Acequia Manager Guide

This is a guideline for the Acequia Water Resource Simulator and the existing structures and functions that allow the simulation to carry forward and the student learning objectives and contributions to the simulator.

About Acequia

Acequias are a network of canals historically used to move water from rivers and other water sources to communities, primarily for the purposes of irrigation. Within New Mexico, acequias have a community infrastructure

This project is largely inspired by the acequias of New Mexico as a metaphor for water resource management to meet the needs of state. Canals in this simulation serve in transferring water from existing water sources to the communities across each region.

About Game:

There exists several regions that describe the current water level of that region and the current water needs of the community. Each region is connected to several water sources that supply and transfer water to other regions.

The user will be provided current conditions to that region and must provide code to the solveProblem() function that will run until the solution satisfies the needs of each region OR until the max time has been reached.

Once the solveProblem() function is complete, the Acequia Manager assesses the solution of the student based on the final state of the region and any penalties experienced during the simulation. A score is determined is based on:

1. 10 points for a region whose final state is not flooded (bool isFlooded = false) and not in drought (isInDrought = false) and a region's water level is above its water need.
2. -1 point for each time the water level exceeds the water capacity of the region, or the water level of the region has reached 0.
3. 100 points If all regions are solved correctly within the allotted simulation time.

The Structure of the Simulator

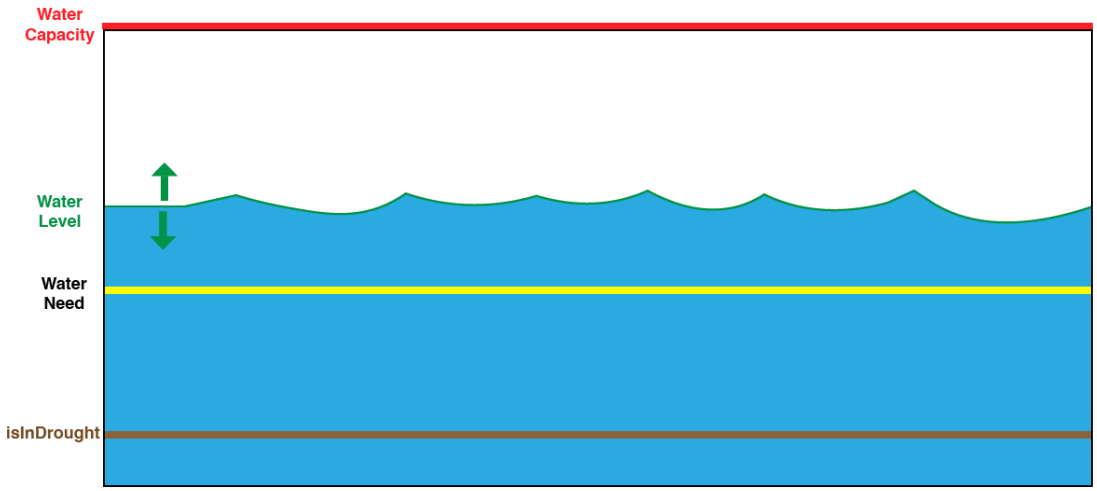
The simulator contains 3 primary classes that interact with each other to represent the interaction of water resources across the State of New Mexico. The student will develop the logic that transfers water between the different regions and the water Sources supplied to that region.

How the simulator works:

1. The top file, **TopMain.cpp**, generates random values at execution time and stores those values in the **RandomValue.dat.**
2. While running the TopMain.cpp file, the current state of each region is displayed to the student.
    a. This will share a summary of the region object to assist with the student's solution
    b. This will also share a time constraint that the student must comply with. This time constraint is the number of simulation runs (in units of hours) and is to guide the student's strategy for resolving each region.
        i. This is NOT real time units. Rather, it represent the units of time within the simulation. For this demonstration, hours.
3. Once the student has been provided the initial constraints, the student may work on the solveProblems() function by:
    a. Working on the Canal objects by:
        i. Opening canals
        ii. Setting a flow rate
        iii. Determining how long (time) each canal may be open
4. When the student has saved their solveProblems() function in **StudentSolution.cpp**, they may press Y (or y) to begin the simulation run with their solution.
5. Once the run has completed, the evaluateSolution() is used to determine the score for the student based on the final state of each region. For each region that meets the two boolean flags as false and the waterLevel is at least above the waterNeed, a student is granted one point. The total sum for each successful region will be the student's final score.
6. The leaderboard displays the scores for students in the order of the highest score.

Classes

Region:
This class represents the primary status of water resource information for a designated area (North, South, West, etc..) . As such this class describes the current state of each region relating the current amount of water sources to the water needs of the region. It also describes the current conditions of the region such as whether the community is in drought or is currently flooded.



The description of the water resources in this class are:

- Name of the region
- The current waterLevel of that region
- The waterNeeds of that community
- The waterCapacity of the community
- A vector of water Sources that are allocated to that specific region: std::vector <waterSource* > waterSupplied
    - These collection of waterSources contribute to the waterLevel of the region
        - Some may have
- A flag indicating drought conditions: isInDrought
    - Flag can be dependent on the current state of the waterLevel with respect to a prescribed waterCapacity
        - If waterLevel is 20% of waterCapacity, region is in drought
    - This flag is used in the evaluation of the student solution and therefore, its state must change with respect to the new state of the region after the student solution has been run
- A flag indicating Flooding conditions: isFlooded

- This flag is used to evaluate if the current water level has reached or exceed capacity. This is used in the evaluation of the student solution.

Counters:

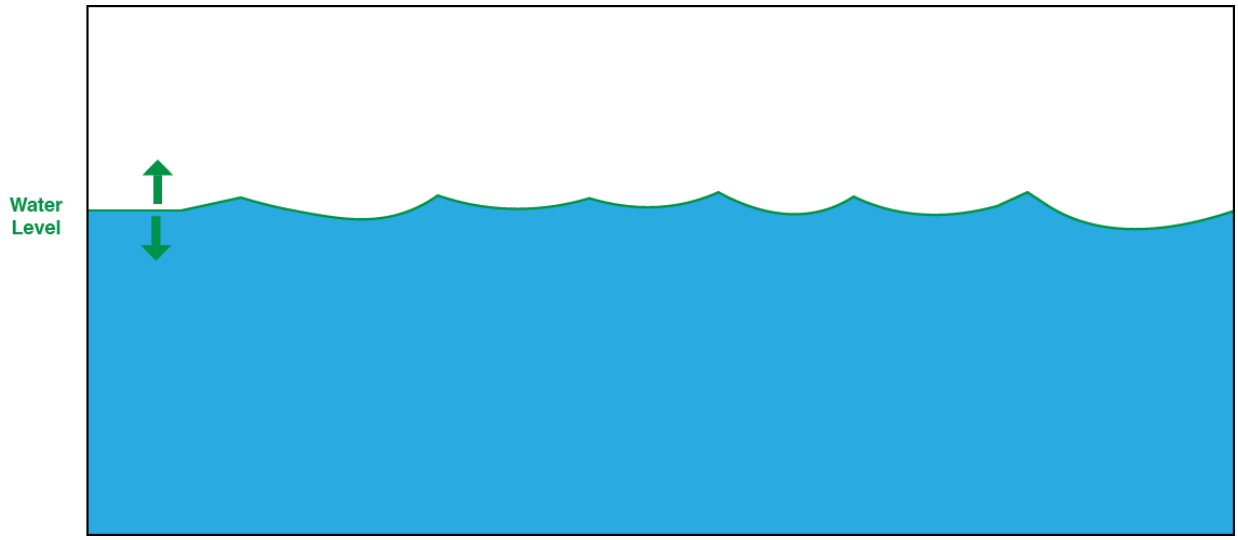
- Overflow
  - This counter is incremented each time the waterLevel exceeds the waterCapacity throughout the simulation run
  - This will be used in determining penalties for the final score
- Drought
  - This counter is incremented each time the waterLevel reaches a level of zero
  - This will be used in determining penalities for the final score

Functions:

- updateWaterLevel(): this function updates both the current water Level as well as updates the boolean flags and counters if conditions are met
- AddWaterSource(): this function adds a waterSource to region at the start of the initialization of the simulation

WaterSource

This class defines a particular water source that is allocated to one or several regions depending on geographic or weather related parameters. There are several defined types of water sources with each type having restrictions to the capabilities of water movement that are set within the class.



Members of this are:

- The name of the water Source
- The waterLevel of that water source (in _____ units)
- The type of water source. The water level type contains several restrictions to how water can move and how that water source can be affected by natural weather such as snow, rain, etc.)

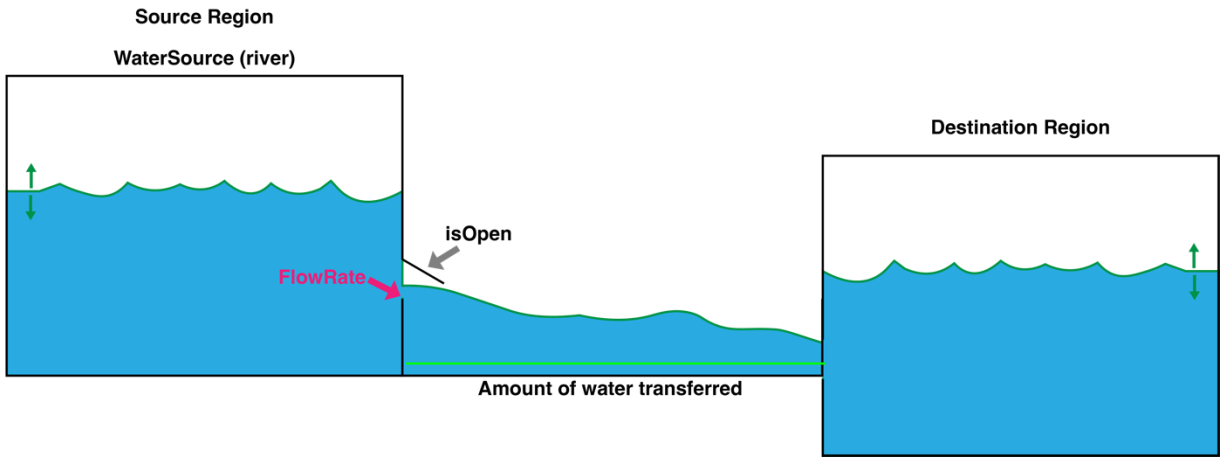
- River
- Dams
- Underground

Functions:

- UpdateWaterLevel(): This function updates the water level of the waterSource. If the waterSource reaches critical levels that water cannot flow, the canal cannot be used to transfer water.

Canal:

This class serves as the class that determines the movement of water sources to regions. Canal serves the purpose of brokering transactions between one or more region and the water sources they are supplied to those regions.



The members of the canal class are:

- Name of Canal
- SourceRegion: This will be the region that is allocating water from one region to the next
- DestinationRegion: This is the region that receives the water from the source region
- WaterSource: This is the waterSource that is used by the canal to transfer water between regions.
- FlowRate: This member represents the rate at which water is traveling through the canal. For now, the rate is between 0 and 1 gal per second
- IsOpen: This member indicates whether the canal is active and transferring water

Functions:

- SetFlowRate(); This allows the user to set the flowRate for a given canal. This member can be changed numerous times throughout the simulation run
- ToggelOpen(): This changes the state of the canal between active and inactive. This also can be changed numerous times throughout the simulator.
- UpdateWater(): This calculates and updates the amount of water transferred between the source and destination regions.

Simulator Structure
The simulator manager serves the purpose to implementing initial parameters of the several classes and connects these structures to inform the water resources and across the state and how those resources can be used to equally serve the needs of the communities across each region.

AcequiaManager Class:
This class holds the vector of several classes as well as the initialization of parameters function, a function to display the state of each region, a function for showing a leaderboard, and getters for the student to access.
Members:

- Vector of Regions: the container that stores the address of each region object
- Vector of WaterSources: the container that stores the address of each WaterSource object
- Vector of Canals: the container that stores the address of each Canal objects
- Map for the leaderboard
- Hour: a variable of the type integer that represents time in units of hours during the simulation run
- SimulationMax: a variable that represents the max length of time for the simulation to run. This variable is represented in units of hours
- SolvedTime: a variable that is used to save and report the amount of time needed to solve the problem of the simulation
- IsSolved: a boolean variable that is used to determine if the simulator has been solved. This will be primarily used in the to stop the simulation prior to reaching the maximus simulation time allotted

Functions:

InitializeRandomParameters(): this initializes the random parameters of each class (region, waterSource, and canals) from a collection of helper functions. These helper function aid in organizing the types of classes being initialized for readability. The list of helper functions are as follows

- InitializeRegions(): The initialize the members of each region and then add that region to the vector containing all regions
    - Each region numerical values of waterLevel, waterNeed, and waterCapacity are randomly generated given the constraint that waterCapacity must always be equal to or higher than either waterlevel or waterNeed.
- InitializeWaterSources(): This initializes the members of each waterSource
- InitializeCanals(): This initializes the members of each Canal
- InitializeConstraints(): This initializes the constraints that work on the region class. In the current case, the boolean flags of each region are changed according to state of the waterLevel, waterNeed, and waterCapacity randomly generated.
- InitializeTime(): This initializes hour to 0 to signify the start of the simulation. The SimulationMax is also randomly generated between 50 and 120 hours(in simulation

units). The isSolved boolean is set to false indicating that the simulation has not been solved.

Constructor/Destructors: Thes are used to construct this class as well as memory clean-up of each class object when the program finishes.

Getters: These are functions that students can access the class objects in this simulation:

- GetRegion(): getter for accessing the vector of region objects
- GetWaterSource(): getter for accessing the vector of WaterSource objects
- GetCanal(): getter for accessing the vector of Canal objects

Display Functions: these functions are intended to show the student the current state of each region and also to show the final score of the student after their solution has been evaluated.

Time Function (nextHour()): This function moves the simulator forward and assesses the number of changes made and applies those changes on each region object.

This function is encapsulated within the AcequiaManager class to ensure to progression of the simulation run by incrementing hour (our time variable), so that students can focus on the task of solving when a given canal should be opened and what the flowRate should be set at for that given hour. The water level units are calibrated so that changes are readable to the user while ensuring the change in one hour is not extremely consequential to penalize the user for going over or under.

Within the function, all water transfers for that hours are updated across each region if a given canal's boolean has been set to true. The manager also calls on the solved function to determine if the simulation has been solved and if so, ends the simulation run.

Solved() Helper Function: This function is meant to evaluate the state of each region and determine if the simulation has been solved. This is accomplished by setting up a test boolean to true.

When looping across each region, if any region does not satisfy any of the criteria to warrant a solved region, the test boolean is changed to false. The value of the test boolean is returned to the isSolved boolean and checked by the nextHour function for determining if the simulation has been solved.

Penalties() Function: This function sums all region overflow and drought counters and returns the sum to the evaluateSolution function for consideration in the user's final score.
EvaluateSolution() Function: This function determines the final score of the user after the solvedProblems function has been completed.

This function evaluates each region to check if the region's final state satisfies the criteria that no boolean flag is true and the waterLevel is above the waterNeed. If so, the score is incremented by 10 points.

This function also subtracts the score by the number of penalties experienced during the simulation run.

If the simulation has been solved completely, an additional 50 points is added to the final while also showing the time the simulation was solved by.

Finally, the score and student name are reported on the leaderboard map.

Non-class function
SolvedProblems() Function: This function is the student-driven function where the student must control the flowRates and activate canals to solve the water levels of each region. The student will write the logic for each action that is taken per hour.

The solveProblems function breaks when the isSolved boolean is equal to true or the hours variable has reached to the simulationMax. Once this fucntion breaks, the manager then evaluates the final state of the region to determine the student score.

Simulation Run

Student Objective:

The student must upload each .cpp file (TopMain.cpp, AcequiaManager.cpp, StudentSolution.cpp and SimulatorMain.cpp). Please have all files in one folder.

Compiling and Executing the Program

The file, TopMain.cpp, will be the only file that needs to be compiled by the student. The student may compile simply by the following in the command line:

G++ TopMain.cpp

The executable code can be run using the regular calls to execute a file. For example:
./a.out

Top Main Executable:

When running the TopMain, the file will generate random values and report those values to the student user. The simulation time available to the student will also be shared (in simulation hours).

```
Current State of the Regions:
------------------------------
Region: North, Water Level: 75, Water Need: 79, Water Capacity: 101
Region: South, Water Level: 4, Water Need: 68, Water Capacity: 156
Region: East, Water Level: 42, Water Need: 95, Water Capacity: 150
------------------------------------------------------------
Please write your solution in the StudentSolution.cpp file.
Your code must solve each region's water needs within the following simulation time: 74
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.
```

After the values have been reported, the student should have enough details about the regions to write their solution in StudentSolutions.cpp file. Details about the canals and the paths they travel can be found on page ().

Working on the Solution in the StudentSolutions.cpp file

Prior to pressing **Y (or y)** in the TopMain program, the student will need to add code to the StudentSolutions.cpp file to work on the vector of canals to activate and set flowRates for a given hour.

```
void solveProblems(AcequiaManager& manager)
{
    auto canals = manager.getCanals();
    while(!manager.isSolved && manager.hour!=manager.SimulationMax)
    {
    //Students will implement this function to solve the probelms
    //Example: Adjust canal flow rates and directions
        if(manager.hour==0)
        {
            canals[0]->setFlowRate(1);
            canals[0]->toggleOpen(true);
        }
        else if(manager.hour==1)
        {
            canals[1]->setFlowRate(0.5);
            canals[1]->toggleOpen(true);
        }
        else if(manager.hour==82)
        {
            canals[0]->toggleOpen(false);
            canals[1]->toggleOpen(false);
        }
    //student may add any necessary functions or check on the progress of each
    //The manager takes care of updating the waterLevels of each region and wa
    //to solve how to address the state of each region


        manager.nexthour();
    }
}
```

The student may also use the get functions for both the Region, WaterSource for informative purposes such as identifying what waterSource carry water for the regions or what canals are connected to the regions. These details can also be found in the AcequiaManager.cpp file.

After the StudentSolution.cpp File is complete and saved, the student may press Y (or y) in the TopMain.cpp program. This will compile and execute the StudentSolution.cpp, AcequiaManager.cpp, and SimulatorMain.cpp.

```
Press Y to test your solveProblems function.
y
Size of Region :3
Current State:
-----------------
Region: North, Water Level: 75, Water Need: 79, Flooded: No, Drought: No
Region: South, Water Level: 4, Water Need: 68, Flooded: No, Drought: Yes
Region: East, Water Level: 42, Water Need: 95, Flooded: No, Drought: No
------------------
Current State:
-----------------
Region: North, Water Level: 0, Water Need: 79, Flooded: No, Drought: Yes
Region: South, Water Level: 139, Water Need: 68, Flooded: No, Drought: No
Region: East, Water Level: 150, Water Need: 95, Flooded: Yes, Drought: No
------------------
Not all regions were solved in time.
--------------------

----------------
Leaderboard:
StudentSolution:-89
```

At this time, the simulator will run according to the instructions provided in the StudentSolution.cpp file. After the simulation has completed, each region is evaluated and a score is generated.

Canals Information (Also can be found in AcequiaManager.cpp):

Canal Name:	Source Region:	Destination Region:
A	North	South
B	South	East
C	North	East
D	East	North