20 September 2020
CSC2002S ASSIGMENT2 :CONCURRENCY
Segomotso Petlele (PTLLES002)

Introduction

This assignment is about using threads to achieve water simulation. The simulation of water running down a terrain from the terrain's highest points to either accumulating on the terrain's basins or eventually flowing of the grid. The notion of water conservation is implemented through water only being created through a mouse click and destroyed by flowing off the grid. Concurrent measures are also put in place for thread synchronization to prevent deadlock, race conditions and careful design of code to prevent bad inter-leaving. The use of the model viewer control pattern is used to hide code architecture from user by using the responsive Graphical User Interface(GUI).

Skeleton Code Modifications Plus new Class

Terrain Class:

The Color 2d array was created to store the original RGB colour form the grid. This was so that if at any grid point water depth was zero, after water flowing out of it, to turn that grid point back to its original color by accessing it through the array. Hence the getC method was created in the Terrain class to retrieve this 2d array of the original grid's colouring.

The permute array was modified to only store the internal grid points which excludes the bounds, on top of that the bounds were also stored in a separate array(bounds) so that when water reaches there, it is set to zero.

The water class to be mentioned later in this report has a its water objects created for every grid-point in the Terrain Class.

FlowPanel Class:

The FlowPanel class is basically the thread class because it implements runnable. The first change in the constructor was adding four new parameters. The first two integer parameters are linked to placing a square of water on the terrain when a mouse is pressed. The second two are for multithreading purposes and are low(for start of traversal for a thread) and high(where the traversal ends for one thread) for dividing the work up between the desired number of threads.

The run method is responsible for water surface comparisons, water transfers and for managing the repainting of the grid when grid point has water and for when water has flowed from grid.

An atomic Boolean is created called play which set a condition to true for playing the simulation. Another atomic Boolean called pause sets the same condition in the run method to pause simulation.

Flow Class:

The Flow class creates an additional three threads with the fp object being used as the first original thread. A mouse listener is created for putting blocks of water on position clicked on the terrain. Three buttons are created, being play(to play simulation), pause(to pause simulation) and reset(to reset the simulation).

Water Class:

The Water class is the additional class created. It is responsible for making water objects that have a unique id(for concurrency issues), water surface value and water depth value.

Design and Implementation

The Model View Control pattern was used for the simulation. The classes present in this design are the above-mentioned classes. The controller are the threads created from the FlowPanel class. The view comprises of four buttons being pause, play, reset and end. On screen, you'll see the terrain image and simulation of water flowing from the grid through mouse clicks. One mouse click fills a grid up with a water depth of 15 units. The reset button clears water from the grid and sets all water to zero. The play button allows the simulation to begin provided there's water on the Terrain. The pause button pauses simulation. The end button closes window.

Concurrency Implemented

The water class uses synchronized locking on its getDepth method in order to prevent incorrect depth results from other threads either incrementing or decrementing the water depth value. This protects against race conditions. The getSurface method also uses synchronize locking to avoid race conditions and for threads reading stale data that is being modified by other threads. The method transferWater is synchronized internally. The Water class has an atomic integer for its water ID to avoid two water objects being created from having the same ID. This is to protected against deadlock when it comes to transferring a water depth unit from one object to another object using the transferWater method. As a result, the transfer method synchronizers and locks depending on which water object has the highest water ID all to avoid deadlock when water is being transferred to other water objects.

The FlowPanel class being the class that runs runnable has an atomic Boolean variable. This is so that the change made to this variable is seen by all other threads. Atomic variable was used over just the volatile variable because multiple threads will be using the same FlowPanel class to modify and update data. This way, the swing library can synchronize and update to all other

Thread safety in the Flow class is also created by using join method on the threads created. The four threads each handle one quarter of the data for processing. This is achieved by modifying the FlowPanel class by adding the low and high parameters as stated above.

Findings

During the creation of this simulation, I have discovered that using four threads caused my simulation to become extremely slow. As a result, the submitted code has another zipped folder

called MultiThreads which is basically the same code but makes use of the four thread design stated above. For practical purposes, I have submitted code that works with only one thread because one thread seems to be fast enough to demonstrate simulation of water even though the simulation itself could have been faster.

Conclusion

Multi threaded programming is difficult and should be designed carefully. This careful design of multi threaded programming must avoid bad inter-leaving and data races for prevention of bad code outcomes and prevent deadlock which causes a program to stop working. Finally, concurrency should be implemented carefully to ensure thread safety while achieving faster speeds of the program.