A C++ Simulation of Forest Fires Burning

Final project for CS–273 Data-Structures

Code and deliverables authored by Eli Erickson, Mitchell Schoenbacher, and Benjamin Still

**Table Of Contents**

* **Introduction**
* **High Level Technical Overview**
  + **Code Organization**
  + **UML**
* **Technical Details**
  + **Creating a Forest**
  + **Simulation Framework**
  + **Region Updating**
  + **Deployment of Fire Teams**
  + **File I/O**
* **Function Index**
* **Running a Simulation**
* **Warnings and Authors Notes**
* **Future Development**

**Introduction:**

**The goal of this project was to improve our team’s ability to implement associative and sequential data structures (namely a map and a quadruply linked list) as well as queues, build on already developed skills to properly handle complex allocation and deletion of dynamic memory (including creation and deletion of informative “log” files), and develop a virtual model of a simplified forest fire.**

**This simulation runs on a n x m grid of “regions” each of which represents 1 square mile, each region has some amount of density and wetness (factors that impact if a region will catch on fire) and wind, which is a constant value that speeds up how much a region will burn in an hour. Once the simulation is running it will iterate through as set number of timesteps each one representing one hour. Once certain burning thresholds are met on regions, firefighters will be deployed and will have a chance to extinguish that region. Otherwise, they will attempt to prevent future burning by fortifying regions.**

**A High-Level Technical Overview:**

***Code organization***

**At a birds eye view, this simulation is based on 4 key objects. A forest contains numerous regions , which in turn may contain fire stations , who hold multiple fireteams . Main() will construct a forest object and employ it’s run simulation function, and then handle outputting information from the simulation. The figures below displays this relationship and elaborates on these objects:**

A diagram of a fire station

AI-generated content may be incorrect.

***UML***

**The below figures are UML diagrams of each of the objects. These will be referenced in the next section:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Technical Details:**

***Creating a Forest:***

**Under the hood, a forest works somewhat like a container holding a quadruply linked list of pointers to regions, that is accessed via a map that assigns each region an unique ordered pair. There are 2 distinct steps that go into creating this forest. First the nodes are constructed and put into the map. Next the map is used as somewhat of an iterator, allowing us to assign north, south, east, and west pointers.**

**To create nodes 2 for loops are used, the first iterating through the x dimension the second through the y. Inside the second loop the node is added to the map at position (i, j), where i is the current iteration of the outer loop and j is the iteration of the inner loop. It can be visualized as “stacking” regions on one another up to the maximum height, and then moving to the next y position and doing it again.**

**To accurately assign pointers, an outer x\_dimension loop runs with 2 separate loops inside of it. The first one runs from y = 0 to y = max – 1 and assigns the north pointer. The second one runs from y = 1 up to y = max and assigns the south pointer. This way the node y = 0 does not get assigned a south pointer, and y = max a north. A similar approach is taken with the east west pointers, however rather than the outer loop controlling x it controls y.**

***Simulation Framework:***

**Once supplied with necessary arguments, the simulation performs it’s setup. First counter variables are created to keep track of simulation statistics, then random regions are struck by lightning and fire stations are randomly assigned. Then the simulation is run. It essentially consists of a for loop that runs on the condition the max number of timesteps haven’t been reached and the forest hasn’t burnt itself out. It will call the update function at each timestep for every region in the forest. Finally a report will be run and the log folder will be created.**

***Updating regions:***

**The update() function is the most important one in regions. It decides how the region will behave on that time step. First a station check is run, this will determine behavior if the region has a station. If it does have a station it will check the surrounding regions to decide if a fire team needs to be deployed or not. Next a team check will run. It will check to see if a team is currently deployed to this region. If a team is deployed and the region is burning, a random roll to determine if the firefighters extinguish is run. Otherwise, if the region is not burning but has a team deployed, they will fortify the region, increasing wetness and decreasing density, making the region less likely to burn. This check also will see how long the current fire team has been working, and remove it if they have worked more than 24 hours ( note this is currently a hard coded limit and many of the fire team data fields are currently unused ).**

**Once these checks have run, if the region is not yet on fire and has not been burnt out, the update function will check the regions around it to see if they are burning and roll to see if it will catch on fire. If the region is on fire, it will burn and amount determined by (random chance + wind) / 10. This function will return the status of if it is burning.**

***Deploying Fire Teams:***

**Fire teams each have a different requirement for being deployed. The first team will be deployed when burning left is less than 90%, then 70%, the 60%. Currently efficiency is not factored in, but in possible future builds, efficiency would be set as well to be proportional to where the fire team appears in the queue. This way as the fire burns more better fire teams get deployed to fight it. Currently, all teams fight with the same efficiency.**

***File I/O:***

**The exportLog function is the only function in the program that uses file I/O. A “log” folder is created in the project directory immediately after program execution begins, and log files are created within this folder at the end of each simulation timestep. Simple text output and the ASCII art forest map are all that are entered into the logs.**

**Function Index:**

***Forest Functions***

lightningStrike and assignStation are both sub functions of running simulation. These functions pick random regions to either set ablaze, or to assign a fire station to. These are only called at the start of simulation. They can be called as many times as the user inputs.

runSimulation is the function to actually run the simulation. This function takes in the maximum number of hours, the number of firestations, the number of lightning strikes, and a factor for the wind. Wind is used as a multiplier for burning regions. It makes fires more likely to spread, and burn regions faster.

checkpoint and printMap are both used during the simulation to show how the fire is developing. Checkpoint is called every 10 hours to print some basic information about how many regions are currently burning or are done burning. printMap prints the entire map. By looping through the forest it prints different characters to visually show how the forest is burning.

exportLog is used to save all of the information about the fire. This is called every hour, and creates files with all of the information about the fire.

***Region Functions***

Update is the main function of regions. If the region is already on fire it decreases how much burning left it has to do, and may put itself out. If the region is not on fire it calls update\_Check\_Burn. This function looks if the neighboring regions are on fire. If so, it attempts to light on fire and decrease the home regions wetness.

Fortify and extinguish are used by firefighters. Fortify decreases density and increases wetness. This makes the region less likely to burn both temporarily and permanently.

**Running a Simulation:**

Upon starting the program, the log folder will be created, and you will be prompted to enter multiple simulation parameters. First, enter the desired x and y dimensions (miles) for the forest you wish to simulate. You will then be prompted to enter the max timesteps the simulation may run for. If you wish to simulate the fire until it is fully extinguished/burnt out, enter a high number such as 1000. Next, enter the desired wind speed. 30-50mph is a recommended speed, as lower or higher speeds may result in a lack of spread or overwhelming burn speed. You will then be prompted for the desired number of lightning strikes and fire stations. 1 strike and (0.01\*regions) are recommended values.

After these parameters are entered, the simulation will run until the max timesteps are reached or zero regions are burning. Every 24 timesteps, a brief checkpoint will be printed to the terminal, allowing the user to vaguely see the progress of the fire. After the simulation is completed, you will be prompted to “enter any key to delete logs”. Do not do this yet!

Upon opening the log folder, you will see many (likely hundreds) of text files titled “step\_”. These text files contain information about the fire’s behavior during that timestep, as well as new events that occurred during the step. You will also see a printed ASCII art map, showing the forest and the fire spreading through it. The map key contains helpful information integral to understanding the map. Once you are satisfied with viewing the logs, enter any key to the terminal to delete them. You have now simulated a forest fire!

**Warnings and Authors Notes:**

Negative numbers cause a lot of problems. A negative time or negative strikes input won't do anything. Time should cause the program to run one step and then stop. Negative strikes would do the same thing because the simulation stops once there is no fire. A negative wind will cause the forest to grow instead of burn.

Beware that the simulation creates a lot of files. The log folder will be filled with new information every step. This is great for seeing lots of detail in how the fire burned. However, if the files are not deleted before the program is run again, they will remain there unless overwritten and can be falsely interpreted as timesteps from the current simulation. The program automatically deletes files at the end of the simulation, but they will remain there if the program is manually terminated early.

**Future Development:**

For the future, there are multiple improvements that could be made to the simulation. Factors such as wetness, wind, and density could all use thoughtful balancing in order improve the realism and spontaneity of the simulation. Some factors are very volatile, changing the outcome of the simulation greatly when minorly tweaked. Other factors could be more influential on the simulation.

Additionally, a less uniform, less linear approach to how fire teams are deployed would increase simulation accuracy and improve predictive value. However for the scope of this project, there was not time to implement this.

Another future improvement would be to improve the ease with which users may follow along with the simulation in real-time, as well as having more detailed logs. Currently, users are not able to see when fireteams are deployed or when they extinguish fires. This improvement would allow for a more informational and immersive experience.