Assignment 6

Animation, MVC and Observer Design Patterns

Date Due: December 2, 2019, 11:55pm **Total Marks: 40**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions.
- **Make sure your name and student number appear at the top of every document you hand in.** These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- **Assignments must be submitted to Moodle.**
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- **Programming questions must be written in Java.**
- **Non-programming questions must be submitted as either `.txt` or `.pdf` files.** If other file types are used, you will receive a grade of zero for that question.

Question 0 (2 points):

Purpose: Git for Bonus Marks **(optional)**

Degree of Difficulty: Easy

As a reminder, you have the opportunity to gain some bonus marks by using git/version control on this assignment. The choice to use git/version control is up to you. This is an incentive program - using git will result in bonus marks, not using git will have no effect on your final grade. You can receive up to 2% extra on your final grade (but you cannot exceed a grade of 100% in the course). Additional details and instructional videos can be found in the folder titled "Git for Bonus Marks" on moodle.

What to Hand In

- Your git-log of Assignment 6

There is a separate submission folder on moodle "Assignment 6 Git Log" for you to upload your git log. You must upload your git log here if you want to bonus marks. Uploading it to the regular Assignment 6 submission folder will result in your git log not being graded.

Evaluation

2pts git logs will be given a grade from 0-2 where:

- 0 indicates that your attempt at using was not good enough to be considered for bonus marks
- 1 indicates an attempt was made but you need to improve the number of commits or the content of your commit messages
- 2 indicates a non-trivial use of git

Question 1 (40 points):

Purpose: Practising GUI design and Implementation.

Degree of Difficulty: Tricky

The objective of this assignment is to get experience working with the model-view-controller and Observer design patterns along with simple animations using `Timers`. You are to add another view to the space invaders game given to you. Space Invaders is a fixed shooter in which the player controls a laser cannon by moving it horizontally across the bottom of the screen and firing at descending aliens. The aim is to defeat five rows of eleven aliens that move horizontally back and forth across the screen as they advance toward the bottom of the screen. The player's laser cannon is partially protected by several stationary defense bunkers that are gradually destroyed from the top and bottom by blasts from either the aliens or the player. The player defeats an alien and earns points by shooting it with the laser cannon. There are various levels of the game.

In particular for Part (a), you are to add another window to the game that should be beside the first window. This new window should show an invader image and have text that states how many invaders are left to be destroyed. In Part (b), you are also expected to modify the game to allow multiple laser fires with a recharge delay.

(a) **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 (line 109 of class `GameObject`):

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

Given the name of the file for an image, the image needs to be read into memory. This is a fairly timeconsuming 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 (line 35 of the class `GraphicsPanel`):

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

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.

(b) **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, so the player should only be allowed to shoot a laser once every 0.5 seconds.

(c) **External Documentation**

Once you have completed the questions above, create a file called `A6_documentation.pdf` that includes all the external documentation for your updated game. External documentation should include:

- A description of how to run your system. What class should be invoked, what method?
- The status of your assignment. What is working and what is not working? What is tested and what is not tested? If it is only partially working, the previous point should have described how to run that part or parts that work. For the part or parts not working, describe how close they are to working. For example, some of the alternatives for how close to working are (i) nothing done; (ii) designed but no code; (iii) designed and part of the code; (iv) designed and all the code but anticipate many faults; or (v) designed and all the code but with a few faults; (vi) working perfectly and thoroughly tested.
- 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, just include the class name in a box. Additionally, only show the previously-existing classes that are directly related to your new classes (inheritance, aggregation, ball-and-socket, and uses association). Don't be alarmed that, many of the previous classes will not be shown at all.

What to Hand In

- A file titled `A6.jar` of your complete system.
- A zip folder titled `A6.zip` that contains all the classes you created and or modified.
- A file titled `A6_documentation.pdf` containing your external documentation

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

5 pts For the jar file (should successfully run and be stand alone)

5 pts For correctly using the Model-View-Controller Architecture

15 pts For the new window and correct animation of the Space Invader image.

5 pts For correctly implementing the timer

5 pts For appropriate internal documentation in your new classes

5 pts For the external documentation