Use Case ID: RT001

Description: User Adds Player to Team

Dependencies: New Player Use Case

Precondition(s): User has a new player. User wants to add a new player to a team. User is on the "Add New Player" menu.

Primary Actor: User

Action:

1. User selects the "Add Player" option from the menu.
2. User enters the name of the player.
3. System presents a list of player types.
4. User selects the desired player type for the player.
5. System presents a list of teams.
6. User selects the team for the player.
7. New player is successfully added.
8. System displays a confirmation message.

Postconditions: New player is added to the team, and the system displays an approval message.

Alternative Paths:

Error:

a. User enters an integer value other than the player's name.

b. System displays an error message.

c. User cancels the process.

Use Case ID: RT002

Description: User Sorting List of Players

Dependencies: None

Precondition(s): User wants to sort the list of players. The user is on the main menu.

Primary Actor: User

Action:

1. User selects "Sort" from the menu.
2. System displays the current list of players.
3. User chooses a sorting criterion (e.g., by name, by team, by coach).
4. System sorts the list based on the selected criterion.
5. System displays the sorted list to the user.

Postconditions: The list of players is sorted according to the selected criterion, and the sorted list is displayed.

Alternative Paths:

Error:

a. User selects an invalid sorting criterion.

b. System displays an error message indicating invalid input.

c. User cancels the sorting process, and the list remains unchanged.

Use Case ID: RT003

Description: Creating Random Players with Teams and Coach Types

Dependencies: None

Precondition(s): The user wants to create teams and random players with different coach kinds. User on the Generate Random Players screen.

Primary Actor: User

Action:

1. The user picks "Generate Random Players" from the menu.
2. The system prompts the user to specify how many random participants to produce.
3. The user enters the appropriate number of random participants.
4. The system produces random players with unique names, coach kinds, and teams that do not contradict with the current data.
5. The system adds randomly created players to the list of participants.
6. The system displays a list of all participants, including freshly created ones, at the terminal.

Postconditions:Random players with different coach kinds and teams are successfully produced and added to the list. The terminal displays a list of all players, including freshly produced ones.

Alternative Paths:

Error:

a. The user enters an incorrect number of random participants.

b. The system generates an error message indicating incorrect input.

c. The user cancels the generating process, and no modifications are made to the list of players.

Use Case ID: RT004

Description: Sorting a Dummy List of People

Dependencies: The Club\_Form.txt file contains a list of random names.

Precondition(s): The user wishes to sort the list of names collected from the Club\_Form.txt file. The user is seeing the main menu.

Primary Actor: User

Action:

1. The user picks "Sort List of People" from the menu.
2. The system retrieves the list of names from the Club\_Form.txt file.
3. The system starts the recursive sorting process, which sorts the list of names in alphabetical order.
4. Once sorted, the system shows the first 20 entries of the sorted list on the screen.

Postconditions: The Club\_Form.txt file's list of names is successfully sorted in alphabetical order using a recursive sorting technique, and the first 20 entries are shown on the screen.

Alternative Paths:

Error:

a. If the Club\_Form.txt file cannot be located or is empty, the system will show an error message stating that the file is empty or missing.

b. If there is a problem reading or accessing the file, the system will display an error message.

c. If a mistake occurs during the sorting process, the system shows an error message stating the problem with sorting.

d. If the sorted list has fewer than 20 records, the system will display a notice stating that there are fewer than 20 records available.

Use Case ID: RT005

Description: Deleting a Player from a Team

Dependencies: None

Precondition(s): User wants to remove a Player from a team. The user is on the Delete menu.

Primary Actor: User

Action:

1. User selects "Delete Player from Team" from the menu.
2. User Search what Player wants to delete.
3. User selects what player wants to remove.
4. System display confirmation message
5. User confirms the deletion.
6. System removes the selected player from their team.
7. System updates the list of players to reflect the deletion.

Postconditions: The selected individual is successfully removed from their team, and the player list is updated appropriately.

Alternative Paths:

Error:

a. If the user cancels the deletion procedure, nothing changes in the list of players.

b. If there is an issue accessing or updating the player list, the system will display an error message.

**Why Activity Diagram and Class Diagram?**

I developed an Activity diagram because I want to show how a system dynamically looks and how system process flow and control flow works. Inter-process flow in rugby application, This diagram was chosen because it is sequential, branching, or simultaneous. The missing parts of activity diagrams compared to Sequence diagrams are messaging. We don’t need the messaging part because our system works like a decision point and action point. System has a login part for the federation if a login successful federation member has to make a decision. Federation members can add players, remove players , update play etc. Also the system is only a console application it shows us error or successful messaging. Another reason is that even if someone does not know the software system, everyone can easily understand our system workflow. My main target is easy to understand. Activity diagrams are generally easier to understand for stakeholders who may not be familiar with technical details or system internals.

**Activity Diagram**: Activity diagrams show the order or flow of operations in a system. They are similar to flowcharts, and show the outcomes of each action, and actions that happen after specific responses. While they can be either concurrent or sequential, all activity diagrams have a beginning and an end state. Widely used in business process modeling and algorithm design.

**Compare with Sequence Diagram**:

I'm comparing Sequence Diagrams because Sequence diagrams are more detailed and technical. Sequence Diagrams delve into the technical intricacies of the system, illustrating the precise sequencing of operations and communication patterns between objects or components. Sequence diagram mostly used for how systems show end use cases, user and system interactions, and system to system interactions. Commonly used for system design, software architecture and scenario analysis, this means if someone doesn’t know anything about a diagram they are never able to understand at first glance. And Sequence Diagram when you want to depict interactions and message flows between different objects or components in a system. **Activity Diagram is Well-suited for business process modeling, workflow modeling, and algorithm design. It was used in our system because we needed an algorithm modeling for the rugby system.**  Sequence diagrams are classified as interaction diagrams, and they visualize how objects collaborate, and focus on the order or time in which they happen. They show end use cases, user and system interactions, and system to system interactions.

I developed a Class diagram because Java is an object oriented programming language. Class diagrams are designed based on OOP (Object Oriented Programming). The goal is to define the classes and the relationships between them in our software. For example Our system has a Player model and a Team model. We want to show some relation between the two model and a class diagram is a good option for us. If we want to extend our project or improve , class diagrams give accessibility to extend.

**Compare with Entity-Relationship Diagram:**

First reason is we don’t have a database for our system. Second reason is that Java is an object programming language. Class Diagrams may be more suitable for representing the system's static structure, including classes, attributes, methods, and relationships. ERD Diagram represents entities, attributes, and relationships in database design and data modeling. For a higher-level view of the system's design and architecture, a Class Diagram may be more appropriate. And for future Class diagrams are more extendable instead of ERD Diagram. **Class Diagram Extensible to incorporate software design patterns, interfaces, and architectural concepts. ERD Diagram Less extensible for representing non-database-related software design patterns.**

**Scenario, User Test 1**: Rugby Administrator log into the system to Add Player to Team

**Description**: As a Team administrator, I want to log into the system so I can add a new player to the team and that new player can join the team and play in a match.

**Acceptance Criteria**:

1. The administrator should be able to enter their credentials through a login interface provided by the system.
2. The system should verify the administrator's identity and enable access to administrative features after a successful login.
3. The system should indicate an error and ask the administrator to try again if the login credentials are not correct.
4. The administrator should have access to the feature of adding a player to a team after logging in.
5. To maintain security, the administrator should be automatically logged out of the system after a certain amount of inactivity.

**Acceptance Task:**

1. Make sure the administrator accesses the system and sees the login interface.
2. Check to see if the system uses the supplied credentials to authenticate the administrator's identity.
3. Check whether the feature that indicates an error when login credentials are entered incorrectly works.
4. Verify that, following a successful login, the administrator has access to the feature that allows them to add players to teams.
5. Verify that the administrator is automatically logged off of the system after a certain amount of inactivity.

**Unit Test**:

1. Verify that the administrator's identity can be verified by the Login class using the supplied credentials.
2. Check to see that the system appropriately reports an error for invalid login information.
3. Verify whether access to administrative features can be granted following a successful login.
4. If the administrator has been logged in for a long time, log out the administrator account.
5. Verify that the system can correctly add new players to the team.

**Scenario, User Test 2**: Search Player in League.

**Description**: As a User, I want to search for a Player.

**Acceptance Criteria**:

1. Users should be able to look for players using the search feature built into the system.
2. It should be possible to search players by name or relevant criteria.
3. The system should show the user the result found.
4. Search results should be related to what the user searched.

**Acceptance Task:**

1. Verify that users are able to type the name of the player or other pertinent parameters into the search field.
2. We must verify that the system returns the correct player searched.
3. The user must ensure that can access the details of the player's result.

**Unit Test**:

1. Verify that the player data can be retrieved from the Search menu using the specified search parameters.
2. Check to see if the system is correctly using the entered search criteria to filter players.

**Scenario, User Test 3**: Update Player Information.

**Description**: As a federation member, I want to update Player information.

**Acceptance Criteria**:

1. The federation member ought to be able to look up a player by name using the system.
2. The federation member should be able to edit Player information.
3. After editing player information our database(.txt) has to be updated and display member player information updated.
4. The system display Successful message after the update is completed.

**Acceptance Task:**

1. Make sure the player search feature is accessible to the federation member.
2. Check that the system shows the right player information for the searched player.
3. Check that the administrator may alter the player's information and save the changes.
4. Display confirmation message after updating.
5. Check txt document after the update is completed.

**Unit Test**:

1. Determine whether the Player data can be updated with new information.
2. Check that the update procedure properly alters the player's information in the database.
3. Testing the validation logic will guarantee that only valid updates are allowed.

**Scenario, User Test 4**: Generate Random Player

**Description**: I want to create random player profiles as a rugby club administrator, assigning them to teams and coach types in order to simulate player registration and load basic data into the system.

**Acceptance Criteria**:

1. It should be possible for the system to produce a predetermined amount of randomly generated player profiles.
2. Random information like name, age, position, and skill level should be included in every player profile that is generated.
3. Every player profile should have a random coach type assigned by the system, selected from a predetermined list of coach kinds.
4. The system ought to show a confirmation message after generation, signifying that the random player profiles were successfully created.

**Acceptance Task:**

1. Check to see if the system offers the ability to create player profiles at random.
2. Verify that every player profile that is generated has arbitrary information such name, age, position, and skill level.
3. Verify that a random coach type is allocated to each player profile from the predetermined list.
4. Verify that teams are formed at random from the available teams in the system by assigning participants to them.
5. Make sure that when the random player profiles are successfully generated, a confirmation message appears.

**Unit Test**:

1. Generate a random player profile and verify that the assigned team is one of the available teams in the system.
2. Generate random player profiles and verify that a confirmation message is displayed indicating the successful creation of profiles.

**Scenario, User Test 5**: Remove Player From Team

**Description**: As a Club administrator, I want to remove a player from the team.

**Acceptance Criteria**:

1. Users should be able to remove a player from a team via the system.
2. The system should provide functionality for users to remove a player from a team.
3. Users should be able to select the player they want to remove from the team.
4. After confirmation, the system should remove the selected player from the team and update the team's roster.
5. Upon successful removal, the system should display a confirmation message to the user.

**Acceptance Task:**

1. Check to see if the system has the ability to remove a player from a team.
2. Test that users can select the team from which they want to remove the player.
3. Check that users are presented with a list of players in the selected team.
4. Check if it is possible to update the team roster and remove the chosen member from the squad.
5. Confirm that a confirmation message is displayed upon successful removal of the player from the team.

**Unit Test**:

1. Simulate the removal of a player from a team and verify that no exceptions are thrown.
2. Verify that a confirmation message is displayed to the user after removing the player from the team.

I've sorted the steps that the user can do with this diagram. I've determined which steps the user will follow. I've determined what steps to go back to if the user makes an error in the data entry, or what errors the system should make. I made a draft on the system requirements and how the project would proceed. First of all, I did research on the algorithm of the system and its working logic. Then I set the paths with a diagram. I investigated the errors in the algorithm and the solutions. When the algorithm didn't have any question marks, I moved to the encoding section and started the project.