CSC2002 Assignment 4

Modifications:

Land class

The appropriate dimension attributes were added. The accessors and mutators method were implemented as well as method to reset the shading. 3 methods are synchronized in this class to provide mutually exclusive access to the data.

The methods are shadow(), getShade() and setShade(). The shadow method uses the getShade() and setShade() methods in its definition, this is why they were synchronized. The shadow() method is also a "write" method, this is also why it was synchronized. The shadow() method will reduce the sun received by the occluded trees. The sunlight values are reduced to 10% of their original value. If a tree partially occupies a block, the value will still be reduced to 10%.

Tree class

The synchronized methods sunexposure() and sungrow() were implemented in this class. The sunexposure() method calculates the average sunlight obtained by the cells containing the trees. This average is done by first calculating the total sunlight by iterating throughout the tree regions and dividing by the number of cells.

The sungrow() method will increase the extent of the trees depending on the sun exposure it has gained.

Both methods were synchronized to prevent race conditions and allow mutually exclusive access to the data as both are "Write" methods.

TreeGrow class

A "reset", "play", "end" and "pause" button has been added to the GUI as well as a year counter.

The reset button will call the reset() method in the class which will set the extent of all the trees to the original value.

The pause button will pause the simulation.

The play button will play the simulation if the simulation has been previously reset or paused.

The end button will close the program.

The simulate() method will commence the simulation when ran. The GUI will set up and the simulation will begin. The method will run the simulation in parallel by creating an object of the simulateLayer class (more details to follow). A ForkJoinPool object is used as a common pool for the parallel operations. The simulation will start with larger trees [18,20) down to [2,0) to properly adjust the shading. The method will also print out the time taken to compute each generation and the number of years calculated.

The main method will run the simulate() method;

Created classes:

layerSimulation class

This class extends the recusiveAction class and uses a parallel divide and conquer approach to calculate the growth of every trees. This will be used to simulate the growth every year.

Concurrency:

A volatile Boolean value is used to check if the simulation is running in the treeGrow class. It acts as a flag for pausing the threads for the simulation.

The GUI and the simulation threads occur concurrently while sharing the same data because of the mutual exclusion access.

The synchronized methods mentioned above were implemented to prevent race conditions when accessing the shared data while running the program. When accessing the data, the synchronized methods prevents interleavings which will compromise the data and thus the program. When using a synchronized method, the thread has to finish its work on the data before another thread can access it. This is shown in the Land class when calculating shadow and in the Tree class when calculating the sun exposure.

The program was run successfully and simulated more than 300 years and no race condition occurred.

Model View Controller:

The model view controller wants the updates and the interface to be separate. This was done by running separate threads for the GUI and the tree growth calculations. The GUI represents the view and the calculations are the model.

The controller accepts input and converts it to commands for the view or model. In this case,the controller was the buttons found on the GUI. When pressed, the buttons will both affect the GUI and the running calculations.

Therefore the Model View Controller pattern is respected as, the uses sees only the GUI and can interact will the program using the Controller(buttons) which will affect the Model(running program calculations)