Group Project Report - Analysis and Design

CS3343

Group 13



Group Members:

|  |  |  |
| --- | --- | --- |
| Full Name | SID | Role |
| Luka Moderc | 55594415 | Project Manager |
| Andela Basic | 55594403 | Assistant Project Manager |
| Uros Cvijanovic | 55304127 | Testing Engineer |
| Phudis Dawieang | 55411086 | Programmer |
| Balaji Varun Aditya | 55304510 | Programmer |
| Jeffers Chan | 55606049 | Testing Engineer |

# Table of Contents

Analysis and Design

Design Constraints

Use cases and Requirements Specification

Class Diagram

Sequence Diagram

Design Principles

Design Patterns

Program flow and Algorithm

Stable Matching Algorithm

Overview of the Stable Matching Algorithm execution flow

Floyd Warshall Algorithm

Overview of the Floyd Warshall Algorithm Execution Flow

# 

# 

# 

### Design Constraints

Background

If it is executed smoothly, our software should output the list of matches, from the input file (set of users). The matching is performed by Stable Matching Algorithm, which heavily relies on the score calculation process. The score is calculated as the summation of three sub-scores, location score, zodiac score and age score. Location score utilizes the longitude and latitude values as parameters for a simple calculation which returns the number representing it. Similarly, zodiac score utilizes the years of birth and returns a number representing the extent to which users are compatible in Chinese Zodiac. Finally, age score represents how well one user fits the age requirements of another user, i.e. whether and to which extent he fits into a closed interval [ lowerAgeBoundary, upperAgeBoundary ].

Design Constraints

The initial challenge was defining the score calculating function. Naturally, several conditions arised. Firstly, it is crucial for a function not be symmetric, i.e. if a user A scores S at user B’s ranking, then the user B should not score the same value of S at user A’s ranking; rather, the function should return the different result based on the their role. Simply, u.calculateScore(v) should not equal v.calculateScore(u). Secondly, suppose the users were denoted as M1, M2… Mn, F1,F2,...Fn where M indicates males and F indicates females. Consider the ranking of M1 (the list of females based on their score at M1). The second requirement states that F1,F2,...Fn should not all have the same result in the ranking of M1. The similar requirement holds for any other user.

Taking these two requirements into consideration, it is obvious that the better design requires more parameters, as that would increase the precision and distinguish the scores more. After careful examination, we decided to calculate the score according to the following formula:

Adding more criteria into consideration would undoubtedly increase the program’s precision. However that was not our main focus.

### Use Case And Requirement Specification

# 

# Use Case of Enter Details

|  |  |  |
| --- | --- | --- |
| Use-Case Name: | Enter Details | |
| Actor(s): | Primary: User | |
| Description: | This use case describes the act of the user entering the details into the system | |
| Reference ID: | DASx01 | |
| Precondition: | - | |
| Trigger: | User enters the app | |
| Typical Course Of Events: | Actor Action | System Response |
|  | Step 1: User enters the appStep 3: User enters the details and clicks on the ‘continue button’ | Step 2: The system displays the fields to be filled |
| Alternative Courses: |  | |
| Postcondition or Results: | Program successfully collects all information and moves on to validate it | |
| Assumption | N/A | |

# 

# 

# Use Case of Validate Details

|  |  |  |
| --- | --- | --- |
| Use-Case Name: | Validate Details | |
| Actor(s): | Primary: System | |
| Description: | This use case describes the act of the system validating the users input | |
| Reference ID: | DASx02 | |
| Precondition: | User enters the details | |
| Trigger: | User clicks ‘continue’ button | |
| Typical Course Of Events: | Actor Action | System Response |
|  |  | Step 1: Check the various fields like Name, Location, sexStep 2: Use the details to calculate shortest distance, matches and scores. |
| Alternative Courses: | Step 2: [Extension point : if details entered is invalid, invoke the use case *Prompt Error Message*] | |
| Postcondition or Results: | Program successfully gets the information and calculates various scores. | |
| Assumption | N/A | |

# 

# 

# Use Case of Request Score

|  |  |  |
| --- | --- | --- |
| Use-Case Name: | Request Score | |
| Actor(s): | Primary: Actor | |
| Description: | This use case describes the act of the user requesting scores. | |
| Reference ID: | DASx03 | |
| Precondition: | Scores have been calculated from input | |
| Trigger: | User wants scores | |
| Typical Course Of Events: | Actor Action | System Response |
|  | Step 1: User requests for the scores. |  |
| Alternative Courses: |  | |
| Postcondition or Results: | Program successfully gets the input | |
| Assumption | Details have been validated | |

# 

# 

# Use Case of Print Scores

|  |  |  |
| --- | --- | --- |
| Use-Case Name: | Print Scores | |
| Actor(s): | Primary: System | |
| Description: | This use case describes the act of the system displaying scores | |
| Reference ID: | DASx04 | |
| Precondition: | User request scores and the score has been calculated | |
| Trigger: | User enters 2 | |
| Typical Course Of Events: | Actor Action | System Response |
|  |  | Step 1: The system displays the scores |
| Alternative Courses: |  | |
| Postcondition or Results: | Program successfully displays the scores | |
| Assumption | Details have been validated | |

# 

# Use Case of Request Shortest Distance

|  |  |  |
| --- | --- | --- |
| Use-Case Name: | Request Shortest Distance | |
| Actor(s): | Primary: Actor | |
| Description: | This use case describes the act of the user requesting shortest distance. | |
| Reference ID: | DASx05 | |
| Precondition: | distance have been calculated from input | |
| Trigger: | User wants shortest distance | |
| Typical Course Of Events: | Actor Action | System Response |
|  | Step 1: User requests for the shortest distance |  |
| Alternative Courses: |  | |
| Postcondition or Results: | Program successfully gets the input | |
| Assumption | Details have been validated | |

# 

# 

# Use Case of Print Shortest Distance

|  |  |  |
| --- | --- | --- |
| Use-Case Name: | Print Shortest Distance | |
| Actor(s): | Primary: System | |
| Description: | This use case describes the act of the system displaying scores | |
| Reference ID: | DASx04 | |
| Precondition: | User request scores and the score has been calculated | |
| Trigger: | User enters 1 | |
| Typical Course Of Events: | Actor Action | System Response |
|  |  | Step 1: The system displays the scores |
| Alternative Courses: |  | |
| Postcondition or Results: | Program successfully displays the shortest score | |
| Assumption | Details have been validated | |

# 

# Use Case of Request Matches

|  |  |  |
| --- | --- | --- |
| Use-Case Name: | Request | |
| Actor(s): | Primary: Actor | |
| Description: | This use case describes the act of the user requesting for matches. | |
| Reference ID: | DASx07 | |
| Precondition: | Matches have been calculated from input | |
| Trigger: | User wants matches | |
| Typical Course of Events: | Actor Action | System Response |
|  | Step 1: User requests for the shortest distance |  |
| Alternative Courses: |  | |
| Postcondition or Results: | Program successfully gets the input | |
| Assumption | Details have been validated | |

# 

# 

# Use Case of Print Matches

|  |  |  |
| --- | --- | --- |
| Use-Case Name: | Print Matches | |
| Actor(s): | Primary: System | |
| Description: | This use case describes the act of the system displaying the matches | |
| Reference ID: | DASx08 | |
| Precondition: | User request scores and the matches have been calculated from input | |
| Trigger: | User enters 3 | |
| Typical Course of Events: | Actor Action | System Response |
|  |  | Step 1: The system displays the matches |
| Alternative Courses: |  | |
| Postcondition or Results: | Program successfully displays the matches | |
| Assumption | Details have been validated | |

# 

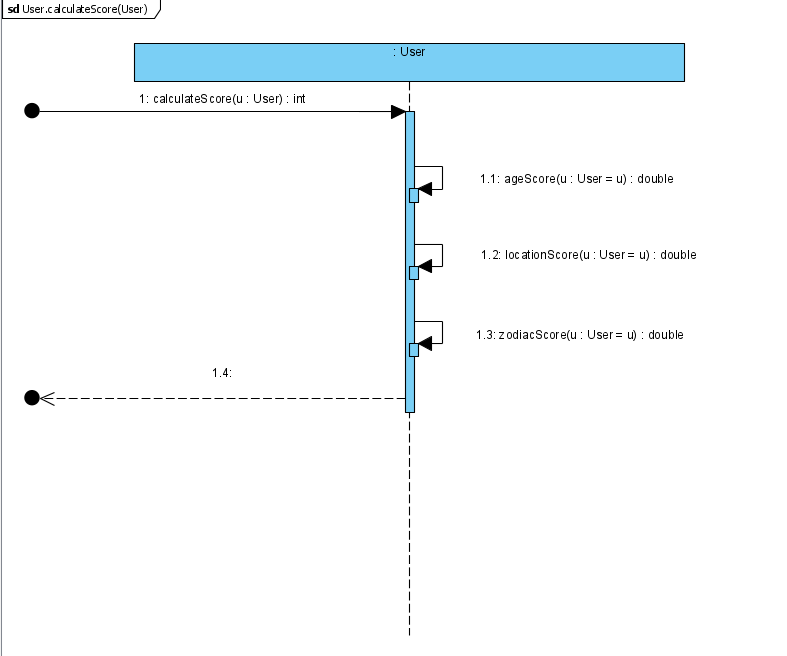
# Class Diagram

# 

# 

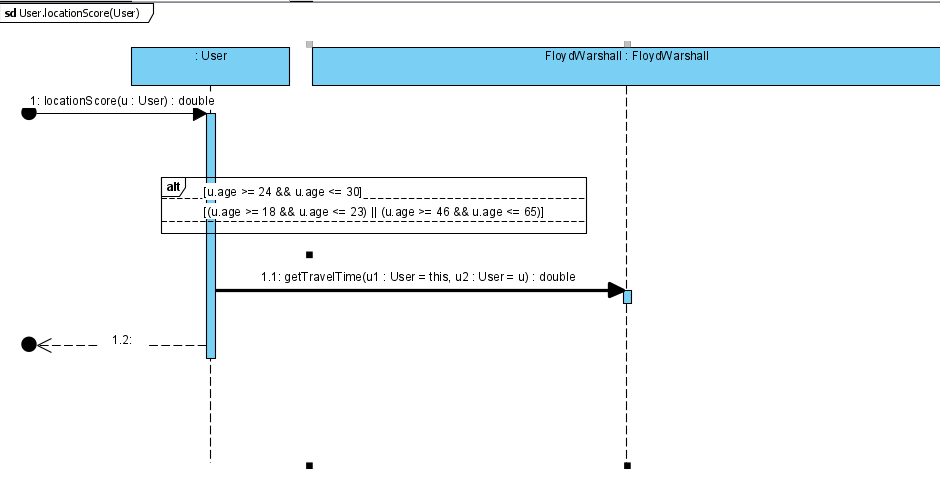
# Sequence Diagram

Sequence diagram for User.calculateScore



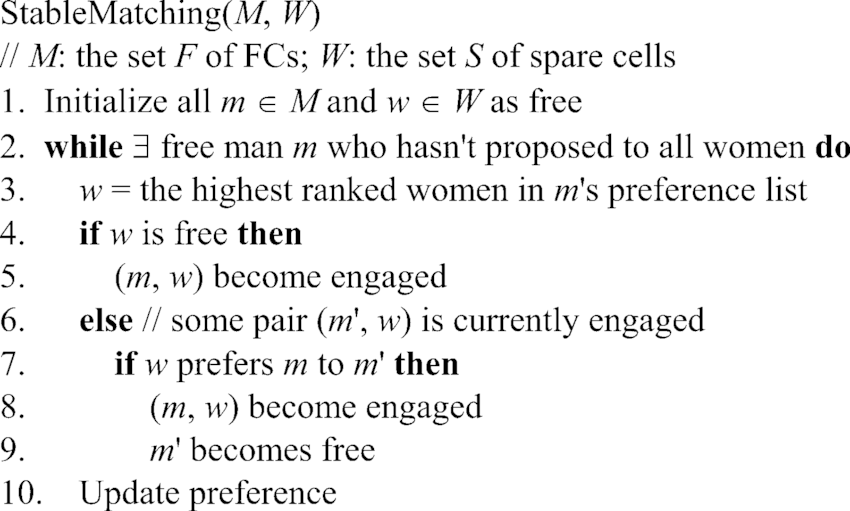
Sequence diagram for User.locationScore()

# 



Stable Matching Algorithm

The main algorithm implemented in our work is the Stable Matching Algorithm.

Stable Matching Algorithm FIg. 1

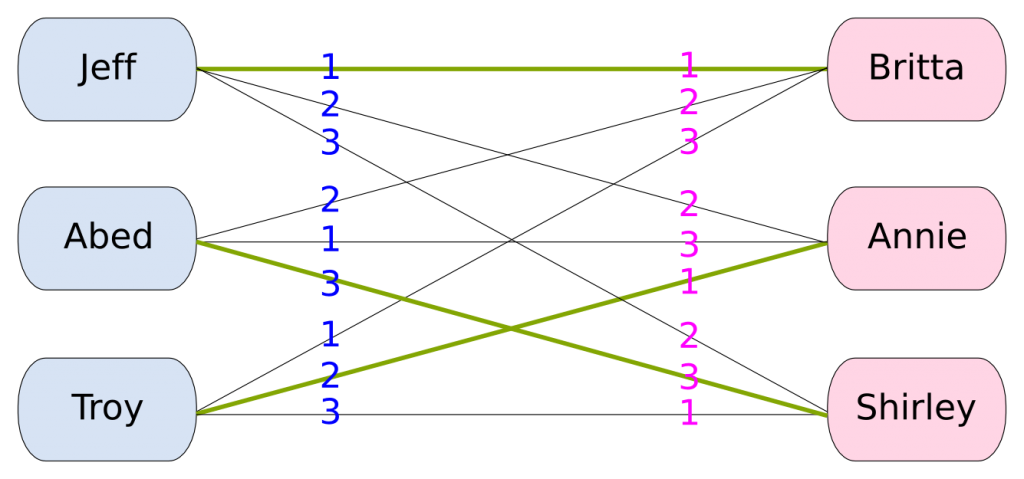
Our implementation can be viewed through two phases:

1. Taking inputs about users and setting up their preference ranking lists.
2. Execution of the Stable Matching Algorithm on the ranking lists and obtaining matches.

How these two steps are executed is described in the next section.

Overview of the Stable Matching Algorithm execution flow

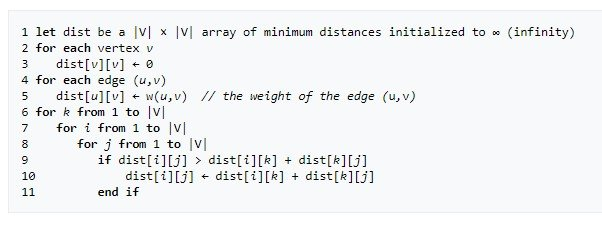
Personal information of our users, such as full name, age and age preference are encapsulated inside the User class object. Male and Female classes extend User class with additional fields and methods that are used inside the stable matching algorithm to enhance its speed of execution. Every user object keeps the ArrayList of rankings of all of the users of the opposite gender. ArrayList contains objects of Element class, one field of the Element object is the User and the other is his / her score. Once the rankings are set the stable matching algorithm can be run. We used the variation of the stable matching algorithm with the black list. In this variation, some members of one gender may not be matched with anyone although there are members of the opposite gender that have not been matched as well. We chose this variation as it allows us to ensure that no user will be matched with other users that do not satisfy the age boundary of the given user. Overall, the speed of our program is O(n^3), with the time for setting up of the ranking lists counted in, which can be regarded as the optimal implementation for our application. For more details regarding our implementation refer to the provided UML diagrams.

Stable Matching Algorithm FIg 2

Floyd-Warshall Algorithm

Floyd-Warshall Algorithm is an algorithm used to find the shortest path for every pairs in O(n^3)

The pseudocode of the Floyd-Warshall is provided below.



The Floyd-Warshall algorithm works by putting each node as an intermediary node between each path, then finding if it provides a shorter path or not.

Floyd-Warshall algorithm is used for the location scoring, which can be summarized to two steps

1. Find all-pair shortest path for all stations
2. Find the best station path for each male-female pair.

The implementation of how Floyd-Warshall is used in our program, and how the best station path is found, is described below.

Overview of the Floyd-Warshall Algorithm execution flow

The score used in the Stable Matching algorithm is based on Age score, Location score and zodiac score. Floyd-Warshall is implemented to determine the location score in this program.

First of all, a list of stations, their coordinates are included in the program. Moreover, the time used to travel between stations which are in the same line are included as a path for the graph.

After that, paths between all stations are calculated using Floyd-Warshall algorithm.

After the shortest paths between all pair have been found, the paths between all male-female pairs will be able to be discovered by when A and B is the station name.

To find the shortest ones, the program runs through all possible station pairs (similar to Floyd-Warshall that you find intermediates with shortest path, even though we loop through this only once because for every station C when we already do Floyd-Warshall for stations before)

### Design Principles

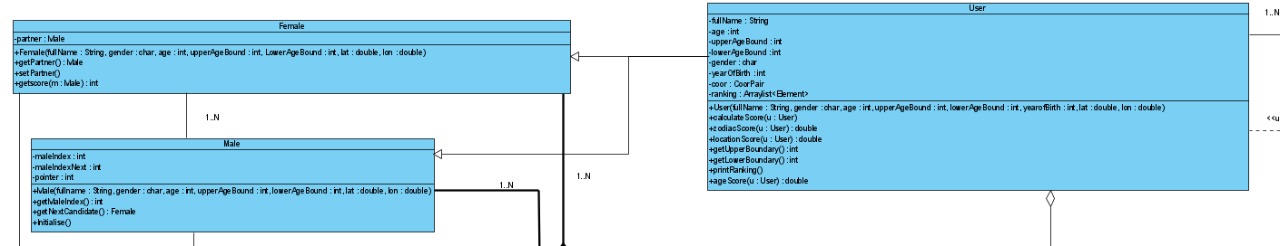
In our project we followed the SOLID design principles in order to keep our code as readable and clean as possible.

1. **Liskov substitution principle**

Classes Male and Female extend User class with additional fields and methods. MaleIndex field is used for fast retrieval of the score, O(1) time complexity, in the ranking lists of female users. On the other hand, the partner field in the Female class is used for keeping track of the found matches.

1. **Open-closed principle**

Through the implementation of the Male and Female classes we used the open-closed principle. The aim was to make possible for future extensions of the code that would include same sex matches but not only opposite sex matches.



1. **Single-responsibility principle**

Every class in our project was carefully designed so that it would serve only one functionality making our code simple and readable. For example, class Stable\_Matching is only used for the execution of the Stable matching algorithm over the ranking lists.

Through our work further design principles were strictly followed as well as the application of these principles is crucial for the future development of our project.

1. **Dependency inversion**

High-level modules do not depend on the low-level modules.

1. **Interface segregation principle**

### Design Patterns

1. **Singleton Pattern**

Stable\_Matching follows the singleton pattern as it is crucial that only one instance of this class exist.

