CS Capstone Design Document

December 3, 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 stakeholder. 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 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 and the Ancestry Data Viewer software is capable of access the GEDCOM file and construct the family tree. The details of the design is described and it 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 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 the needed assistance to guide us throughout the process.

2 Design viewpoints

2.1 Parser

We are given a GEDCOM file and the first thing we need to do is to use a parser to capture the relevant data. There are existing parsers that we could use but there are concerns about the performance. Most of the existing parsers for GEDCOM files are on Github and there is no detailed information about the application. Therefore, we decided to create our own parser because we are unable to understand the performance of the existing parsers until we try it out.

The GEDCOM file has its own unique format and the data are not randomized. 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 to able identify the information we need and information we don't need with some kind of trend or symbol and come 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 small GEDCOM files first and then try it on larger files. It is easy to identify whether we collected the correct information in the small files.

2.2 Data Storage

We got raw data after we parse the GEDCOM file. The raw data needs to be manipulated and stored for later access because we don't want to waste resources and time on capturing the data from the GEDCOM file again. We won't be using a database for storing the data because this will cost us time and effort. Using a database is just unnecessary because we could store the data on the local desktop and it is much easier. 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 directly used by other tools after we stored it, but it's 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. Using the data structure can easily perform search functions and illustrating relational data. We decided to use graph as our data structure because it is able to illustrate the different types of relationships, unlike other data structures. Graphs have vertices and edges. We will be using names from the GEDCOM file as the vertices in the graph and relationships as edges. The edges not only could display the connection but it can also store a number to identify the different relationship.

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 start with small files and then larger files.

2.4 User Interface

Our decision for the UI tool is to use UMG UI Designer. This is because the visualization is done via Unreal Engine, therefore, the most compatible tool is the UMG UI Designer. Additionally, the UMG UI Designer is simple to use and is already built in to Unreal Engine, so it should be straightforward to use.

The UMG UI Designer will connect to the visualization in both VR and application view. The UI created will be implemented within the application, with the UI being designed for simplicity. 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, there will need to be at least a slight amount of redesigning.

While the UI is unlikely to be directly implemented until after at least some of the visualization is set up, it is possible to spend some quantity of time designing 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 the UI can be modified as needed. 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 starting point, 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 there is no overlap and the connections refrain from going past each other as much as possible. Due to various familial relationships, completely negating intersections may be impossible.

The algorithm is likely to start with pre-existing formulas, including the Ahnentafel system and various implementations of binary trees, though it will need to be expanded to other graph systems. 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 integrating various graph display algorithms.

Once the algorithm is designed, it must be integrated to be visualized. This will be done by creating nodes for the visualization aspect of the application to read, then creating the connections. The algorithm should also specify what type of connection is used, such as sibling or spouse.

Though the algorithm will be created for this application, it will be heavily dependent on previous graph data. On a simple family, a modified tree would suffice, simply having parent nodes linking to a child which is linked to a new node, representing their spouse, which link to their children. This can quickly escalate, however, since the GEDCOM file can also include a spouses family, which can include another spouses family, and so on, in order to generate a massive graph. Therefore, to test the algorithm, steadily increasing GEDCOM files must be used as input, and they must still output data in a clear to read manner.

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 adjust visualization components. Unreal Engine allows developers to work with materials and texture. To customize each node, there is a material editor window included. The displayed nodes will be set to a plain color and lines will be their own node with certain classification. They will be classified under certain relationships, which will be represented by different colored or textured lines. For example, the relationship between two members who are divorced but have children will be represented by a dotted line. Texts can also be attached to nodes, this will be utilize to display a members name. Nodes can be duplicated and grouped together, this will make sharing properties between similar nodes easier. The background can also be changed to a neutral gray scale color.

To view the data, there is a camera option in the folder of preset classes. A camera actor can be placed onto the display level. Camera configuration can be modified through a menu. Our camera angle will be simple, placed as a static fixture where 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 like uniform scaling and tiling awareness with objects. The scaling tool will be helpful to manipulate the nodes around, rotation tool is also available but wont be necessary since we are examining a flat data. Both objects and camera can be moved in an XYZ-direction, using arrows to correspond to a direction.

A node display for each of our software function can be displayed on a menu bar, this will include the desktop to VR toggle. To set up the functionality for visualization option, a blueprint menu can be used to set up triggers and actions. An open-source data can be found that toggles between the two displays. The blueprint menu can also be used for the direct lineage option, by using it to only display a certain relationship between nodes. The relationship, for example, would be called direct this way only nodes that are direct related to each other will be displayed.

In the end, using Unreal Engine to display and edit nodes is very simple. We arent looking for complex graphics or action performance, so this minimalistic display is what we desire. Because Unreal Engine is attempting to reach out to a wider audience, they have release new built-in plugins to benefit non-gaming 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 Engine is developed with a built-in virtual reality component. The VR configuration is quite simple, just check the enable box for Gear VR so that it is supported for display. Launching a preview of the application would be used later to examine the application, to do this we would just select VR Preview.

Nodes will be added for data display, there are many preset actors that can be used. However, if we cant 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 not be large enough where it takes the whole screen, but can relatively be 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 will move uniformly.

The relationship between the nodes will be connected through a pipe-like structure. The tree structure itself should be static and not be too interactable by the user. The user should only be able to interact with the tree by being able to select on a node. When a node is selected, it should be highlighted in a dim green glow, all of its relationship should be highlighted in their corresponding colors as well. These color highlights will be triggered by certain button actions that will be configured using the blueprints.

The VR configurations allows the developer to choose a default point for the user when they have a headset on. The camera should be initialized at the center of the ancestry tree. If a member is selected initially then that member should be shifted to the center of the screen. The camera should have mobility properties for when the user shifts the tree. Instead of moving the tree data itself, otherwise things can get out of bound.

In addition to the VR experience, if the user needs to type a name into the search bar. An on-screen keyboard should appear, it should be presented as another window object that is facing flat towards the camera.

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 Engine contains built-in API VR plugins that helps us integrate VR hardware. We will be integrating either the Oculus Rift or the VIVE to our application, both of which is handled by Unreal Engine without additional configuration. Motion remote controllers setup is also built into the software for simple button configuration. The built-in plugin provides a headset and controller tracker, thus the headgear and controllers motion is already accounted for and will reflect the users movement. If the user decides to look behind them, it should just reveal a plain colored background.

Events for the controllers will be configured using their blueprint interface or behavior trees. The primary events were 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 shall be highlighted and so will its relationship with other nodes. Since all nodes will react like this, they can be duplicated from the same initial object and share the same properties. Buttons on the menu bar will change to a darker shade upon being clicked and then redirected to the right display.

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

In the end, our software isnt asking for complex interaction besides from shifting positions and displaying different tree features. Luckily Unreal Engine has a built-in VR API that includes head tracking and controller configuration. The blueprint interface makes 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.