Project 3 – Acequia Water Resource Simulator

Name: Izzy Adegbenro and Ash Vigil

Professor: Dr. Siamak T

Date: 5-19-25

1. What This Project Is About

This project simulates how water flows through an acequia system which is basically, a network of canals. Our job was to write the solveProblems() function inside StudentSolution.cpp, which controls how water gets distributed from one region to another. The idea is to make sure regions that need water get it without overfilling or wasting it.

2. How My Code Works

We wrote a few helper functions to find and control canals, and then added logic inside the main solveProblems() loop to open or close canals based on how much water each region has or needs.

Finding the Right Canal

```
6   // Find index of the first canal from a source region
7   int findCanal(const std::vector<Canal*>& canals, const std::string& sourceRegion) {
8       for (int i = 0; i < canals.size(); ++i) {
9           if (canals[i]->sourceRegion->name == sourceRegion) {
10              return i;
11          }
12      }
13      return -1; // Not found
14  }
```

This just searches the list of canals and returns the index of the first one that starts from a specific region. It's used later to open or close that canal.

Opening or Closing Canals Manually

```
17  void release(const std::vector<Canal*>& canals, const std::string& region) {
18      int index = findCanal(canals, region);
19      if (index != -1) {
20          canals[index]->toggleOpen(true);
21          canals[index]->setFlowRate(1.0);
22      }
23  }
24
25  // Close all canals from a region to stop water flow
26  void close(const std::vector<Canal*>& canals, const std::string& region) {
27      int index = findCanal(canals, region);
28      if (index != -1) {
29          canals[index]->toggleOpen(false);
30      }
31  }
32
```

These are basic helper functions we made to either fully open or close a canal. They're not strictly necessary for the main simulation loop, but they made testing easier and could be useful for other logic.

Main Simulation Logic

```
// Main solution logic, at least the best i could come up with
void solveProblems(AcequiaManager& manager) {
    auto canals = manager.getCanals();
    auto regions = manager.getRegions();

    while (!manager.isSolved && manager.hour != manager.SimulationMax) {
        for (auto canal : canals) {
            Region* src = canal->sourceRegion;
            Region* dst = canal->destinationRegion;

            // Determine if the source has extra water
            bool sourceHasExtra = src->waterLevel > src->waterNeed;

            // Determine if destination is in drought or still below its need
            bool destNeedsMore = dst->waterLevel < dst->waterNeed;

            // Extra condition: avoid sending water if destination is already near capacity
            bool destNotNearCapacity = dst->waterLevel < (dst->waterCapacity - 10);

            if (sourceHasExtra && destNeedsMore && destNotNearCapacity) {
                canal->toggleOpen(true);

                // Dynamically adjust flow rate based on how much the destination needs
                double deficit = dst->waterNeed - dst->waterLevel;
                double flowRate = std::min(1.0, deficit / 20.0); // Cap flow rate
                canal->setFlowRate(flowRate);
            } else {
                canal->toggleOpen(false);
            }
        }

        manager.nexthour();
    }
}
```

The idea here is simple: if a source has more water than it needs, and the destination needs water (but isn't almost full), then I open the canal. The flow rate is based on how much the destination is missing so if it only needs a little, it gets a small flow. If it needs a lot, it gets more (up to a cap of 1.0). If any of those conditions aren't met, the canal stays closed.

3. Screenshots of the Program Running

```
fmafj@Ashs-Asus MSYS /c/Users/fmafj/github-classroom/UNMECE/final-project-ash-and-izzy
$ ./a.exe
Current State of the Regions:
-----------------------------
Region: North, Water Level: 54, Water Need: 57, Water Capacity: 153
Region: South, Water Level: 95, Water Need: 70, Water Capacity: 164
Region: East, Water Level: 42, Water Need: 67, Water Capacity: 189
------------------------------------------------------------
Please write your solution in the StudentSolution.cpp file.
Your code must solve each region's water needs within the following simulation time: 109
When you have saved your code and ready to run the simulation, you may press Y to run.
Press Y to test your solveProblems function.
y
Current State:
-----------------
Region: North, Water Level: 54, Water Need: 57, Flooded: No, Drought: No
Region: South, Water Level: 70, Water Need: 70, Flooded: No, Drought: No
Region: East, Water Level: 67, Water Need: 67, Flooded: No, Drought: No
------------------
Not all regions were solved in time.
--------------------

----------------
Leaderboard:
StudentSolution:10

fmafj@Ashs-Asus MSYS /c/Users/fmafj/github-classroom/UNMECE/final-project-ash-and-izzy
$ ./ready.exe
Current State:
-----------------
Region: North, Water Level: 54, Water Need: 57, Flooded: No, Drought: No
Region: South, Water Level: 70, Water Need: 70, Flooded: No, Drought: No
Region: East, Water Level: 67, Water Need: 67, Flooded: No, Drought: No
------------------
Not all regions were solved in time.
--------------------

----------------
Leaderboard:
StudentSolution:10
```

These screenshots show that the simulation is running properly. Water is flowing from sources to destinations, and canals are opening or closing based on need. The results update each hour until everything is balanced or time runs out.

4. Summary of What we Did

Basically, we:
- Looked at each canal one by one,
- Checked if the source had more water than it needed,
- Checked if the destination still needed water (and wasn't too full),
- Opened the canal and set a flow rate based on how much was needed.

This lets the system balance itself out over time, and we use manager.nexthour() to move to the next hour and recheck everything again.

Test Results Summary

The solution has been tested across multiple simulation runs, each representing a different configuration of initial water levels, needs, and capacities for the North, South, and East regions.

In the first test, South and East regions met their water needs, while North fell short, resulting in a score of 18. In the second test, North again failed to reach the required water level, even though South and East were successful. This led to a score of 10.

In the third test, both South and East were in drought and failed to meet their needs, resulting in a score of 0. The fourth test demonstrated significant improvement, where all regions nearly reached their exact water needs but fell short by a margin of less than 0.001, likely due to floating-point precision errors. Despite the near success, the score remained at 10 or less.

Performance Analysis

The solution performs okay in dynamically adjusting water flow and responding to regional needs. It shows a strong grasp of how to coordinate between multiple regions, accounting for both excess and deficient water levels. However, a recurring limitation is the failure to completely satisfy all regions within the given simulation time.

A key issue appears to be floating-point precision. In the final test, water levels such as 52.0004 and 68.9999 were observed, which are extremely close to the needed values but still result in simulation failure. This suggests that slight adjustments to flow rates, such as incrementing them by a small buffer (e.g., 0.01 or 0.5), may improve success rates and help fully meet the simulation objectives.

5. Final Thoughts

This project helped us get better at managing multiple conditions in a loop and thinking about how to simulate real-world systems like irrigation. It was fun figuring out how to send the right amount of water without flooding or starving any of the regions.