# MDD Project 1 Report

Team 1
Anthony DuPont, Clayton Marriott, Jordon Dornbos

July 31, 2014

# 1 Project Restated

## 1.1 Problem Definition

Youth soccer leagues can be difficult to organize when you have kids of varying skill, age, etc. The "Team Creator" builds teams according to specific criteria, and organizes the given data in an intuitive way.

### 1.1.1 Functionality Provided

1. Teams are created with skill level in mind. Players are ranked on a scale of 0-4.
2. Players are assigned a team based on age range, with the following ranges available: 8-9, 10-11, 12-13, 14-15, and 16-18.

### 1.1.2 Functionality Not Provided

1. The addresses of the players are not tracked.
2. Proximity of players' addresses does not influence team creation.

## 1.2 Model Creation

### 1.2.1 Signatures

The following signatures are defined in our model:

- Gender: each Person has a gender (MALE or FEMALE). Declared with *enum*.
- Person: the overarching person signature. Declared as *abstract* because Player is logical extension of it.
- Player: extends Person.
- MalePlayer: extends Player. A specific atom type for males.
- FemalePlayer: extends Player. A specific atom type for females.

- Team: contains sets of male and female players.
- League: a grouping of teams.

### 1.2.2 Relations between Signatures

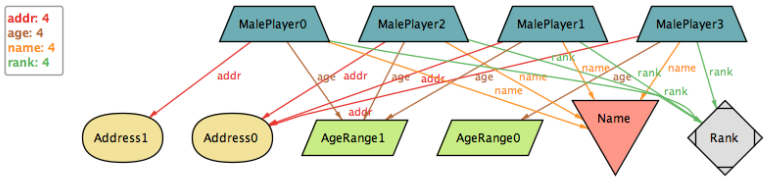
- Person: mapped to Gender.
- Team: mapped to a set of MalePlayer, and a set of FemalePlayer.
- League: mapped to a set of Team.

### 1.2.3 Multiplicities

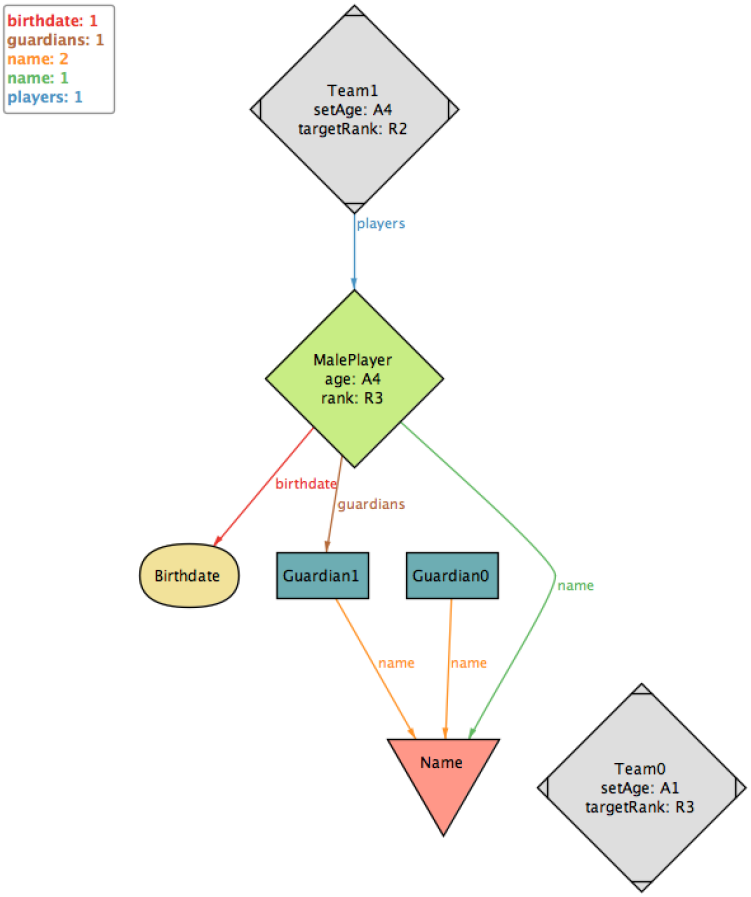
We have primarily used *one* in our relations because it only makes sense for those fields to be mapped to one object. For example, the gender field.

### 1.2.4 Visualization

The first design was fairly simple, but did not have very much in the way of controlling logic.




The second diagram shows the much-improved second version. There were still some issues with logic, but the resut is at the least much prettier.




The final version is not as pretty, but is logically easier to follow and has cut out much of the redundant information that really wasn't needed in the model.

## 1.3 Scope-Based Model Checking

### 1.3.1 Facts

1. *fact { all p: Player | one t : Team | p in getPlayers[t] }* = Players can only be part of one team. It is not an assertion because it must always be true; we do not want to check if it is true or false, because the way our model operates it is assumed to be always true.

2. *fact { all m: MalePlayer | isMale [m] }* = Male players are defined as male. This is not an assertion because this should always be true. Examples/counterexamples should not test this fact.

### 1.3.2 Assertions

1. *assert { all t: Team | one l: League | t in l.teams }* = All teams are in a league.

2. *assert { all t: Team | countMales [t] = countFemales [t] }* = Every team has an equal number of males and females.

#### 1.3.3 Functions

1. *fun countMales (t: Team) : Int* = Returns the number of male players on the given team.
2. *fun countFemales (t: Team) : Int* = Returns the number of female players on the given team.

#### 1.3.4 Predicates

1. *pred isMale (p: Player)* = Returns true if the given player is male. Used in the "all male players are male" fact.
2. *pred isFemale (p: Player)* = Returns true if the given player is female. Used in the "all female players are female" fact.

#### 1.3.5 Scope Check

Larger scopes like 32 and 40 work because there are 4 teams and need at least 4 boys and girls on each team. Scopes smaller than this do not because there are not enough kids to satisfy the requirements.