CS Capstone Design Document

December 4, 2017

Ancestry Data Viewer

Prepared for

Ashley McGrath ____________________ Signature ________ Date

Prepared by

Group 22

Team Ancestry Data Viewer(ADVR)

YongPing Li ____________________ Signature ________ Date

Monica Sek ____________________ Signature ________ Date

Le-Chuan Chang ____________________ Signature ________ Date

Abstract

The purpose of this design document is to describe the design of our software to our client. There are many pieces of our software. We highlight the choices we made and how it will do the job for each part to demonstrate our design viewpoint. Each choice we made is based on the research we each did separately and we combined our work to create this design.

CONTENTS

1 Introduction

1.1 Scope

The purpose of the Ancestry Data Viewer software is to get data from a GEDCOM file and display the data in a clear and easy to read manner. The GEDCOM file format is difficult to read in plain text, so the Ancestry Data Viewer software should be capable of accessing the GEDCOM file to construct the family tree contained within the file. The details of the design is described and is separated into different parts of the software.

1.2 Purpose

The purpose of the Software Design Document is to demonstrate the different parts of the software, how it does the job, and why we choose to do it this way. It also acts as a guideline when we implement our software in the future. Changes could occur during our implementation and we will need to modify the document in order to keep track of what we have done to create the software.

1.3 Intended Audience

The intended audience of this document is our client, instructor, and teaching assistant. This document is part of the process for developing the software required by our client. Our instructor and TA will provide assistance to guide us throughout the development process.

2 Design viewpoints

2.1 Parser

We are given a GEDCOM file and the first task to we need to perform is to use a parser to capture the relevant data. There are existing parsers on Github, but there are concerns about the performance, and there is no detailed information about the implementation. Therefore, we have decided to create our own parser, because we are only able to understand the performance of the existing parsers after they are implemented.

The GEDCOM file has its own unique format. This format uses text as a symbol for indicating the types of information. The first step for us to create our own parser is to analyze the GEDCOM file format. We need parse between information we need and information we do not need when coming up with an algorithm. The parser will use this algorithm and filter out the unnecessary information and collect the needed information. The information that is to be captured are the names of people and their relationship with others. The relationship includes husband-wife, parent-child, and divorce.

For testing purposes, we should test our algorithm with both small GEDCOM files first and larger files. It is easy to identify whether we collected the correct information in the small files. If the files are too large, we will need to spend a lot more time confirming whether we got the correct information, likely through an automated process.

2.2 Data Storage

We will get raw data after we parse the GEDCOM file. The raw data needs to be manipulated and stored for later access because we do not want to waste time on parsing the data from the GEDCOM file again. We will not be using a database for storing the data because this will cost us time and effort. Using a database is unnecessary because we can store the data on the local desktop. Another benefit of storing the data on the local desktop is because we are able to store the data in different formats. We could manipulate the data and make it into a CSV file or other formats for future usage

In order to test whether we stored all the information correctly, we also need to store the raw data into a file and compare the two files. The content should be similar but in different formats.

2.3 Data Structure

The data can be used by other tools after it is stored, but it is not easy to create the lineage view and perform the find common ancestor function. We can insert the data into a data structure to making access easier. There are certain data structures that can easily perform search functions and illustrate relational data. We have decided to use a graph as our data structure because it is able to illustrate the different types of relationships. Graphs have vertices and edges, so we can use names from the GEDCOM file as the vertices in the graph and relationships as edges. The edges can not only could display the connection, but it can also store a number to identify the different types relationships.

The testing of the data structure is to compare it to the data stored. We need to confirm whether the names and relationships are displayed correctly. The files to be tested should have multiple people's data in order to create a good graph to check. We shall check with both small files and larger files.

2.4 User Interface

Our decision for the UI tool is the UMG UI Designer. This is because the visualization will be done via the Unreal Engine, therefore, the most compatible tool is the UMG UI Designer, which is already built-in.

The UMG UI Designer will connect to the visualization in both VR and application view, though each will have their own UI. The UI created for the application view will be designed for simplicity, and will be similar to various other applications similar applications, such as Word. Since UMG UI Designer works for both VR and application view, there is no need to use a different tool for either mode, however, since VR is fundamentally different from application view, and there are significantly fewer VR applications, there will be a completely different UI for VR.

While the UI is unlikely to be directly implemented until after at least some of the visualization is set up, it is possible to still design it, including developing a persona for potential users, and creating paper prototypes. By using these two methods of design, a clear picture regarding how intuitive the design is should be painted, and thus problems with UI can be predicted and remedied. Additionally, the paper prototypes can be used to showcase the design to the client and various potential users for feedback before implementation.

After implementation, the UI must still be tested. For this purpose, a user study will be used to test how intuitive and how aesthetically appealing the user found the UI, both in VR mode and in application mode.

2.5 Display Algorithm

We will need to design an algorithm, most likely using the Ahnentafel system as a base, and heavily borrowing concepts from other graph-based mathematics. The algorithm can then be designed to display all nodes in a visually clear manner, in this case meaning that no nodes overlap and the connections refrain from crossing other connections and nodes as much as possible. Completely negating intersections may be impossible, due to circumstances such as incest or three married children.

The design of the algorithm is likely to be the most complicated portion of the application, therefore, it should begin as soon as possible, starting with researching graph theory.

Once the algorithm is designed, it must be integrated to be visualized. This will be done by creating nodes from the data retrieved and stored by parser, and then sending it to the visualization aspect of the application to visualize.

The algorithm will require a large amount of flexibility, for instance, on a simple family, a tree would suffice. This can quickly escalate, however, since the spouses family can be included, and the spouses family can include another spouses family, and so on, in order to generate a massive graph. Therefore, to test the algorithm, steadily larger GEDCOM files must be used as input, and their generated graphs must have no overlap in nodes and the connections must be at least somewhat reasonable to follow.

2.6 2D Visualization

In this section, we will be examining the Unreal Engine and how it will be incorporated for the 2D visualization of our program. We will primarily discuss the properties of nodes that are used for display and the camera view of the program.

Once data has been processed and displayed, we can use other features in Unreal to create and adjust visualization components.To customize each node, there is a material editor window, which allows it to have a certain color. Lines can also be nodes, with colors representing the relationship. For example, the relationship between two members who are divorced but have children can be represented by a red line. Text will also be attached to nodes to display a members name.

To view the data, a camera can be placed on a static fixture, and the focused node will be placed in the center of the camera view. From here we can use other parameter tools to help display the data, for instance, the scaling tool to manipulate the nodes. Both objects and camera can be moved in an XYZ-direction, using arrows to their corresponding direction.

The UI will feature the ability to change views, including the ability to switch to VR mode if it is enabled. Unreal can be used for the direct lineage option, by using it to only display a certain relationship between nodes, and hiding all other nodes.

In the end, using Unreal to display and edit nodes is very simple. We are not looking for complex graphics or action performance, so this minimalistic display is all that is required. Due to the fact that the Unreal Engine is attempting to reach out to a wider audience, various new built-in plugins to benefit non-gaming application have been released, which will greatly assist in the development of the application.

2.7 3D Visualization

In this section, we will be examining the Unreal Engine and how it will be incorporated for the 3D VR visualization of our program. We will primarily discuss the properties of nodes that are used for display and the camera view of the program.

To begin designing in VR, Unreal is developed with built-in virtual reality tools. The VR configuration is quite simple, and launching a preview of the application would be useful later, to test the application.

Nodes will be added for data display, and if we cannot find a certain model then we can use the built-in Rhino 3D to build our own model and import it. The primary display node would be a thin-width box that will present the members name. It will be displayed as a solid color, possibly adding glossy texture to make it more appealing without interfering with the text. The size of the node should be about the size of a street sign, so it is not too overwhelming. Each node should share the same properties, this includes having a unique text attached to them so it displays its data.

The relationship between the nodes will be connected through a pipe-like structure. The tree structure itself should be static and not be too modifiable by the user. The user should only be able to interact with the tree by selecting a node, after which it should be highlighted in a dim glow, and all of its relationship should be highlighted as well. These color highlights will be triggered by certain button actions that will be configured using the blueprints.

The camera should be initialized at the center of the ancestry tree. If a member is selected, then that member should be shifted to the center of the screen. The camera should be able to move across the tree, instead of moving the tree itself, to prevent the tree from getting out of bound.

We are not trying to develop hyper-realistic graphics for this project, so having simple colors and structures is what we are implementing. This means we can focus our performance cost onto other aspects, rather than graphics. Camera configuration is a very important aspect, fortunately, Unreal has a user-friendly camera motion setup.

In the end, we arent trying to develop hyper-realistic graphics for this project so having simple colors and structures is what we desire. This means we can focus our performance cost onto something else if needed, rather than high graphics. Camera configuration is a huge part in this section, luckily Unreal Engine has a user-friendly camera motion setup.

2.8 VR API

In this section, we will be examining the Unreal Engine and how it will be incorporated for the VR API of our program. The primary topics we will be discussing are hardware compatibility and button configuration.

To begin implementing VR, Unreal contains built-in API VR plugins that help integrate VR hardware. We will be integrating either the Oculus Rift or the VIVE to our application, both of which can be handled by Unreal without additional configuration. Motion remote controller compatibility is also built in, for simple button configuration. The built-in plugin provides a headset and controller tracker, so the headgear and controllers motion will reflect the users movement. If the user decides to look behind them, they should see a plain colored background.

Events for the controllers will be configured using the Unreal Engines blueprint interface or behavior trees. The primary events we are looking for are button triggers with the controller. The user will be able to select a node using the controllers. When a node is selected it and its related nodes will be highlighted.

To shift the screen, the user will click a secondary button and the screen will move in the appropriate direction. The object will be a flat workspace, so the camera orientation will move linearly in the X- or Y- axis. To zoom in and out of the ancestry data, the camera actor will move on the Z-axis.

In the end, our software does not require complex interaction, aside from shifting positions and displaying different tree features. The Unreal Engine has a built-in VR API that includes head tracking and controller configuration, and its blueprint interface will make it easy to set up button event triggers to manipulate the application.

3 Conclusion

This document holds the important information about the design we made currently. We decompose the large problem into smaller pieces. For each piece of the software stated, we describe its functionalities and our choices for each part is supported by research to justify our choice. This document will act as a guideline for our future implementation because there is a detailed description of how each part works. It will also act as a record book that needs to be modified if we had to change our design.