CMPT 270 – 2017-2018 Term 01

Assignment 6:

Model-View-Controller Architecture

Submission Deadline: 4 December 2017 at 11:55pm – Submit via Moodle!
(Note – late submissions will be accepted until 7 Dec. at 11:55pm without deduction, but no submissions will be accepted after that time)

Description

The objective of this assignment is to get experience working with the *m*odel-view-controller architecture and simple graphics and using Timers. You are to add another *v*iew to the space invaders game given to you. In particular, you are to add another window to the game that should be beside the first window, and show an invader image and text that states how many invaders are left to be destroyed. You are also expected to modify the game to allow multiple laser fires with a recharge delay.

Deliverables

***** Upload .java files, .class files and .txt/.pdf files for documentation within one .zip file *****

As usual, you should use Moodle to submit a jar file for your complete system. Be sure to include the .java files, the .class files, and all the other files needed to run your system (the properties file, and the .png files) so that the jar file is self-running. As will be mentioned below, the data files might need to be placed in two places. If so, make sure that they are in the correct places.

Part 1: New Window specifications

You will add another window to appear beside the main game window. This window should appear once the game playing is started, and disappear once the game playing stops. Thus, the extra window should not be present when the welcome, high scores, or save scores views are present.

The new window is to display an image for an invader. There are two images for an invader — one with arms up and one with arms down. You should switch between them every time an invader is killed. Note that the images are white on a black background so they only look good when shown on a black background, therefore your panel should have a black background.

The images for the game objects are stored in files. The names of these files could be hard coded into the program. However, it is a common task to change to different images, and it should be easy to do. To facilitate this, the file names are stored in a separate file. Thus, to change to different images, it is not even necessary to recompile the project. To make the file easy to change, it is a text file (SpaceInvaders.properties). As it is common to store the properties of a project in a text file, Java has a special class (Properties) to handle such files. In the game space invaders, the Properties class is used by the class PropertiesDiskStorage (of package util) to load the images of an entity of the game. Thus, you can get a list of the names of the files for the images of an invader by the following:

```
List<String> imageNames = PropertiesDiskStorage.getInstance().getProperties("invader");
```

(see line 109 of class GameObject).

Given the name of the file for an image, the image needs to be read into memory. This is a fairly time-consuming task so it should not be done unless necessary. One approach is to read all the images at the start. This will delay the start of the system (to read all the images). Also, it is common for not all the possible images to be needed, so reading all possible images can be wasteful. The approach taken is to read images as needed. Those already read are stored in a HashMap, with the file names used as keys, so that no image is read twice. The approach of using local storage to save a copy of something that is time-consuming to obtain is called using a cache. Caches are often used when accessing resources over the web, or accessing items from disk. In the space invaders game, the class to handle image access is called ImageCache that is implemented using the Singleton pattern. Thus, when an image is needed, the image can be obtained by

```
BufferedImage image = ImageCache.getInstance().getImage(imageName);
```

(see line 35 of class GraphicsPanel).

The new window is also to have a text message stating the number of invaders that remain. In order to know when this information might have changed, the window should be a GameObserver, and be added to the list of observers of the game. In order to access the count of the number of invaders remaining, a method for that purpose should be added to the GameInfoProvider interface.

You should follow the model-view-controller architecture. Thus, the controller will create and set up the new window. The classes in the game package and the view package should know nothing about the new window, except the game now needs to implement the method to obtain the number of invaders. The new class/classes should be placed in a new package.

Note that the new window should be created before the panel with the game, as Java leaves the focus in the component last created.

Part 2: Repeated Laser Fire specifications

The space invaders game only allows one laser to be fired at a time. This is too boring :(You should modify the game so that a multiple laser beams can be fired by the player. Laser guns take some time to be recharged, however, so the player should only be allowed to shoot a laser once every 0.5 seconds.

Additional Guidelines

External Documentation

For external documentation, include the standard specification of how complete and correct the project is, and the maintenance manual.

This time for the maintenance manual, include a UML diagram for your new class/classes. For each new class, include all its features in the UML diagram. For previously-existing classes, no features are needed, and only show the previously-existing classes that are directly related to the new classes (inheritance, aggregation, ball-and-socket, and uses association). Hence, many of the previous classes will not be shown at all.

NOTE

There is a complication with respect to the jar file for space invaders. As you are given a jar file for space invaders, you might not need to take any action. However, the complication is explained in case something

is changed so that you need to take action. The project uses the text file SpaceInvaders.properties and several image files (in a directory called images). In order to be able to read these files, the files need to be placed where the project can find them. The complication is that when running the project within Eclipse the files are expected to be one place, and when running it as an executable jar file built within Eclipse, the files are expected to be in another place. Therefore, you may need to put the text file and the image directory in two places (if they are not already there). The recommended operation to use to read an image from a file is:

```
URL url = ClassLoader.getSystemResource(fileName);
BufferedImage image = ImageIO.read(url);
```

(see line 56 of class ImageCache).

A file to be used by a class is called a resource. A class loader is an object to load the compiled classes of the project at run time. The static method ClassLoader.getSystemResource looks for the file specified by fileName on the search path used to load classes. Thus, when running the project in Eclipse, the directory with the images must be in the same directory as the .class files. The text file is similar except that the read looks like:

```
InputStream propertiesStream = ClassLoader.getSystemResourceAsStream(propFileName);
properties.load(propertiesStream);
```

(line 76 of PropertiesDiskStorage of package util)

because the read is done from an input stream. The key aspect is that Java is still looking for the file in the same directory as the .class files. Assuming that you are storing the source (.java) files separate from your compiled (.class) files, the source files will be in the src directory, and the .classes files will be placed in an invisible bin directory. The method ClassLoader.getSystemResource looks for the images directory in the same directory as the .class files, so you must put the images directory and the properties file in the bin directory. You can use the import operation to do this as the Browse button links to the File System that knows of and will show the bin directory. Note that since the images are stored in a subdirectory, the fileName string must include the directory name and a "/" between the directory name and the actual file name. Unfortunately, when Eclipse creates a jar file, it does not include any non-class files from the bin directory. Hence, when you create an executable jar file for your project, Eclipse will not include the SpaceInvaders.properties file or the images directory. To have them included, they should be placed in the src directory. Therefore, the text file and the images directory should be placed in both the src and bin directories. Fortunately, dependent upon how you unpack the jar file provided, the properties file and the images directory might automatically be put in both places. Also, if you choose to have the source and class files put in the same directory, then the text file and images directory need only occur once (in this same directory).

Marking (Total 40 marks)

(8) External documentation & JAR file executing stand-alone

(8) Appropriate use of Model-View-Controller architecture for accessing game information

(14) Animation of the invader in a custom panel (or frame)

(10) Multiple lasers can be shot every 0.5 seconds

This is an individual assignment. You are encouraged to discuss the general concepts of classes, types, containers, etc. with you classmates, but the specific details of the Flight Reservation System in this assignment should be done completely individually. Students that copy / share work will be penalized.