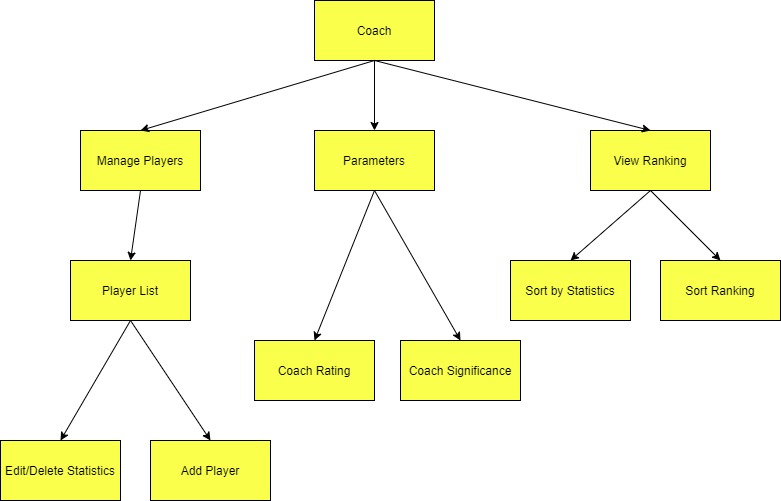
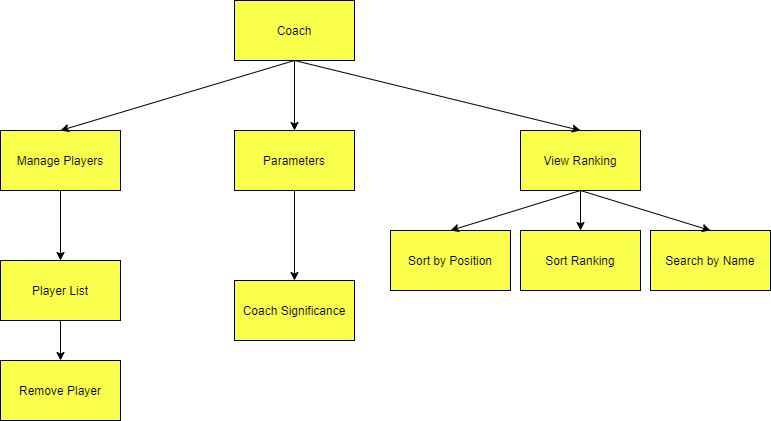
## Diagrams:

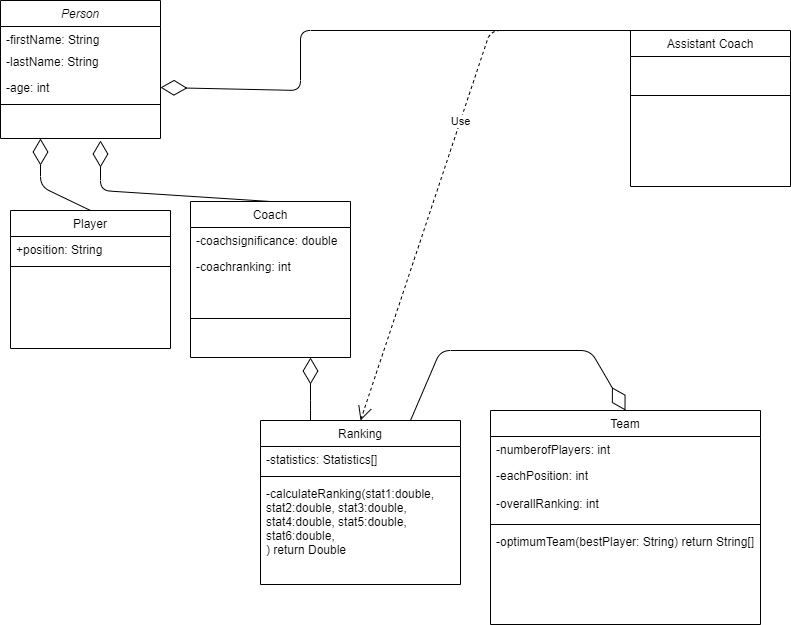
Decomposition Diagram - Pre Development



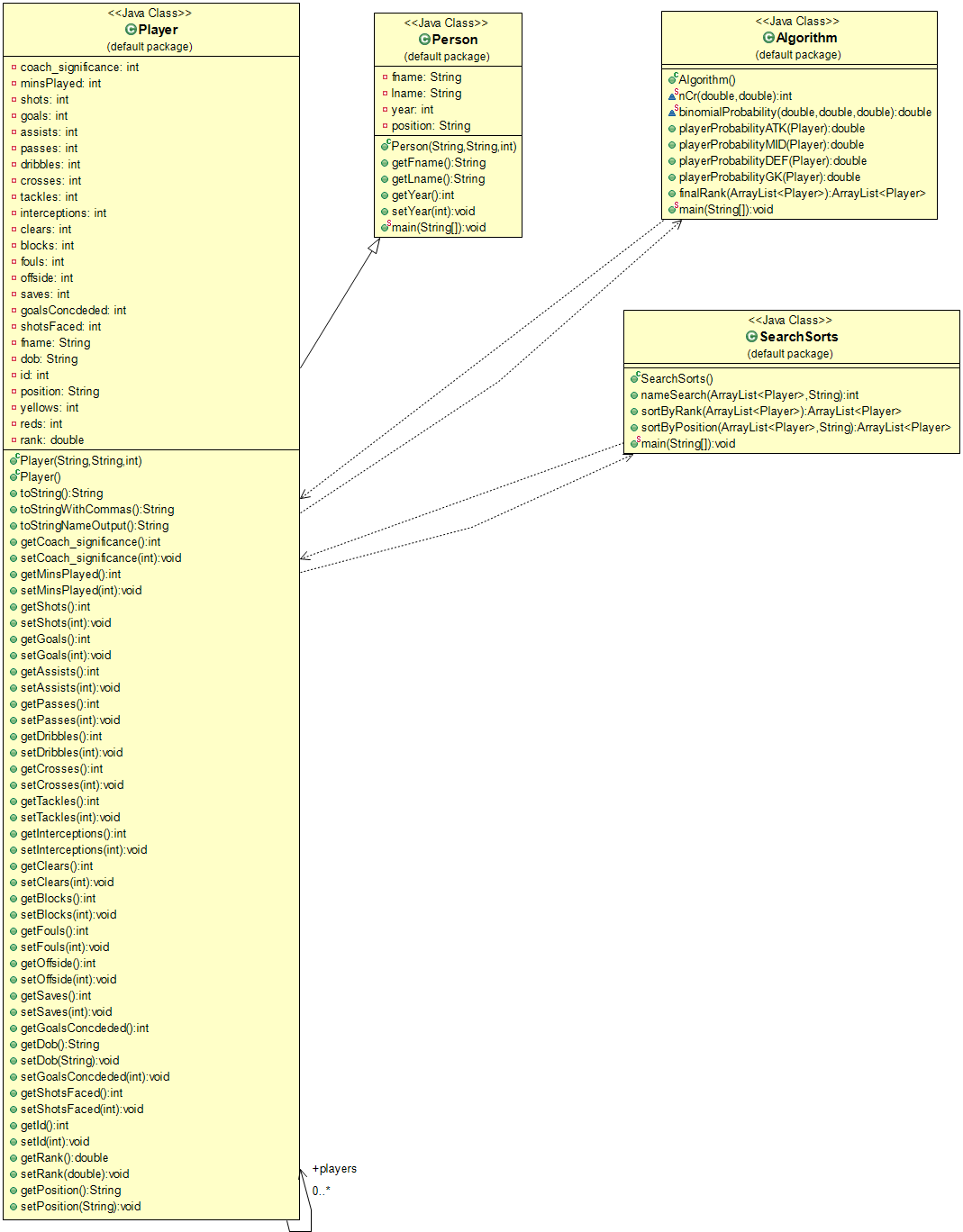
Decomposition Diagram - Post Development



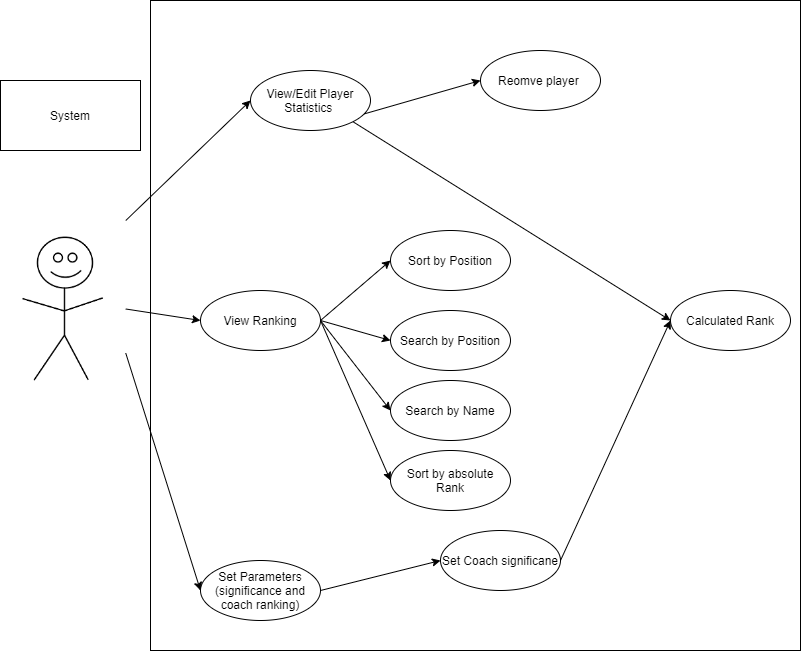
UML Class Diagram - Pre Development



Final UML Diagram with Functions - check marcos - shared folder



Use Case Diagram



Data Dictionary: Appendix of variables name, type, and a short description of their

intended use in the system. To improve technical documentation of the system

For better comprehension of the system.

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Modifier | Description |
| Player | | | |
| Fname (first name) | String | private | Each person has a first and last name. |
| dob | String | private | Each player is born after 2003 (under 19) |
| position | String | private | A player must have a position |
| rank | int | private | Each player has a calculated rank |
| Coach | | | |
| coachSignificance | int | private | Coach has an influence on the ranking algorithm which takes into account the 2 variables |
| Ranking | | | |
| statistics | arraylist [double] | private | Player statistics to be stored in an array list for further calculations |
| Team | | | |
| numberofPlayers | int | public | Each team must have a minimum number of players, with a specific number in each position |
| eachPosition | int | public |
| overallRanking | int | public | Each potential team with the ranking of the team combined |

## Algorithms

Sorting Algorithm: All the players are stored in an ArrayList with their relevant statistics. The players will be sorted by their calculated ranking (algorithm below). The ranks are doubles, and as there are a maximum of 100 players at any given time, a sectionsort is appropriate.

PLAYER = ArrayList of students in a class being sorted

NO\_ELEMENT = number of elements in array

SMALL\_ELEMENT = area for holding the smallest element found in that pass

NOW\_SMALLEST\_POS = the value of the current position in which to place the smallest element

1. function selectionSort(PLAYER): void

2. NOWSMALLESTPOS = 1

3. loop while NOW\_SMALLEST\_POS <= (NO\_ELEMENT – 1)

4. X = NOW\_SMALLEST\_POS

5. SMALL\_ELEMENT = PLAYER.get(X)

6. Y = X + 1

7. loop while Y <= NO\_ELEMENT

8. if PLAYER.get(Y) < SMALL\_ELEMENT then

9. X = Y

10. SMALL\_ELEMENT = PLAYER.get(Y)

11. end if

12. Y = Y + 1

13. end loop

14. PLAYER.set(X,PLAYER.get(NOW\_SMALLEST\_POS)

15. PLAYER.set(NOW\_SMALLEST\_POS, SMALL\_ELEMENT)

16. add 1 to NOW\_SMALLEST\_POS

17. end loop

18. end function

(Adapted from: Robertson, Lesley Anne.)

Searching Algorithm: To retrieve specific player information (individual statistics/rank etc.), the client can enter a name which will check the player ArrayList and if found, output the relevant information. As the ArrayList is sorted in descending order by rank, the ID numbers can not be used for a binary search, therefore a linear search is being used, which should still be efficient due to a maximum of 100 players to be searched at any given time.

1. function linear\_search (ArrayList, name)

2. for each item in the ArrayList

3. IF match item == name

4. return item location

5. end if

6. end for

7. end function

Ranking Algorithm

To form a ranking, a variety of statistics must be used in an attempt to accurately gauge player performance. This introduces an issue as the statistics are constantly changing with each game played and must be updated on the database frequently. Furthermore, the ranking must be relevant to specific positions, ranking players as a “one size fits all” is not only unfair but also inappropriate due to the nature of certain positions and their roles in the team.

As a result, the algorithm takes into consideration the player position, and weighs the relevant statistics differently to gauge their ability in their specific positions. For example, a striker has greater weightings towards shots and goals, whereas a defender has more influence on blocks and tackles. For further accuracy of the ranking system, the “expectedGoals, expectedTackles etc.” is calculated using a binomial distribution. This statistical analysis calculates the probability of a successful tackle, goal or any other key statistic for the position and influences the ranking along with the coach's significance. This is an important part of the algorithm as the probability can greatly influence the ranking and is a key factor to understand the player’s ability in their position.

Fig. 1: Flowchart of ranking process

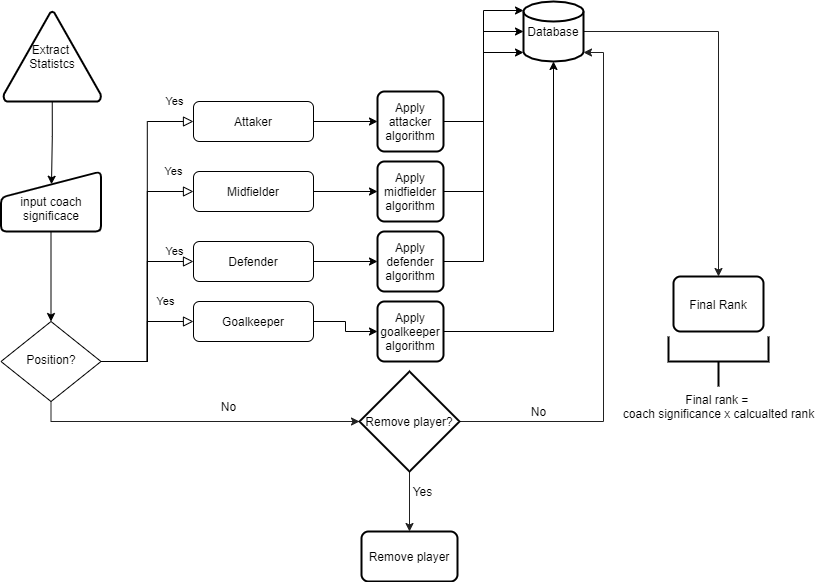
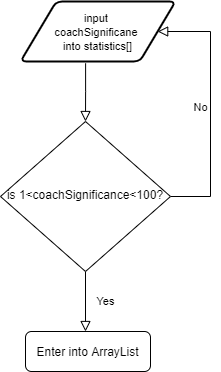


Fig. 2: Validation of coach input

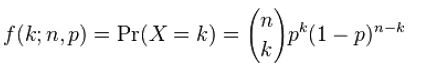


**Algorithm for ranking**

Application of the binomial distribution, as mentioned previously, this form of statistical analysis will be used to quantify key information of the player based on the relevant statistics. The general formula for the distribution is as follows;

Binomial Probability = Bi(P)

coachSignificance = cS



Where k, n, p changes dependant on the position of the player, e.g, if attacker, n = goals, k = shots, p = shots/goals, to calculate the probability of scoring at any given attempt. This is similar with the remaining positions where n, k, p change to relevant statistics for each position

|  |  |  |  |
| --- | --- | --- | --- |
| Position | n | k | p |
| Attacker | Goals | Shots | Goals ÷ Shots |
| Midfielder | Assists | Passes | Assists ÷ Passes |
| Defender | Goals conceded | Shots Faced | Goals conceded ÷ Shots Faced |
| Goalkeeper | Saves | Shots Faced | Shots Faced ÷ Goals Conceded |

Once this has been calculated, it is then multiplied by the weighted scoring:

Attackers:

Midfielders:

Defenders

Goalkeepers

The calculated ranks are then rounded to 2 decimal places as per client's request.

Test Plan

\*Everything apart from coachSignificance are internal validations

Class: Person

firstname, position: String

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Validation | Test Data Type | Input Data | Expected Output | Test Passed  Test Failed |
| Format Check  (alphabet only)  Length Check  (x>1) | Normal | “Bob” | Accept | ✔ |
| Abnormal | “xXBob420Xx” | Reject and input again | ✔ |
| Extreme | 222 | Reject and input again | ✔ |

Class: Coach

coachsignificance: int

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Validation | Test Data Type | Input Data | Expected Output | Test Passed  Test Failed |
| Range Check  (1<x<100) | Normal | 15 | Accept | ✔ |
| Abnormal | two | Reject and input again | ✔ |
| Extreme | 1.5 | Reject and input again | ✔ |

Class: Team

numberofPlayers, eachPosition: int

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Validation | Test Data Type | Input Data | Expected Output | Test Passed  Test Failed |
| Range Check  (0<x<24) | Normal | 15 | Accept | ✔ |
| Abnormal | two | Reject and input again | ✔ |
| Extreme | -3, 21 | Reject and input again | ✔ |

overallRanking: int

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Validation | Test Data Type | Input Data | Expected Output | Test Passed  Test Failed |
| Range Check  (x>0) | Normal | 180 | Accept | ✔ |
| Abnormal | sixteen | Reject and input again | ✔ |
| Extreme | -3, 3999 | Reject and input again | ✔ |

Class: Ranking

Statistics[]: double

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Validation | Test Data Type | Input Data | Expected Output | Test Passed  Test Failed |
| Presence Check | Normal | Statistics != null | Accept | ✔ |
| Abnormal | null | Reject and input again or option do delete player | ✔ |
| Extreme | null | Reject and input again or option do delete player | ✔ |