CSC2002S Assignment 2

Concurrent Programming: River Flow Simulation

In this assignment, you will design a multithreaded Java program, ensuring both thread safety and sufficient concurrency for it to function well. This builds on the problem presented in the parallel computing solution from the first assignment.

1. Problem Description

You will implement a multithreaded water flow simulator (Fig.1) that shows how water on a terrain flows downhill, accumulating in basins and flowing off the edge of the terrain.

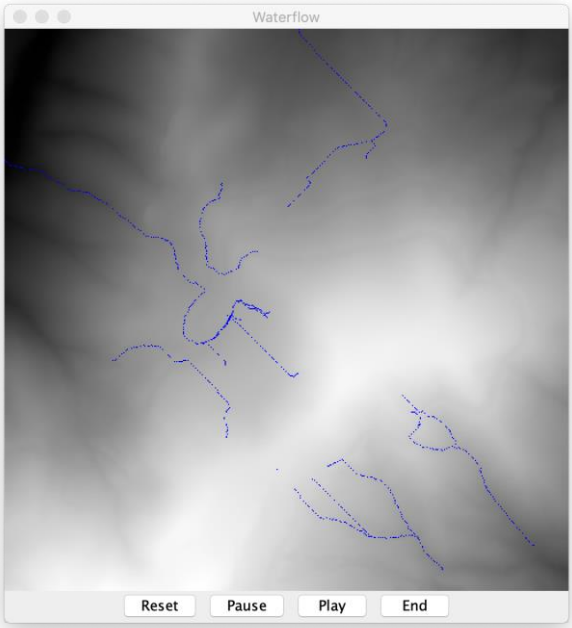

Figure 1. The main GUI window for the water flow simulator. Note that this mockup is missing the required year counter.

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

- A main display window that shows the landscape as a greyscale image, with black representing the lowest elevation and white the highest. Overlaid on this should be an image representing the locations of water in blue. Note that although the water will have a depth, represented as an integer value, the colouring is the same uniform blue for any depth value greater than zero.
- A counter that displays the number of timesteps since the start of the simulation (not shown in Fig. 1).
- A ‘reset’ button that zeroes both the water depth across the entire landscape and the timestep count .
- 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.

- Mouse input that allows the user to click on the display to add a square of water to the simulation at the corresponding position on the terrain, irrespective of whether the simulation is currently running or not.

You should create a class to represent water on the terrain, where the water depth at each terrain grid position is encoded as an integer. Each water unit corresponds to a depth of 0.01m (e.g., a water value of 5 means a depth of 0.05m). A single timestep of simulation should operate as follows:

- Water is cleared from the boundary (x=0, y=0, x=dimx-1, and y=dimy-1) by setting values there to zero. This represents water flowing off the edge of the terrain.
- All grid positions not on the boundary are traversed in a permuted order (see the *getPermute()* method in the accompanying skeleton code). This helps to reduce unevenness in the speed with which water flows across the terrain.
- For each grid position (x,y) the water surface ($s_{x,y}$) is calculated by adding water depth ($w_{x,y}$) to terrain elevation ($h_{x,y}$). Thus, $s_{x,y} = w_{x,y} + h_{x,y}$. The current water surface at (x,y) is compared to the water surface of the neighbouring grid positions. A single unit of water is transferred to the lowest of these neighbours, so long as the water surface of this neighbour is strictly lower than that of the current grid position. Otherwise no water is transferred out of the current grid position.

Your simulation should be carried out by 4 threads, each responsible for a portion of the permuted list of grid positions. These should synchronise on each timestep. That is, no thread should be allowed to start the next timestep of simulation before all others are complete. For correctness your program should exhibit "fluid conservation". Water can only be created through user mouse input and destroyed by reaching the boundary. The simulation itself is only responsible for moving water over the terrain.

You are provided with skeleton code for the assignment (package Flow). When executed, this skeleton provides an incomplete GUI interface with none of the buttons. It will display the underlying terrain but does not include any simulation or display of the overlying water. 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 in the same format as used for the first assignment.

2.2 Controls

Your program needs to have a reset, pause, play, and end button to control the state of the simulation. There also needs to be a display of the current timestep. 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 landscape and water 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.

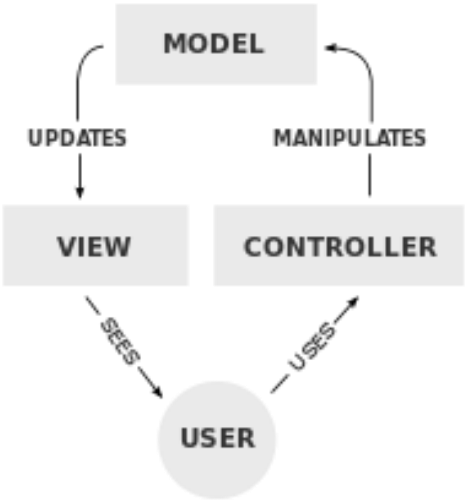

Fig. 3. The Model-View-Controller pattern has a clear separation between the display of the information (model) and its internal representation.

In this case, the model comprises the classes such as Terrain. 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, 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 game that you think merit extra credit. There are many things that you can do to improve this simulation. However, the simulation must still conform to the basic operations set out above.

3.2 Assignment Submission Requirements

- You will need to submit a GIT usage log as a .txt file (use `git log –all` to display all commits and save this as a separate file).
- 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.

Target Deadline:

10am on 14th September 2020

- Submissions will be accepted until the **hard deadline of 10am 17th September 2020** without penalty. Submissions after the hard deadline will not be accepted.
- 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.

3.1 Extensions to the assignment

If you finish your assignment with time to spare, you can attempt one of the following for extra credit:

Order traversal from lowest to highest instead of random order.
Show the depth of water using shades of blue.

3.2 Assignment marking

Rough/General Rubric for Marking of Assignment 4	
Item	Marks
Code – conforms to specification, code correctness, style and comments	25
Documentation – all aspects required in the assignment brief are covered	20
GIT usage log	2
Makefile – compile, docs, clean targets	3
Total	50

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**.