Kostiantyn Makrasnov
Yuriy Fedas
CS273 – Spring 2018
May 17, 2018

Final Summary for the World Population Simulator

"A world without limits!"

General Project Summary

Our simulation specifically simulated the population of specific continents based on the researched net growth and disaster occurrence and mortality rates. The simulation will simulate the location of different disasters as well as their probable death tolls. All of the calculations were shown in real time at the rate that the user prefers. Such approach provides a global perspective on what type and how much of disasters occur within each continent as well as their impact on the population growth. The graph of the simulation will primarily be the view of the simulation occurring as well as the population rates available by the end of the simulation. Going into more detail, please observe the images below.

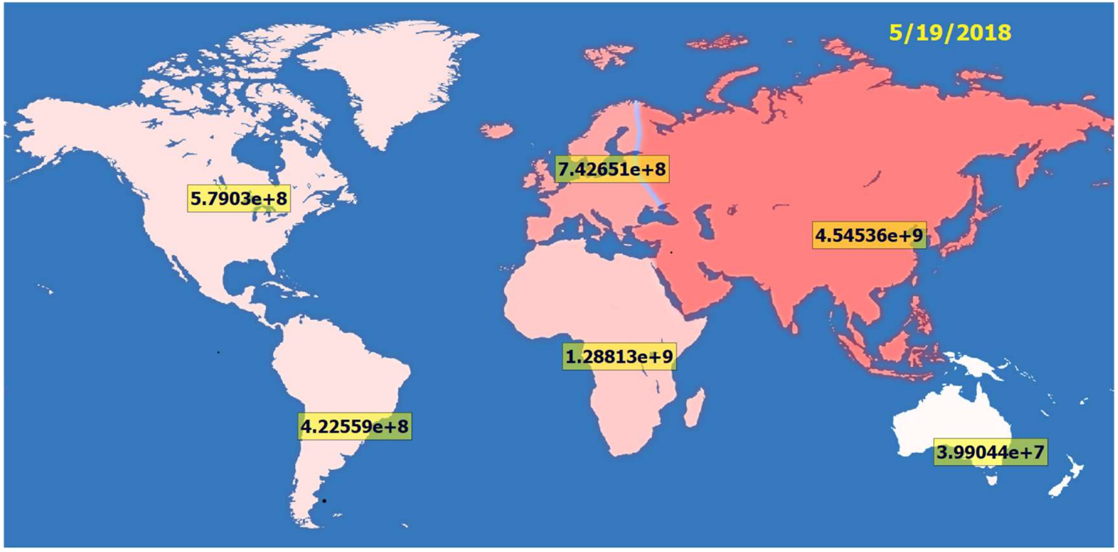

Figure 1. Initial View of the Simulation Window

Within the initial simulation view we can see the population amounts for each continent as wells as the population density as indicated by the number of red overlays on specific continents. However, this information can be found without the use of simulation, the actual simulated data and conclusions can be seen if we skip 5 months days ahead…

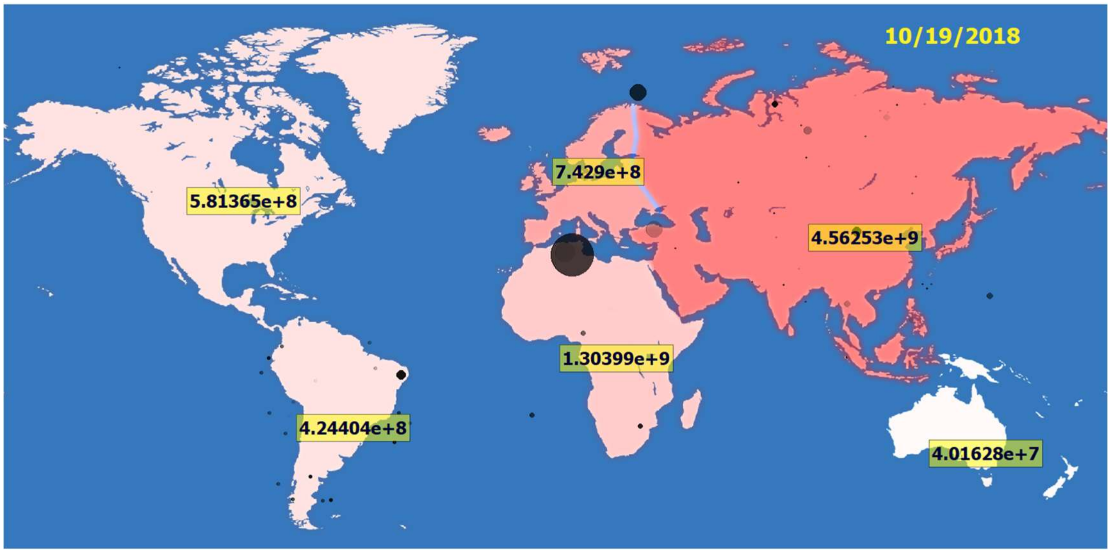

Figure 2. View of Simulation Five Months After Start

In the image for the five-month result above we can observe the diversity between continents based on their disasters. The qualitative results of this simulation which cannot be plotted on the standard graph is in the distribution of disaster within each continent. Although their positioning is someone random and doesn't correlate to realistic values, we can see the differences of relative death tolls per disaster within each continent. Additionally, based on the researched values of population growth within continents, we can also calculate the amount of impact the occurred disasters had on the population (given the researched population growth doesn't already take into account deaths by disasters). At this stage the software doesn't do this automatically therefore we can perform the following simple calculation and display the found information in a graph. Note: more precise population values were printed out in the console and used for this calculation.

Calculation:

$$\begin{aligned} &\mathit{Disaster\ impact}\ (\text{people deaths}) \\ &\quad = (\mathit{intial\ population}\ (\text{people}) + \mathit{daily\ population\ growth}\ \left(\frac{\text{people born}}{\text{day}}\right) \\ &\quad * \mathit{time\ passed}\ (\text{days})) - \mathit{simulation\ \ population\ value}\ (\text{people}) \end{aligned}$$

Graph:

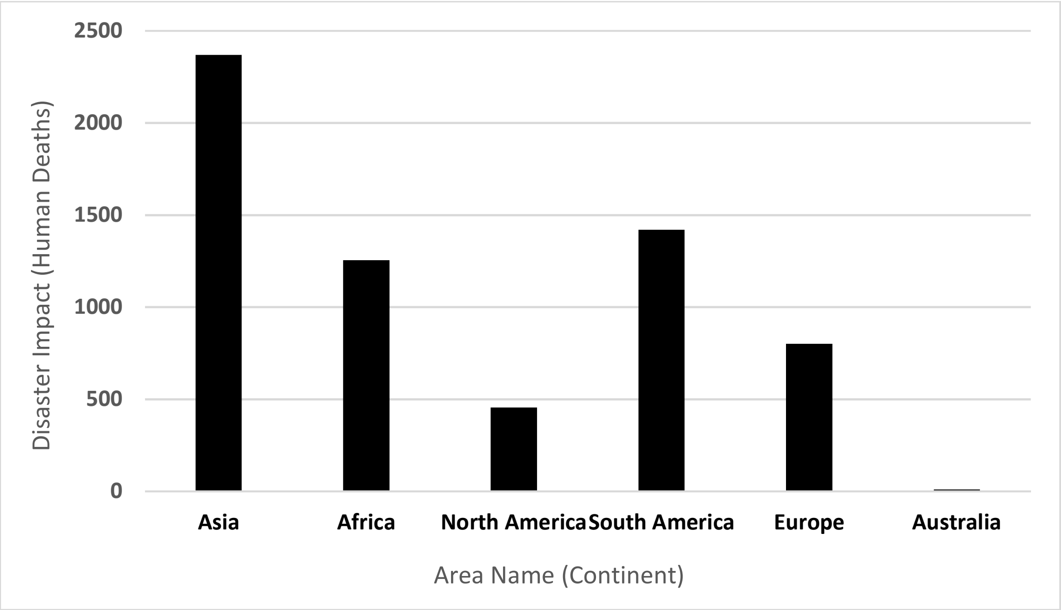

Figure 3. Graph of Population Disaster Impact (for 100 days) for each Continent on Earth (Excluding Antarctica)

From the above graph we can deduct that disasters have the most effect in Asia, which makes sense as it initially had the highest population density (as seen in Figure 1. by the continent being most red). The other results are more intersecting though, in that South America has much more disaster impact than North America even though the population densities between the two continents wasn't much different. The results become clearer as we inspect the disaster occurrence rates and see that South America has much higher instances of lightning strikes and other disasters per year. Other similar information can be observed and investigated in a likewise manner.

The different design changes of this project are hard to summarize in a concise manner. Therefore, the below info consists of the description of the final implementation of the program.

Kostiantyn Makrasnov
Yuriy Fedas
CS273 – Spring 2018
May 17, 2018

Backend Summary (by Yuriy Fedas)

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.

Short Frontend Summary (by Kostiantyn Makrasnov)

The front end of the application consisted of both inherited and auto generated Qt classes. The MainWindow class handled user input and different Graphics items inside a QGraphicsView. The polymorphism in our program consisted of the inheritance of the base classes QGraphicsItem and QGraphicsTextItem within which we had to implement our own paint function that performed the items animation and a bounding box function that returned the size of the item. In order to be able to replay the calculated frames in the future, the design of the backend somewhat changed from our initial view of the program. A SimDeltaOutcome class was created along with GlobalDay and ContinentalDay. These classes acted as snapshots that stored the respective level of information and allow for the front-end animation (paint) functions to access the needed day's info in a relatively short amount of time (maps were used for to achieve fast access). Note: to see the full UML Diagram please see FinalSpec.docx