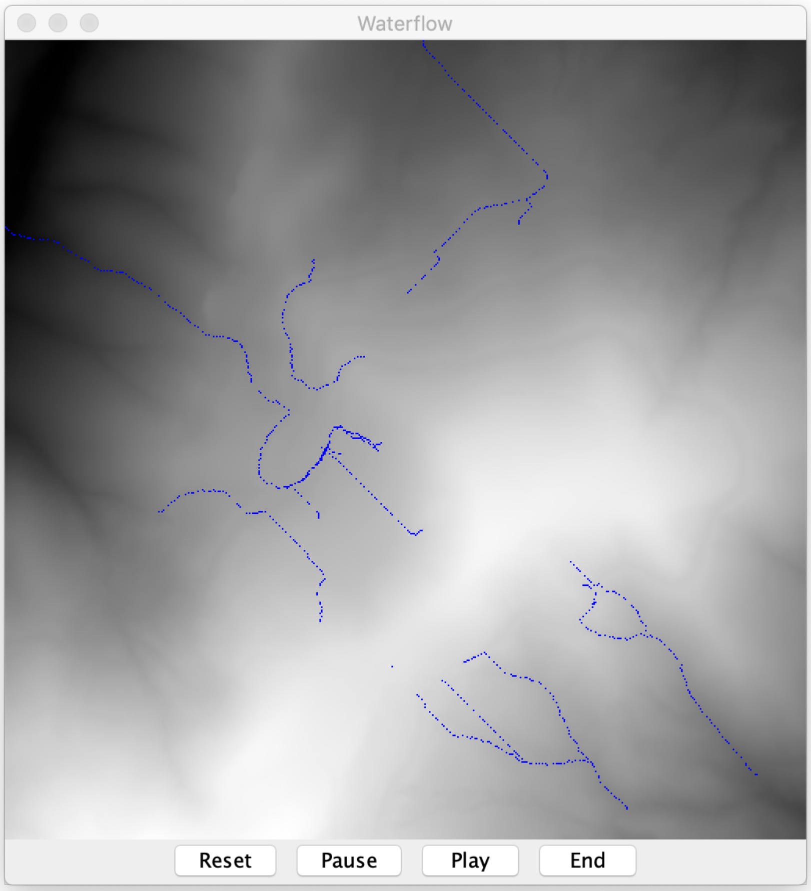
CSC2002S: Assignment 2

Concurrent Programming: River Flow Simulation



By: Keeran Bezuidenhout (BZDKEE001)

Date: 23 September 2020

Introduction

This assignment uses synchronization and concurrency to create a simulation model of rivers flowing over a given terrain. The multithreaded program depicts how the river water flows downhill, accumulates in basins as well as how the water flows off the edge of the terrain.

In this assignment, we use synchronization mechanisms such as atomic variables and the synchronization keyword from Java to control and coordinate the threads running in the simulation. This ensures there is thread safety, no deadlock and liveness while running the program with its graphical user interface (GUI).

Program classes

Classes added

- The $Water$ Class
  - Models the water in the simulation and manages water movement across the terrain.
- The $ThreadsManager$ Class
  - Organizes individual threads to get the location of the water and prepares the movement of the water to the target locations.
- The $TimerDisplay$ Class
  - Counts the timestep of the concurrent processes.

Modifications to existing classes

- The $Flow$ Class
  - The four buttons (reset, pause, play and end) required for the GUI were added.
  - The timer for the timesteps recorded added.
  - Handles user mouse clicks.

- The $FlowPanel$ Class

  - **Added Atomic variables** – an $atomicBoolean$ variable named $play$ was added. The program user can pause and play the simulation at any point in time while the program is being executed. This action affects the threads running and to ensure correctness and efficiency throughout the program.
  - **Added** $drawWater()$ $method$ **–** paints the water that appears on the surface level layer over the terrain
  - **Added** $play()$ $method$ **–** sets the $atomicBoolean$ variable $play$ to true
  - **Added** $reset()$ $method$ **–** resets the terrain landscape
  - **Added** $stop()$ $method$ **–** once the stop button is pressed by the user, this method sets play to false
  - **Added** $run()$ $method$ **–** runs the threads created in the $ThreadManager$ Class

- The $Terrain$ Class
  - No modifications were made to this Class

The Model-View-Controller pattern

This project makes use of the Model-View-Controller design pattern, where (*Water, ThreadManager*) represents the Model, (*FlowPanel, Flow*) the View, and (*Terrain, TimerDisplay*) Controller Classes.

Explanation of Program Concurrency Code

Liveness

Liveness occurs when a concurrent program can run despite having threads require mutually exclusive access to critical sections of the code. The only critical section in the program occurs in the $riverFlow()$ method when water cells need to be moved to the cell with the lowest elevation and water level.
A synchronization block was added in the $Water$ Class to ensure mutually exclusive access by threads to that critical section of the code.
Even though each thread needs to wait if another thread is executing the $riverFlow()$ method in the $Water$ Class, there is still liveness in the program because the threads will inevitably get access to that section. And that is the only section of the code that requires mutually exclusive access; therefore, the threads will otherwise move simultaneously throughout the simulation which achieves liveness.

No deadlock

Deadlock occurs when more than one thread is waiting for other threads to release the resource it requires, in order to release the resource that it possesses. This ultimately leads to a circular waiting of resources. Deadlock occurs when all of the following conditions are met:

1. Mutual exclusion
2. Hold and wait
3. No pre-emption
4. Circular wait

Conclusion

In this assignment, we see that process synchronization is a powerful tool for the coordination of processes. And to implement this project and have it function well we have ensured both thread safety and sufficient concurrency.

We also implement the Model-View-Controller pattern resulting in a robust solution where parts exhibit modular encapsulation.