Java Programming 2 – Lab Sheet 7

This Lab Sheet contains material based on the lectures up to and including the material on concurrency.

The deadline for Moodle submission of this lab exercise is 4:30pm on Thursday 12 November 2020.

Aims and objectives

- Writing multi-threaded code
- Using Locks and Conditions

Set up

1. Download **Laboratory7.zip** from Moodle.
2. Launch Eclipse as in previous labs (see the Laboratory 3 lab sheet for details)
3. In Eclipse, select **File** → **Import** … (Shortcut: **Alt-F, I**) to launch the import wizard, then choose **General** → **Existing Projects into Workspace** from the wizard and click **Next** (Shortcut: **Alt-N**).
4. Choose **Select archive file** (Shortcut: **Alt-A**), click **Browse** (Shortcut: **Alt-R**), go to the location where you downloaded `Laboratory7.zip`, and select that file to open.
5. You should now have one project listed in the Projects window: **Lab7**. Ensure that the checkboxes beside this project is selected (e.g., by pressing **Select All** (Shortcut: **Alt-S**) if it is not), and then press **Finish** (Shortcut: **Alt-F**).

Submission material

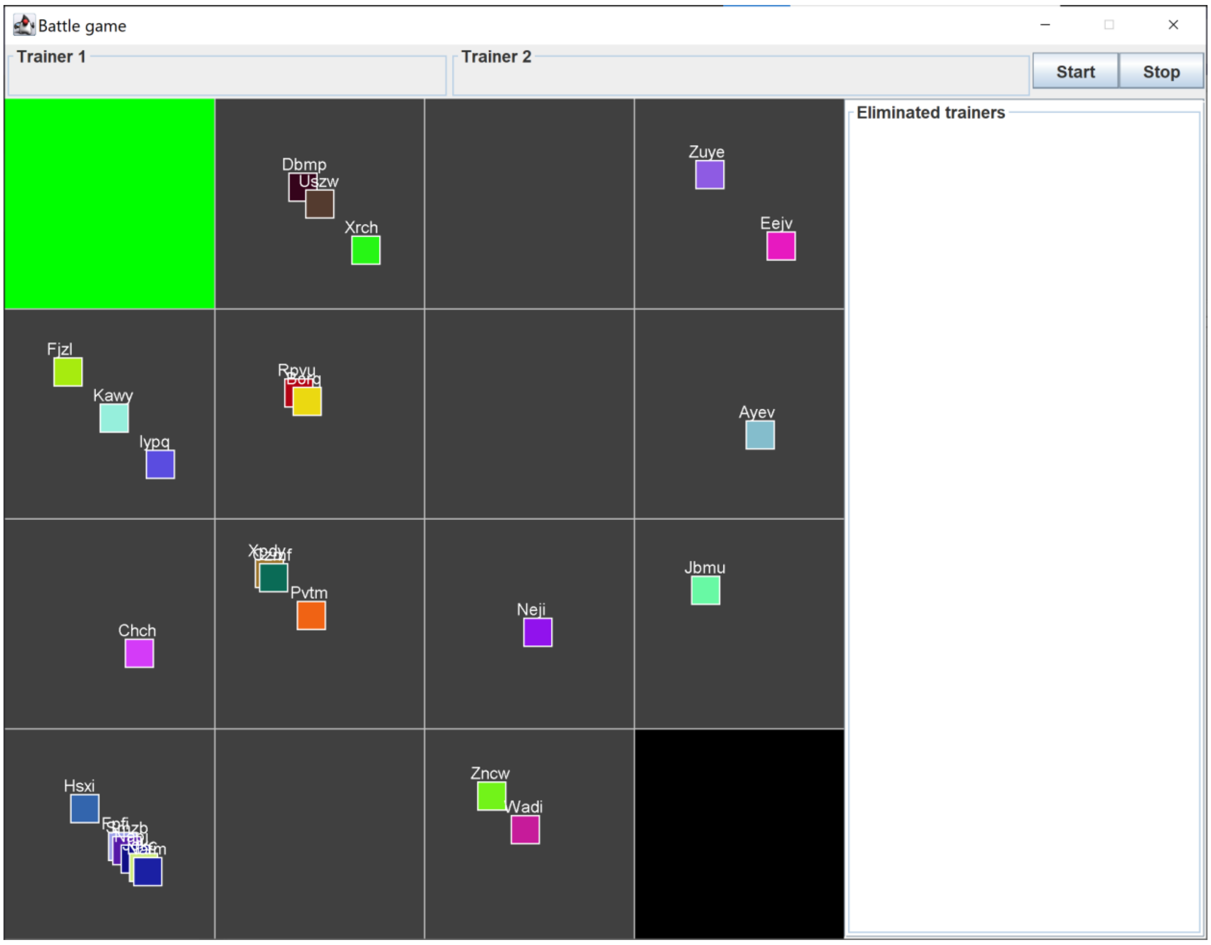
Again, this lab builds on the work we have done in previous labs. As part of the starter code, you have been provided with the classes from Lab 5 (this lab does not rely on any of the code added in Lab 6), along with additional classes that represent a very simple 2D graphics world.

In this lab, you will write methods to allow trainers to move around a battle field, and to battle each other in a special battle area. Each trainer is represented by an on-screen "sprite"[1], and each trainer sprite runs in its own thread. Access to the battle area is controlled by a Lock and an associated Condition.

You will first update the sprite class to allow the trainers to move around the screen, and then will add additional code to support the battle behaviour. I will first describe how the behaviour is expected to work at a high level. After that, I will describe the exact methods that should be implemented or modified to support the required behaviour.

[1] https://en.wikipedia.org/wiki/Sprite_(computer_graphics)

When you run the **GameWindow** class, it produces a window that looks like the following:



Each of the coloured squares represents an individual Trainer object; the label above the square is the first four characters of the trainer's name. The trainers used in this program are randomly generated – each has a randomly selected name and a randomly generated set of monsters.

Each trainer should move around the board randomly. Most of the board areas have no special properties; however, the green square at the top left (the "battle area") and the black square at the bottom right (the "resting area") are special, as described below.

- No trainer can enter the resting area unless it is moved there because of losing a battle
- At most two trainers can be in the battle area at any time
- When a trainer attempts to enter the battle area, the following happens:
    - If there are already two trainers in the battle area, the trainer will wait until they have left and then will enter
    - If there is no trainer waiting in the battle area, the trainer will enter the area and wait for an opponent
        - In this case, the trainer's name should appear in the "Trainer 1" field while they are waiting
    - If there is already a trainer waiting in the area, then the second trainer will enter the area and the two trainers will battle
        - In this case, the newly arrived trainer's name should be added to the "Trainer 2" field

  - After a battle finishes, the following updates are made:
    - The “Trainer 1” and “Trainer 2” fields are cleared out
    - The losing trainer is moved to the resting area (the black square) and stays there for the rest of the game
    - The losing trainer’s name is added to the “Eliminated trainers” list
    - The winning trainer is removed from the battle area and begins moving around the board again

Part 1: Making the Trainer sprites move

The on-screen squares are represented by the **TrainerSprite** class, while the game board is represented by the **GameWorld** class. Your first task is to modify these two classes to make the trainers move around the board. At this point, you don’t need to worry about the battle area or resting area.

First, you need to update **TrainerSprite** so that it implements the **Runnable** interface, so that each sprite can be run in a thread. The **run()** method should have the following structure:

- In an infinite loop (i.e., **while(true)**) …
  - Choose a new location to move to (use **getNewLocation()**)
  - Move to the selected location (use **setLocation()**)
  - Pause for 250ms (use **Thread.sleep()**)
- If the thread is interrupted at any point (i.e., via **InterruptedException**), you should **break** the while loop

You also need to implement the **startSprites()** and **stopSprites()** methods in **GameWindow** so that they start and stop the trainers from moving, respectively.

- **startSprites()** should create a **Thread** object for each sprite and then **start()** that thread
- **stopSprites()** should **interrupt()** each running sprite thread

Note that you will need to add a field to **GameWorld** to store the collection of **Thread** objects created by **startSprites()** so that they can be interrupted in **stopSprites()**.

Take care that things work properly if the user clicks “Start” or “Stop” multiple times in a row – you should not have more than one thread active at a time for each trainer, and should not crash if either button is pressed repeatedly.

After you have implemented this behaviour, the window should work as follows:

- When you click “Start”, the trainer boxes begin moving randomly around the screen
- When you click “Stop”, the boxes stop moving
- You should be able to click the two buttons repeatedly and this behaviour should always work

There is a video on Moodle showing the expected window behaviour after Part 1 has been completed.

Part 2: Implementing the battle area

Once you have the trainers moving around correctly, the second step is to implement the battle area as described above. Here are the steps to make this work.

Updating TrainerSprite.run()

You need to update the **run()** method of **TrainerSprite** so that it correctly deals with the battle area. The changes should be as follows

- Change the condition for the **while** loop so that it runs as long as **battleReady** is true
- After getting a new location from **getNewLocation()**, you need to do the following:
    - If the new location is the resting area (use **world.isRestingArea()**), the trainer should not move – that is, the trainer should just stay in its current location in this case
    - If the new location is the battle area (use **world.isBattleArea()**), then you should call **enterBattleArea()** instead of moving to the location
    - Otherwise, the trainer should still move to the target location as usual

Implementing GameWorld.enterBattleArea()

The final step is to implement the battling code in the **GameWorld.enterBattleArea()** method. The header of this method has already been provided for you, and I have also provided the necessary code to obtain and release the lock at the start and the end of the method. Your task is to fill in the body of the method to implement the battling process.

The first step in the method after the lock is obtained should be to update the location of the sprite to (0, 0). The next steps depend on the value of **waitingSprite**:

- If **waitingSprite** is null, then there is no trainer currently waiting. You should set **waitingSprite** to the current sprite, update the value of the "Trainer 1" field with **setTrainer1 ()**, and then call **await()** on the condition – this will make the trainer thread wait until another trainer enters the battle area.
- If **waitingSprite** is not null, this means that there is a trainer already waiting, so you should make the two trainers battle, as follows:
    - Set the value of the "Trainer 2" field with **setTrainer()**
    - Call **Trainer.doBattle()** on the two trainers
    - Make the current thread sleep for 1000ms using **Thread.sleep()**
    - Figure out which of the two sprites corresponds to the winning trainer, and which to the losing one
        - Call **setBattleWinner()** on the winning trainer sprite
        - Call **setBattleLoser()** on the losing trainer sprite
        - Add the losing trainer to the eliminated list with **addEliminatedTrainer()**
    - Reset the variables so that new trainers can enter the battle area
        - Set **waitingSprite** to null
        - Clear out the "Trainer 1" and "Trainer 2" fields
    - Finally, call **signal()** on the condition to wake up the thread corresponding to the other trainer sprite

There is a video on Moodle showing the expected behaviour after Part 2 has been completed.

Testing your code

There are no test cases provided for this lab, as testing multithreaded code is very complex. You can test your code by using different number of trainers (change the value in the **GameWorld** constructor inside **GameWindow** – the default value is 25) and/or changing the speed to be faster or slower (change the arguments to the various **Thread.sleep()** method calls). I strongly recommend including **System.out.println()** statements throughout your code so that you can trace the behaviour of the threads as you develop the behaviour. You can leave the **println** statements in the submitted code if you want – it might help your tutor to understand the behaviour when testing.

How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not. Before submission, make sure that your code is properly formatted (e.g., by using **Ctrl-Shift-F** to clean up the formatting), and also double check that your use of variable names, comments, etc is appropriate. **Do not forget to remove any "Put your code here" or "TODO" comments!**

When you are ready to submit, go to the JP2 moodle site. Click on **Laboratory 7 Submission**. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code – probably **…/eclipse-workspace/Lab7/src/game** -- and drag only the *two* Java files **TrainerSprite.java** and **GameWorld.java** into the drag-and-drop area on the moodle submission page. **Your markers only want to read your java files, not your class files.** Then click the blue save changes button. Check the .java files are uploaded to the system. Then click **submit assignment** and fill in the non-plagiarism declaration. Your tutor will inspect your files and return feedback to you.

Outline Mark Scheme

Your tutor will mark your work and return you a score in the range "Excellent" (*****) to "Very poor" (*). Example scores might be:

5*: you completed the lab correctly with no bugs, and with correct coding style

4*: you completed the lab correctly, but with stylistic issues – or there are minor bugs in your submission, but no style problems

3*: there are more major bugs and/or major style problems, but you have made a good attempt at the lab

2*: you have made some attempt

1*: minimal effort

Possible (optional) extensions

Here are some additional things to try if you want further practice with multithreaded programming and/or 2D graphics:

- At the moment, when there is only one Trainer left, they will wait in the battle area forever. See if you can work out how to detect this situation and declare that Trainer the winner.
- See if you can change the behaviour of the rest area so that a Trainer that stays there for a long enough time has their monsters healed and is able to move around again.
- Try making it so that Trainers battle whenever they run into each other, instead of needing to enter the battle area
- You could also update the graphics in a wide variety of ways. For example:
    - Change the appearance of trainers depending on their state (battling/fainted/battle ready)
    - Update the appearance of the battle area while a battle is taking place
    - Add sounds, fancier graphics, or whatever you feel like