CSC2002S Java Parallelism and Concurrency Assignment 2 2018

Concurrent Programming: Tree Growth Simulation

In this assignment, you will design a multithreaded Java program, ensuring both thread safety and sufficient concurrency for it to function well. This is an extension of the parallel computing solution that you developed in the first assignment.

1. Problem Description

You will implement a multithreaded tree growth simulator (Fig.1).

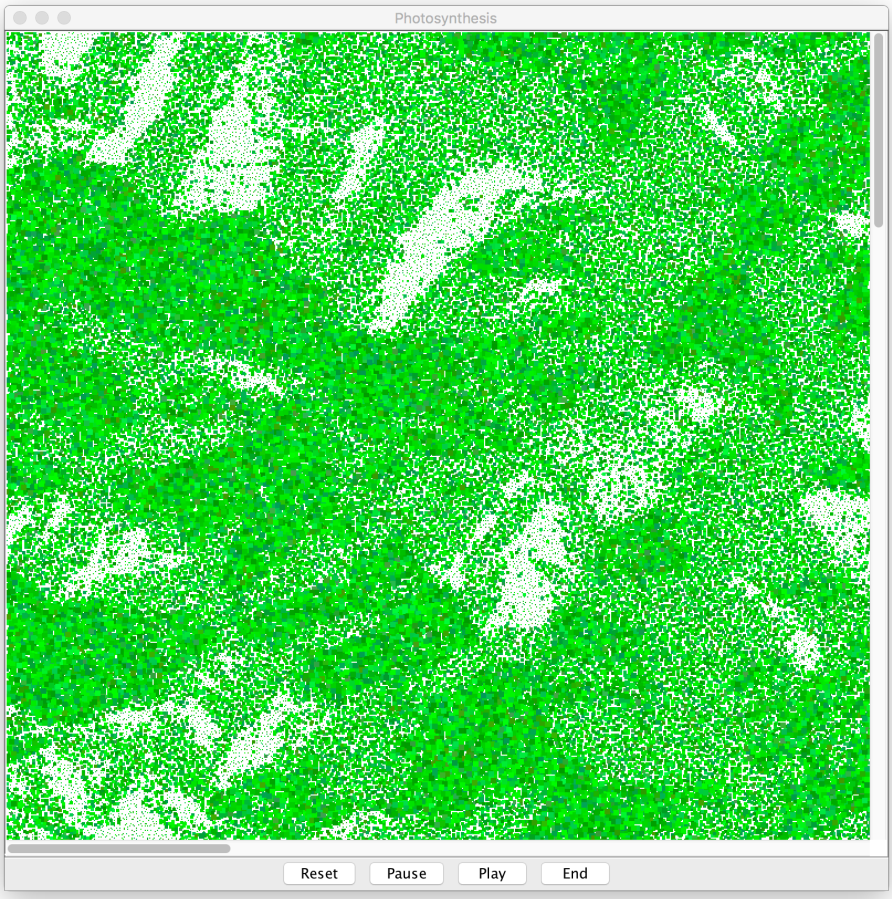

Figure 1. The main GUI window for the tree simulator. Note that this mockup is missing the required year counter.

The user interface to display the results of this simulation should have the following behaviour:

- A main display window that draws the trees as rectangles in different shades of green at correct locations in the panel. Trees are drawn in layers (all trees with an extent of [0,2) meters, then [2, 4), and so on). In this way larger trees occlude smaller trees.
- A counter that displays the number of years (generations) since the start of the

simulation. Note that this is not shown in Fig. 1.

- A ‘reset’ button that sets the extent of all trees to a value of 0.4 and the year count to 0, representing a forest of saplings.
- A ‘pause’ button that temporarily stops the simulation.
- A ‘play’ button that either starts the simulation or allows it to continue running if it was previously paused.
- An ‘end’ button that closes the window and exits the program.

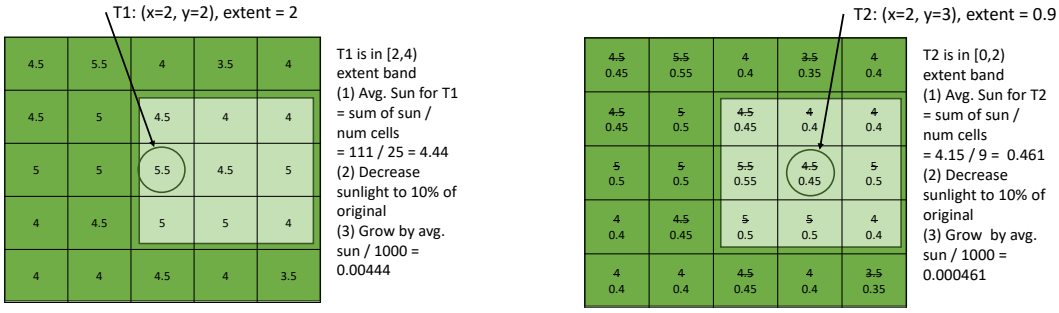

Figure 2. Sunlight simulation for two trees (T1, T2) in different layers. (Left) growth and shading calculations for T1, which takes place first because its larger extent puts it in a layer that is calculated earlier. (Right) T2 has reduced access to sunlight since it is in the shade of T1 and will grow less.

A single generation of simulation (a year) should operate as follows (see Fig. 2):

- Trees have a center position (which is where their trunk is located) and an extent (a span to either side of this central position). Thus, the trees in Fig. 2 are T1: (x, y) = (2, 2); extent = 2 and T2: (x, y) = (2,3); extent = 1, assuming that the top-left corner is (0, 0). Note that this is different from the representation in the first assignment, where T1 would be labelled as (x, y) = (0, 0); extent = 5.
- Simulation takes place in layers. First all trees with extent in the range [18, 20), then [16, 18), and so on down to [0, 2). This simulates larger trees dominating smaller trees in the forest as they compete for resources. The order in which trees grow and cast shade within a layer is unimportant. However, a tree must have mutually exclusive access to a given sunlight cell when being simulated to avoid race conditions. For example, if T3 has extent 16.5 and T4 has extent 17.2 they could be simulated in either order (because both are in the [16, 18) layer), but if they overlapped one would need to have its calculations completed before the other was allowed to start.
- At the individual tree level simulation takes place in 3 steps:
    1. Calculate the average sunlight (s) in the cells that the tree covers.
    2. Reduce the sunlight in these cells to 10% of their original value. Thus, trees in later layers will receive less sunlight. This simulates the filtering of sunlight through the canopy in a forest.
    3. A tree then grows in proportion to the average sunlight divided by a factor of 1000: newextent = extent + s / 1000.

You are provided with skeleton code for the assignment (package treeGrow). When executed, this skeleton provides an incomplete GUI interface with none of the buttons. It will display the initial state of the forest but does not include any simulation. You must build on the skeleton, improving, adding threading and ensuring thread safety when necessary. You must use appropriate synchronization and your solution should allow for maximal concurrency: operations should not be serialized unless necessary.

2. Requirements

2.1 Input

Your program must take a single command-line parameter: <intput_file>
This encodes data for the landscape and initial forest in the same format as used for the first assignment. You may reuse any code that you developed for loading such files.

2.2 Controls

Your program needs to have a reset, pause, play, and end button to control the state of the forest and simulation. There also needs to be a display of the current year. None of this functionality is available in the skeleton.

2.3 Output

The GUI needs to display the results of the simulation as it occurs. Ideally, the rendering of the forest should occur at a faster rate than the simulation to ensure that none of the detail is missed by the user

2.4 Code Architecture

When extending the code, you are expected to follow the Model-View-Controller pattern (shown in Fig. 3) for user interfaces. This very common pattern for software architecture separates the internal representation of the information from the display of the information to the user.

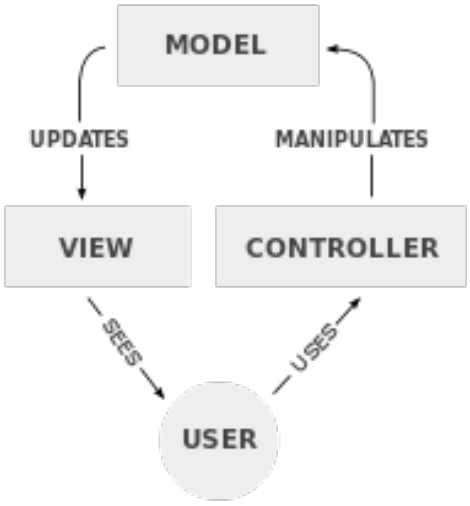

Fig. 3. The Model-View-Controller pattern has a clear separation between the display of the information (model) and its internal representation.

In this case, the model comprises the classes such as Land and Tree. The view is the GUI and the controllers will be the threads that you add to alter the model and the view, such as the simulation engine.

3. Submission

3.1 Report

You need to write a concise report detailing and explaining the coding you have done. The report must contain:

- A description of each of the classes you added and any modifications you made to the existing classes.
- A description of all the Java concurrency features you used and why they were necessary (e.g. atomic variables, synchronized classes, synchronized collections, barriers etc.).
- You will need to explain how you wrote the code to ensure:
  1. thread safety (for both shared variables and the Swing library). You should describe when you need to protect data and when you don't – and explain why.
  2. Thread synchronization where necessary
  3. liveness
  4. no deadlock.
- An explanation of how you validated your system and checked for errors (esp. race conditions).
- An explanation of how your design conforms to the Model-View-Controller pattern.
- Any additional features/extensions to (or improvements on) the basic simulation that you think merit extra credit. There are many things that you can do to improve this simulation.

3.2 Assignment Submission Requirements

- You will need to create and submit a GIT archive
- Your submission archive must consist of BOTH a technical report and your solution code and a **Makefile** for compilation.
- **Label** your assignment archive with your **student number** and the **assignment number e.g. KTTMIC004_CSC2002S_Assignment2.**
- Upload the file and **then check that it is uploaded.** It is your responsibility to check that the uploaded file is correct, as mistakes cannot be corrected after the due date.

Due date:

10am on 1st October 2018

- **Late** submissions will be **penalized at 10%** (of the total mark) per day, or part thereof.
- The deadline for marking **queries** on your assignment is **one week after the return of your mark.** After this time, you may not query your mark.

3.3 Assignment marking

Your mark will be based primarily on your analysis and investigation, as detailed in the report.

Rough/General Rubric for Marking of Assignment 2	
Item	Marks
Code – conforms to specification, code correctness, style and comments	50
Documentation – all aspects required in the assignment brief are covered	50
Total	100
Code does not execute / run	-50
Code executes but no GIT repository	-20
GIT repository but no regular updates (at least on 5 separate days)	-15

Note: submitted code that does not run will result in a mark of zero. **Any plagiarism, academic dishonesty, or falsifying of results reported will get a mark of 0 and be submitted to the university court**.