Team M
Cycle One
Requirements Document

Functional Requirements:

1. Given a person, determine if they are known to the app
    a. Uses: looking for an ancestor, preventing people from being added twice
    b. Search for:
        i. Name
        ii. Birthday
        iii. City of birth/residence
        iv. All 3 of these conditions are useful because it narrows down the number of people who have the same names, birthdays or cities
    c. User can input the information from (b); return a list of people who apply with this information
2. Add a person to the app
    a. Automatically pass (1) to check if the person is in the app or not
    b. Information to specify:
        i. Name
        ii. Birthday
        iii. City of birth/residence
        iv. Any known family relationships (spouse, parents, children)
    c. Uses: filling out the user's family tree
    d. Limits: the specified information is required to add a person to the app
    e. User can enter in information listed in (b) to generate a new node on the family tree graph with the associated information
3. Collect information on each family
    a. Read in a text file
        i. contents:
        ii. people (name, date of birth, birth place, current residence, etc.)
        iii. people's relationships (spouse, parents, children)
        iv. for each specific person:
    b. Save added information back to the text file
        i. When information is updated on the app, fill in this information on the text file
    c. This will automatically fill in the family tree based on the information in the file
    d. Uses: adding large amounts of data to the app so it does not have to be done manually (by adding each person with (2))
        i. Saves large amounts of information to a text file so it is not lost

   e. User can upload a file to automatically generate or update an existing graph with the information in the file
4. Record the start and end dates of a partnership
   a. Read in the file:
      i. Create the relationships for spouses
      ii. Read in the dates from the partnership section of the file
      iii. First date signifies beginning of relationship, second date is the end
   b. Save added information back to the text file
      i. When information is updated on the app, fill in this information on the text file
5. Record children in a new or existing partnership
   a. User finds a specific partnership in the tree
   b. User adds a child:
      i. Information to specify:
         1. Name
         2. Birthday
         3. City of birth/residence
         4. Any known family relationships (spouse, parents, children)
   c. Save added information back to the text file
      i. When information is updated on the app, fill in this information on the text file
6. For any person, find people of a specified relationship
   **Utilizing the relationship using "mother of"," father of" or "child of" is the least erroneous way to navigate a specific relationship. This is because there can be a possibility of multiple grandparents and multiple grandchildren within any branch of the overall family tree.
   a. Parent is Mother(Full name) is mother of "c", Father(Fullname) is father of "c".
   b. Grandparent is searched through "mother of".
   c. Child is Child(Full name) is child of "a"
   d. Grandchild is searched through "child of".
   e. Cousin is searched through "child of".
   f. Partner is a person who does not have "child of" defaults as partner.

7. Determine if two people are related (they will have a common ancestor somewhere "up" in their family tree)
   a. Create two lists to collect the data of each person as it is scaling towards the root of the family tree
   b. Compare each person added to the list to see if they match
   c. If matching, this is the related ancestor

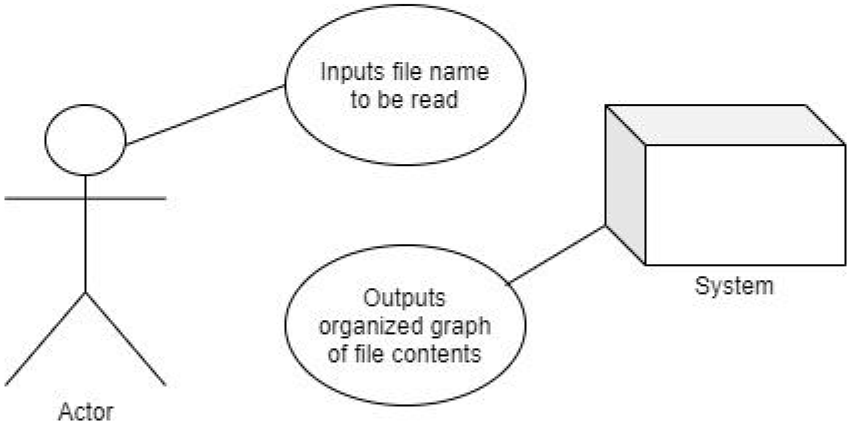
d. If root of family tree is reached and no relationship established, no related ancestor exists
e. Limits: only shows first related ancestor, does not continue to check for other ones

Nonfunctional Requirements:

1. Use Java v11 and IntelliJ for our program development. Save our work in the course GIT repository for your team: Comp330Fall2020TeamM
2. Structure the system so that the user interface is separate from the logic and searching functions.
3. Have a well-structured functional decomposition of the app into separate parts. This decomposition should support separate development of key components by individual programmers.

Use Case Models/Narratives:

**Event: Read in file**


Inputs file name to be read
System
Outputs organized graph of file contents
Actor

Use Case Narrative:

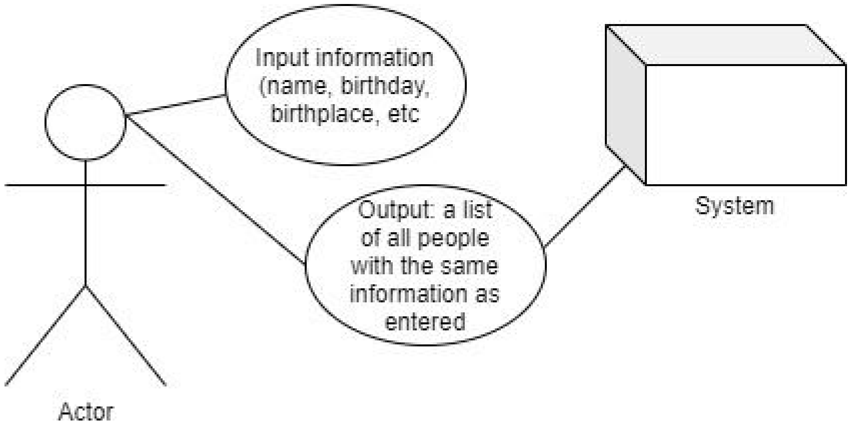
User

1. The user uploads a file to be read into the app

System

1. The system reads the file and separates the pieces of information into a graph data structure
2. The system outputs the graph to the user

**Event: Determine if a person is known to the app**


Input information (name, birthday, birthplace, etc
Output: a list of all people with the same information as entered
System
Actor

Use Case Narrative:

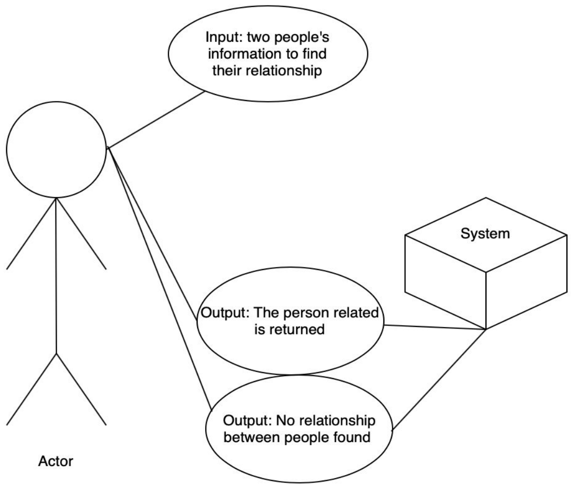
<u>User</u>

1. The user is given options for information to be entered
    a. Name (required)
    b. Birthdate
    c. Place of birth
    d. Spouse/Parents/Children
2. The user submits the entered information for the search

<u>System</u>

1. The system searches through its list of people until a matching name is found
2. The system searches through the information of that found name, and if the information matches what the user inputted, the system outputs the person along with the associated information
3. This process continues until the list of people is completely searched

**Event: Determine if two people are related**


Input: two people's information to find their relationship
System
Output: The person related is returned
Output: No relationship between people found
Actor

Use Case Narrative:

<u>User:</u>

1. User inputs names of the two people they want to find a relationship for
2. User submits the info into the search

<u>System:</u>

1. The system will traverse the tree towards the root for each person
2. It will create a list for each person of related ancestors and compare each person added, checking if they are identical
3. If the system finds a person who overlaps for both, this person is outputted to the user as the member both are related to
4. If the root of both trees is hit and no relationship is found, the system outputs that the two people are not related.