PCOO - Final Assignment Report
Water Distribution

Armando Sousa 76498
ammsousa@ua.pt

January 25, 2020-12

Contents

Chapter 1

Project description

A small dam is the sole provider of water to a town. The water is first sent from the dam to the various deposits located throughout the town. These deposits are then the ones in charge of providing water to every house in the town.

Whenever a house tries to fetch water and the deposit is empty, an alert is sent to the dam's console. This message is read by a worker that then turns the appropriate valve to refill the empty deposit with water from the dam. When the deposit is filled the house can start retrieving water. The houses are able to retrieve water from any deposit in town.

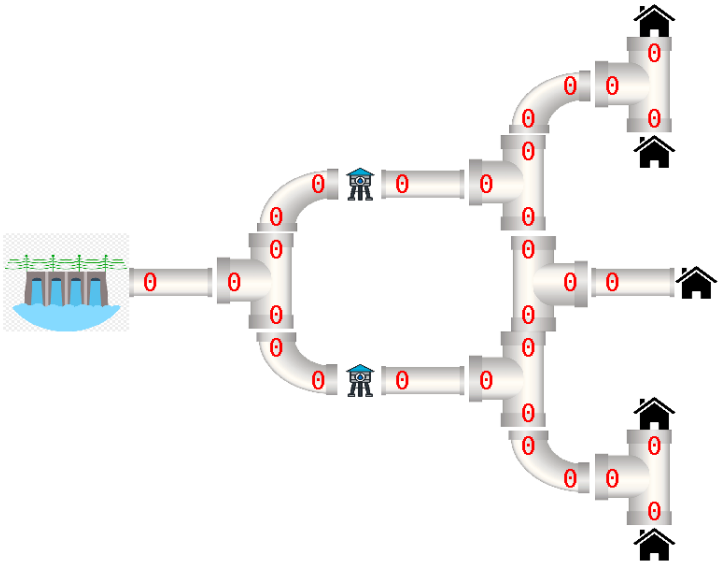

Figure 1.1: Example of Town Map

Chapter 2

Implementation

The first thing the program does is create a Map object. The map class is where everything in the board is drawn initially and where the methods used to simulate the movement of water are located. This object is the only of its type created and server as a parameter for every thread instantiated. It contains information about the locations of the different houses, deposits, pipes and the dam.

After creating the map, the program has the positions of every deposit, console and house in the map. With this information the different shared objects, that include the AlertConsole and Deposits, as well as the Workers and Houses' threads can be created and started.

```
SharedAlertConsole con = new SharedAlertConsole(new AlertConsole(map.consolesPositions[0], map));

int i = 0;
SharedDeposit[] dep = new SharedDeposit[map.depositsPositions.length];
for (Position position : map.depositsPositions) {
    dep[i] = new SharedDeposit(new Deposit(Configuration.DEPOSIT_MAX_WATER_CAPACITY, i, position, map));
    i++;
}

for (int j = 0; j < Configuration.NB_WORKERS; j++) {
    new CThread(new Worker(j,dep, con)).start();
}

CThread[] t = new CThread[map.housesPositions.length];

i = 0;
for (Position position : map.housesPositions) {
    t[i] = new CThread(new House(dep, con, position, Configuration.HOUSE_WATER_CONSUMPTION_RATE,
                                                    Configuration.HOUSE_MAX_WATER_CONSUMPTION));
    t[i].start();
    i++;
}
```

Figure 2.1: Threads and shared objects creation.

private

2.1 Threads

2.1.1 House

The house thread extends the CThread class and has simple life cycle. Until the house consumes the maximum volume of water it needs, a value that is defined when the object is initialised, it will request water from one of the existing deposits at random.

If the deposit it chooses is empty or doesn't have enough water to accommodate its needs, the thread will send an alert to the AlertConsole object saying that the deposit is empty. Afterwards it will wait until it receives a broadcast and the deposit is full once again. Only then will it start fetching the water from the deposit. If the water container isn't empty when the house access it, the thread will start collecting water immediately without sending any kind of alert.

Once the total amount of water consumed by the house is equal or greater than its max water consumption the thread will stop requesting water and terminate. The speed at which this happens depends on how the house's water consumption rate defined in the variable of the same name.

2.1.2 Worker

A worker thread's life cycle depends heavily on whether or not there are any alerts in the AlertConsole.

Workers start by waiting for a new alert to arrive at the AlertConsole's queue. Once an alert arrives, a worker thread is woken up, removes the alert from the queue and then starts the process of refilling the deposit specified by in the alert received.

Once the deposit is back to full it sends a broadcast to all waiting House threads. In the end it repeats the process once again if the queue still isn't empty or waits until a new alert arrives at the console.

2.2 Shared Objects

There are 2 types of shared objects implemented: The deposits that hold the water and the alert console that keeps track of the alerts sent. The synchronisation in these objects is kept using the Mutex and MutexCV classes from the pt.ua.concurrent library.

2.2.1 Deposit

The class where the water is kept. Workers fill them with water when they are empty the houses retrieve water from them, until they're either empty or don't have enough water to fulfil their requests.

A deposit always starts with its full capacity of water. The house threads call the useWater method which subtracts it water volume needed from the current water level. When the deposit empties to 0 a worker will be woken up and restore its water level to its maximum capacity.

While the container is being refilled the house threads wait using a MutexCV object. The worker threads on the other hand are never kept waiting inside the SharedDeposit object and only make use of the refill method.

2.2.2 AlertConsole

The AlertConsole java class implements the methods that allow the workers to know when and what deposits need to refilled. The alerts are kept in an Integer queue where the queue's elements are the ids of the the deposits that are empty.

The addAlert method are called by the House threads whenever a deposit becomes empty. The id of the empty deposit is added to the queue and a broadcast is sent to all Worker threads currently waiting.

These worker threads are waiting inside the readConsole method. When they become active again, they remove the first element in the queue and return its value.

2.3 Map

Besides the methods described before, both the SharedAlertConsole and the SharedDeposit class also possess 2 extra methods that are identical . The startReplenishing and stopRepleneshing methods are responsible for calling the method inside the map object that update the board when water starts or stops running.

This method called updateMap has as its parameters the initial position from where the water departs, that can either be the console or deposits' position, the destination position, that can be a deposits or houses' position, and the volume of water.

To simplify the movement of the water through the pipes, the algorithm implemented is very simple. Water always moves from left to right. Anytime it reaches a T junction and has to decide which way to go it will decide based on the y position of its destination compared to its own position. If the destination's y position is lower than its own y position the water will move up, else it will move down.

Every time the updateMap method is called, it is first verified which type of pipe the water is currently own since different pipes have different shapes. Since some of these pipes have more than one entrance it is also needed to take into account the water's direction in a pipe so that when eventually the water

is drawn or erased from the board it is in the correct positions. Water is drawn when the newWaterVol parameter in the function is greater than 0 and is erased when it is lesser than 0.

Throughout the map there are several red numbers that change as the program runs. They represent how much litters of water are flowing through a certain pipe and to help see when a pipe is being used to to supply more than one place at the same.

This method is recursive and only stops when the current position equals the final destination position.

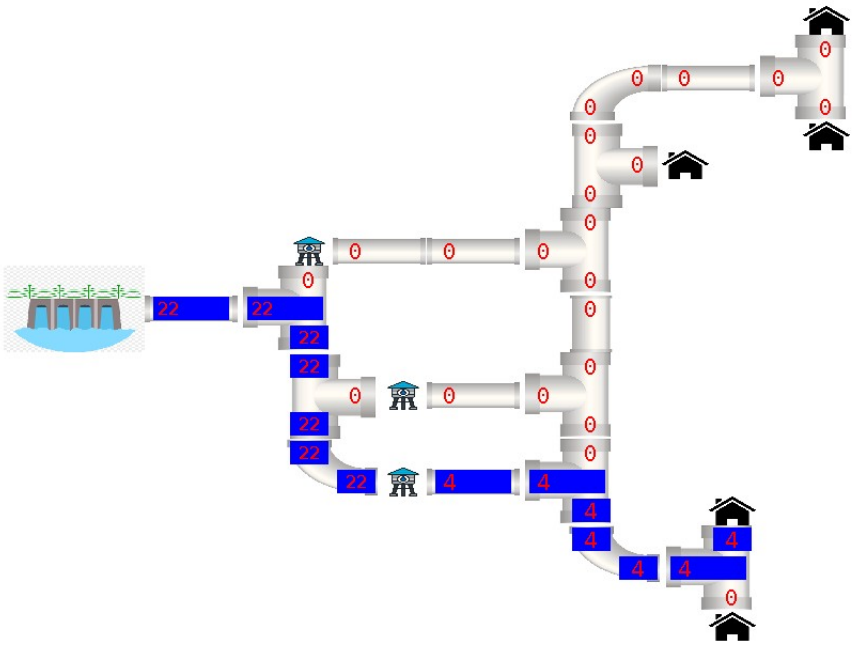

Figure 2.2: Example of deposit being refilled and then supplying water to a house.

Chapter 3

Problems

3.1 Graphical errors

In some cases when the water travelling through the pipes and the red numbers will seem to randomly disappear without any reason. This is caused by a bug in the gboard library and can be solved by minimising and opening the gboard window again.

Figure 3.1: Graphical error present. Gelem with value 0 doesn't appear.

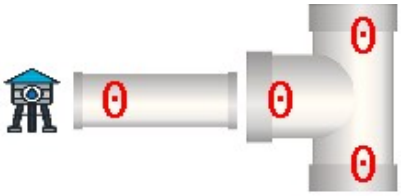

Figure 3.2: Graphical error no longer present. Gelem with value 0 is visible.

When constructing a new map for the program, it is important to make sure that all houses have at least one path connecting them to all of the deposits in town. If not there is the possibility of the house choosing to retrieve water from a deposit impossible to reach.

Chapter 4

User Manual

4.1 Requirements

The project makes use of both the pt.ua.concurrent library and the pt.ua.gboard libraries, develop by professor Miguel Oliveira Tomás of the university of Aveiro, and as such both are required to run the program successfully. They should be placed in the root folder of the project.

4.2 Running

There are 3 scripts inside the root folder for compiling, running and cleaning the program.

The run script can accept 2 arguments. The first argument specifies which map to use when running the process and the second one the speed of at which the program runs. If no arguments are given then the program runs using the default map, inside the DefaultMap.txt file, at a set speed to run the program. If only the first argument is given then the program runs at the default speed but with the map inside the file given as an argument.

Three different map files are included.

Example using the map in AltMap2.txt:

```
$ ./compile
$ ./run AltMap2.txt 280
```