**Write-up for WorldPopulationSimulator – The Backend**

In this write-up, I will attempt to describe how we got the WorldPopulationSimulator to work, what design decisions we made early on, and what areas could be optimized, but for the sake of time, were left merely functional.

To start off, I would like to describe what the WorldPopulationSimulator aims to accomplish. Based on the continental population growth rates (birth rate – death rate), the Simulator models the population change per day per continent, and when taken the sum of continents, the global population change. The net change is a rate that is unique to each continent, as determined by extensive online research. Because continental and global population fluctuations aren’t all that interesting in themselves, the architecture team working on the algorithms of calculations decided to bring in another factor of continental population change: natural disasters. The continental population net change does not account for disasters, which are random and have various levels of severity (in terms of fatalities and number of occurrences). The WorldPopulationSimulator research team spent many weeks, nay, months scouring national disaster databases to bring as accurate estimates on disaster occurrences and fatalities per occurrence as possible. Using the rates acquired by diligent research, our architecture team designed the data retrieval and main drivers of our applications.

I will now describe the design. As Bill Gates said, “It all starts with a text file.”1 A text file that is installed at initialization contains the rates of occurrence per continent and deaths per disaster. The storing of data in a text file has benefits, such as easy modification at a later time without recompilation. The data file will become important, as you will later see.

The top-most class is called Simulator.h, which defines Simulator objects. The Simulator automatically defines a Globe object (without your consent) to contain the Continents and their rates. The Globe, surprisingly enough, contains a vector of Continent objects. The Continent objects contain a vector of possible disasters that can happen on that Continent, stored as a Disaster type. The Continent class also has a Disaster queue, which holds the disasters that are to happen for that simulation time interval (day) until all the disaster calculations finish, and then modify the continental population as each Disaster gets popped off the queue.

At program initialization, a Simulator object is created. As I mentioned above, the Simulator object automatically creates a Globe object (automatic meaning the Globe initializer is in the Simulator constructor). The Globe constructor does some interesting things, which I will now describe. The Globe initializes the Continent vector with Continent objects. At Continent creation time, a struct is created (called `values`) that holds values (rates) for that continent, as stored in the data file. The struct is populated by a function in the Utility library, which finds the line number in the file where the specific data is stored, converts the value from a string to a double, and stores the value in the correct variable in the values struct. After the Continent objects are created, instead of them holding individual member variables, the object contains a `values` struct member variable, which holds the individual variables. At this point, a Simulator, Globe and 7 Continent objects are created (this is an instantaneous process). As a Continent object is being created, it contains a function in the constructor to initialize the Disaster vector in each Continent with the set types of disasters. For the sake of this simulation, each Continent contains the same 7 disasters. The difference is that the continental population of certain continents is affected less in some continents than others, because of no occurrences of that particular disaster, or because of the sparse human population (i.e. even though Antarctica may have earthquakes and volcano eruptions, the population of Antarctica is low enough that it isn’t affected). The interesting part happens each day the simulation runs, a value whose magnitude is indicated at program initialization.

On each update, the Globe object will walk through each Continent update function. The update function looks at the rates of occurrences of disasters and calculates whether a disaster will occur that day, using a combination of blockchain, encryption, TPMS, anti-lock braking systems and other advanced techniques. If a disaster is set to occur on that specific day, the disaster is added to the queue, and the amount of fatalities is calculated. The fatalities then reflect on the continental population, and incidentally, the global population. The advanced Graphical-User-Interface displays the occurrences of disasters in the continents. The shifting of continental populations is also reflected in the color of the populations: as the populations grows, the continent becomes redder.

There are many rooms for improvement in this application. Being students, Kostia and I dedicated a lot of time to get the simulation running, but we couldn’t implement every feature we wanted to in the short amount of time we had. Some things I would improve would be the initial reading of the rates from the values text file to the values struct, which lives in Continent objects. Basically, the text file is opened for each line that is read, which is about 19 \* 7, or roughly 135 times, which is grossly inefficient. A better way would be to open the text file, read all the values per continent, then close the text file connection. That would reduce the opening and closing of the text file down to only 7 times. What would be even more elegant would be to use a database to store initial rates, as well as collect data points as each day passes, like net population growth and percent of the original population that is still alive after n number of days. I would port this code to C# because of the much better graphics support and easy MSSQL Server integration (Python would work just as well). The Qt graphics is pretty robust, but picky in terms of compiler and other factors.

1 He didn’t actually say that…