when none is found
   */
  private Thing findThingByName(String name) {
    for(int i=0;i<allThings.size();i++)
      if(allThings.get(i).hasName(name)) {
        return allThings.get(i);
      }
    System.out.println("WARNING: Failed to find thing with name: "+name);
    return null;
  }
```

```
75
76    /**
77     * This method creates and returns a new ClickableThing based on the properties specified a
78     * strings within the provided parts array.
79     *
80     * @param parts contains the following strings in this order:
81     *   - C: indicates that a ClickableThing is being created
82     *   - name: the name of the newly created thing
83     *   - x: the starting x position (as an int) for this thing
84     *   - y: the starting y position (as an int) for this thing
85     *   - message: a string of text to display when this thing is clicked
86     *   - name of thing to activate (optional): activates this thing when clicked
87     * @return the newly created object
88     */
89    private ClickableThing loadNewClickableThing(String[] parts) {
90      // C: name: x: y: message: name of object to activate (optional)
91      String name = parts[1].trim();
92      int x = Integer.parseInt( parts[2].trim() );
93      int y = Integer.parseInt( parts[3].trim() );
94      String message = parts[4].trim();
95      Thing activate = null;
96      if(parts.length > 5) activate = findThingByName(parts[5].trim());
97      // create new thing
98      ClickableThing newThing = new ClickableThing(name,x,y,new Action(message, activate));
99      return newThing;
100   }
101
102   /**
103    * This method creates and returns a new DragAndDroppableThing based on the properties spec
104    * as strings within the provided parts array.
105    *
106    * @param parts contains the following strings in this order:
107    *   - D: indicates that a DragAndDroppableThing is being created
108    *   - name: the name of the newly created thing
109    *   - x: the starting x position (as an int) for this thing
110    *   - y: the starting y position (as an int) for this thing
111    *   - message: a string of text to display when this thing is dropped on target
112    *   - name of thing to activate (optional): activates this thing when dropped on target
113    * @return the newly created object
114    */
115   private DragAndDroppableThing loadNewDragAndDroppableThing(String[] parts) {
116     // D: name: x: y: target: message: name of object to activate (optional)
117     String name = parts[1].trim();
118     int x = Integer.parseInt( parts[2].trim() );
119     int y = Integer.parseInt( parts[3].trim() );
120     Thing dropTarget = findThingByName(parts[4].trim());
121     if(!(dropTarget instanceof VisibleThing)) dropTarget = null;
122     String message = parts[5].trim();
123     Thing activate = null;
124     if(parts.length > 6) activate = findThingByName(parts[6].trim());
125     // create new thing
126     DragAndDroppableThing newThing = new DragAndDroppableThing(name,x,y,(VisibleThing)dropTar
127         new Action(message, activate));
128     return newThing;
129   }
```

(3/1) Note that we updated the first line within the .room file in the provided zip today, so that there is no images relative folder reference, nor the .png extension. If you have an older version of this file, please either make this edit manually, or download a fresh copy using the link above.

It's not required that you play through this provided game, but it is a nice way to test some of the functionality that you have implemented. You are encouraged (but not required) to write additional test methods of your own in a separate file, but do not submit these tests with your assigned code. One specific test that will be helpful to test is that all deactivated things are removed after a single call of EscapeRoom's the draw() method. Since this is called many times each section, it may otherwise be hard to notice if several calls are needed to remove several deactivated things.

AUTOMATED GRADING NOTE: The automated grading tests in gradescope are not using the full processing library when grading your code. They only know about the following fields and methods (referenced directly from this assignment). If you are using any other fields, methods, or classes from the processing library, this will cause problems for the automated grading tests. Such references must be replaced with references to one or more of the following:

- two fields within PImage: int width, and int height.
- three fields within PApplet: int mouseX, int mouseY, and boolean mousePressed.
- four methods within PApplet: PImage loadImage(String), void image(PImage,int,int), void size(int,int), and void main(String)

SUBMISSION

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the course style guide, you should submit your final work through gradescope.com. Your score for this assignment will be based on your "active" submission made prior to the hard deadline of **9:59PM on Thursday, March 7th.**

EXTRA CHALLENGES

Here are some suggestions for interesting ways to extend this simulation, after you have completed, backed up, and submitted the graded portion of this assignment. **No extra credit will be awarded for implementing these features**, but they should provide you with some valuable practice and experience.

- Try updating your VisibleThing class to support animations. These animations will work by loading several numbered image files (name1.png, name2.png, name3.png, etc), and then cycle through those images to display a different one every time update() is called.
- Try expanding the possible actions that result from clicking or drag and dropping things. Some ideas of useful additions include: 1) the ability to activate multiple objects, 2) the ability to

deactivate objects (potentially many), and 3) the ability to clear the current room's contents and load a new room in it's place.

- Try creating new kinds of things for the user to interact with in different ways. Could you implement a key pad that is activated by a specific sequence of key presses, or maybe a ScalableThing or RotatableThing that can be manipulated by the mouse without any translation.
- Try creating new graphics, puzzles, and games of your own!