# *CSC207 Assignment Phase 0*

Group Name: Group-022

Team Members: Poorvi, John, Rahul, Aman, Alexia & Mary

October 15, 2021

## Domain

| | |
|---|---|
| 🟥 | Essential/Key Features |

Domain Name: **Matchmaker**

- Player Representation - *keeping track of stats/%wins + name, picture, age, bio*
    - Availability
    - Location (proximity)
    - Skill level (NTRP/UTR)
    - Ranking by District/Region
- Matching Algorithm
- Messaging feature
- Court data
- Rating feature (Feedback)
- Scheduling feature (Scheduler) - *linked to Player availability*
    - Save past matches
    - See who you played with in the past
    - Court Availability Probability Calendar
        - Ie. Google Maps “Busy Times” Timeline
        - Allow people to say if they want more players → dependent on current-day/active match preferences

## Specification

- Running the program allows users to swipe through potential tennis players to match with, prioritized by location and skill level.
- Users have the ability to create profiles detailing personal information, tennis skill levels and availability, that is displayed to other potential matches.
- When a mutual match is made (both users independently choose to match with the other), users have the ability to instant message each other, see information about their overlapping availability, and schedule matches.

- From their profile, users can also see a list of their upcoming matches, their past matches, as well as personal statistics about their wins and losses.

| player | |
|---|---|
| ● Store name, age, location, bio [store skill level], and player availability [store schedule flexibility] | Collaborators |
| ● Getter for everything above | matchMaker |
| ● Constructor sets ____ details | scheduler |
| ● Setters for everything above | |

| game | |
|---|---|
| ● Store date, time, court, players, restrictions | Collaborators |
| ● Getter for everything above | matchMaker |
| ● Constructor sets ____ details | scheduler |
| ● Setter for everything above | |

| Main | |
|---|---|
| ● Runs createProfile | Collaborators |
| ● Prompts user to input info | matchMaker |
| ● Runs the background process of outputting suggested matches | |

| scheduler | |
|---|---|
| ● Compare availability between two players | Collaborators |
| ● Return overlap between two players in an ArrayList | player |

| Manager | |
|---|---|
| ● Take in Manageable items and creates an ordered list of presentable matches | Collaborators |
| ● Returns each item from generate Match and feeds it to the UI | scheduler |
| ● Collects input from UI (Y/N) for each player and either skips to the next potential or adds each player to their respective list of matches | matchMaker |

| matchMaker | |
|---|---|
| ● Compare players by skill and location | Collaborators |
| ● Returns a weighted average as a double | player |

<u>Walkthrough</u>

Our user launches our app for the first time, and is prompted to create a player profile detailing their name, age, location, a bio, phone number, skill level and their availability. Once their profile is complete, our user is presented with algorithmically-determined player matches (based on match requests, availability, location and skill level), presented one at a time, with the option to either pass on or request to play with that player. If a player is passed on, the next potential player is presented; if a user requests to play with a player, a match is only created if both players have requested to play with each other.

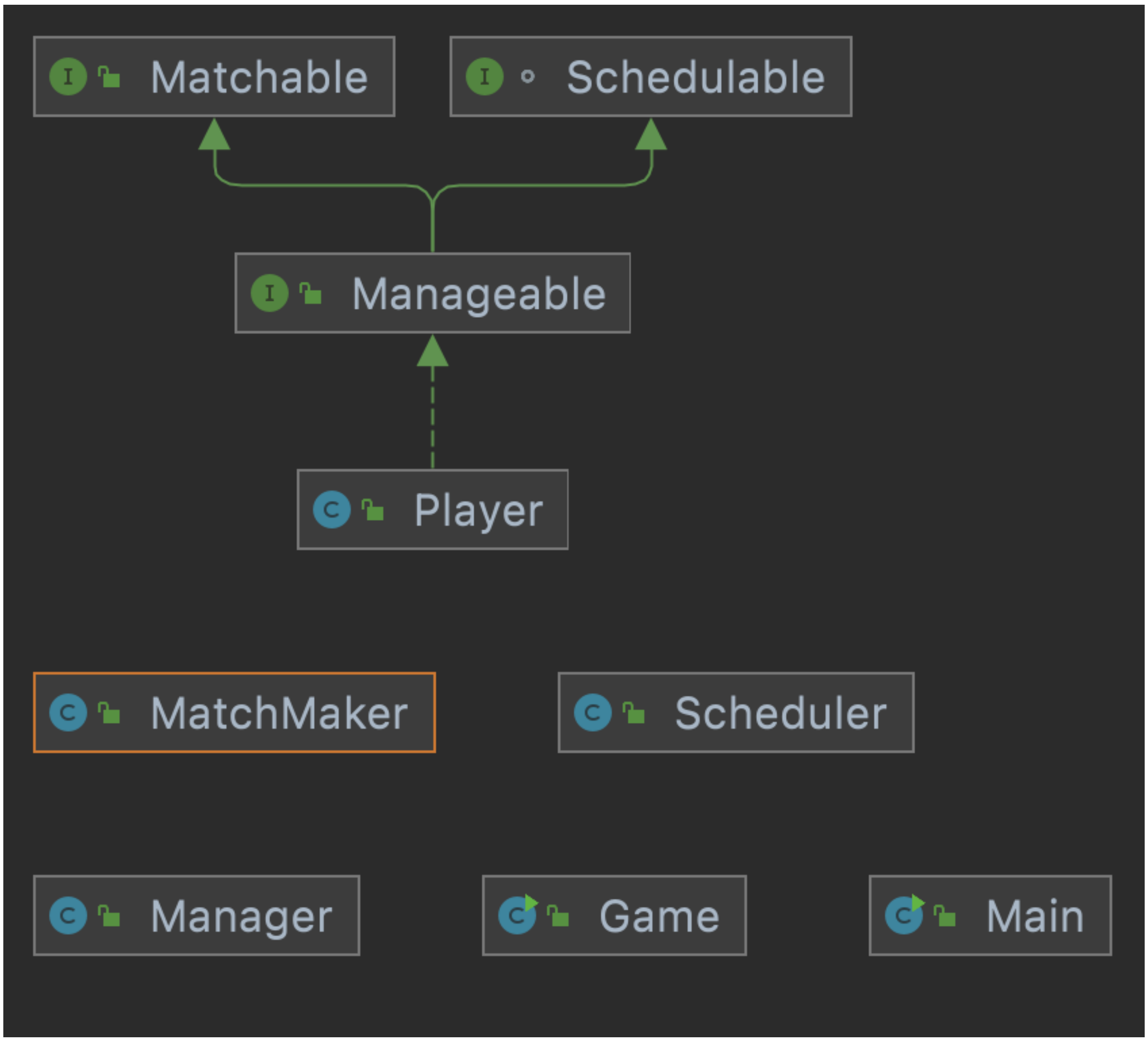
Once a match is created, both users' contact information is shared to each other.* It is up to the players to determine when, where, and how (singles vs. doubles) to play. To help with this process, a scheduling feature is displayed on the same page, highlighting overlaps in availability. When a consensus is reached, the user can officially schedule a game by choosing a time through the scheduling feature and input additional details such as date, time and the court location. Returning to the user's home page, the user has the option to view a log of their upcoming matches, as well as the past matches they have played along with statistics about their play history (e.g. % of games won).

*We understand that this is bad design, and issues a safety risk for users. This is currently a step only performed within the closed group of developers and will be modified to a more safe option in Phase 1.

## Progress Report

The specification for our program allows the users to create a profile that allows them to input their username, location, skill level, age, and availability. Therefore, each user would be exposed to other users who are similar in skill, location, and availability first and other users who do not have those similarities after those that are likely to be matched. As a result, since each user has their phone number within their profile, if two users like each other, they would be able to contact one another. In conclusion, the purpose of our program would be to create an environment where tennis players can be introduced to other tennis players based on similar region, skill level, and availability.

Our CRC model above showcases the two entity classes that our program is built upon (Player and Game), the three use case classes (scheduler, manager, and matchmaker), and one UI class (main). We can see a visual representation of how each of our classes are related below:



As a result, we can see how our entity class Player, which is what initializes the new profile for a user as well as holds information for all the associative getter and setter methods for all the attributes within a user's profile, implements Manageable which is the interface that extends two other interfaces known as Matchable and Schedulable. Those interfaces, respectively, depend on our use case classes MatchMaker and Scheduler. Within each of those classes their main responsibilities being comparing two players skill, location and availability. Main, the UI interface, will then display to the user potential players that result from the algorithms in MatchMaker and Scheduler.

Our scenario walk through then **(input scenario walkthrough from above)**

The workload was divided evenly amongst all the members, where each member worked on implementing a class and an interface if necessary. As a result, collaboration and communication were key to compiling the skeleton program. In addition, the unit test as well as scenario walk through was written up after figuring out as a group the initial skeleton program. Throughout this process, we have seen that communicating (i.e., meeting up in person and being around each other while coding each class) has been effective in debugging and working on the synergy between each of the classes. In addition, communicating different concepts, ideas, and potential structures for each class and their associative methods has been a success. A potential open question that we have is how we may implement a server, which is a far sophisticated method of storing data, such that users can have their matches saved as well as having each user be able to interact with other users across different devices.