Connor McGuire

Solution Planning

Solution planning

Table of Contents

[Analysis and Design 2](#_Toc41232123)

[Business model 2](#_Toc41232124)

[CRC cards 2](#_Toc41232125)

[Top Level Use Case 4](#_Toc41232126)

[Use case Scenario 5](#_Toc41232127)

[Use case scenario 1 5](#_Toc41232128)

[Use case scenario 2 6](#_Toc41232129)

[Activity Models 7](#_Toc41232130)

[Add Profile diagram 7](#_Toc41232131)

[Add Player Diagram 8](#_Toc41232132)

[Static model 9](#_Toc41232133)

[Sequence Diagram for List Profiles 11](#_Toc41232134)

[Sequence Diagram for Delete Profile 11](#_Toc41232135)

[View Model 12](#_Toc41232136)

[Top Level Use Case 13](#_Toc41232137)

[User Interface Design 13](#_Toc41232138)

[WPF Pages 14](#_Toc41232139)

[Log in Page 14](#_Toc41232140)

[Mainframe 15](#_Toc41232141)

[Players page 16](#_Toc41232142)

[Profile Page 17](#_Toc41232143)

[Bibliography 18](#_Toc41232144)

# Analysis and Design

# Business model

After having my second meeting with the client of simply football, we decided on developing certain parts of the system due to time constraints and considering the most important features of the system to be included. This would end up being creating two types of people that can be added to the club, players and youth players. Alongside this, the system would also have to record player profiles and record them by season.

To do this, I had to look over the use case diagrams and the class diagram I had created in the inception planning and mend them to the new required specification.

Firstly, I had created CRC cards so I could have a better understanding of what the different classes were and how they interacted with each other.

## CRC cards

|  |  |
| --- | --- |
| Class name: Person  Superclass  Subclasses: Player, Youth | |
| Responsibilities:  Knows the name and address of players | Collaborator(s):  Address  MainFootball |

|  |  |
| --- | --- |
| Class name: Player  Superclass: Person  Subclasses: | |
| Responsibilities:  Knows player forms info | Collaborator(s):  Address  MainFootball |

|  |  |
| --- | --- |
| Class name: Youth  Superclass: Person  Subclasses: | |
| Responsibilities:  Knows Youth forms info | Collaborator(s):  Address  MainFootball |

|  |  |
| --- | --- |
| Class name: Address | |
| Responsibilities:  Knows the address of players and youth | Collaborator(s):  Person |

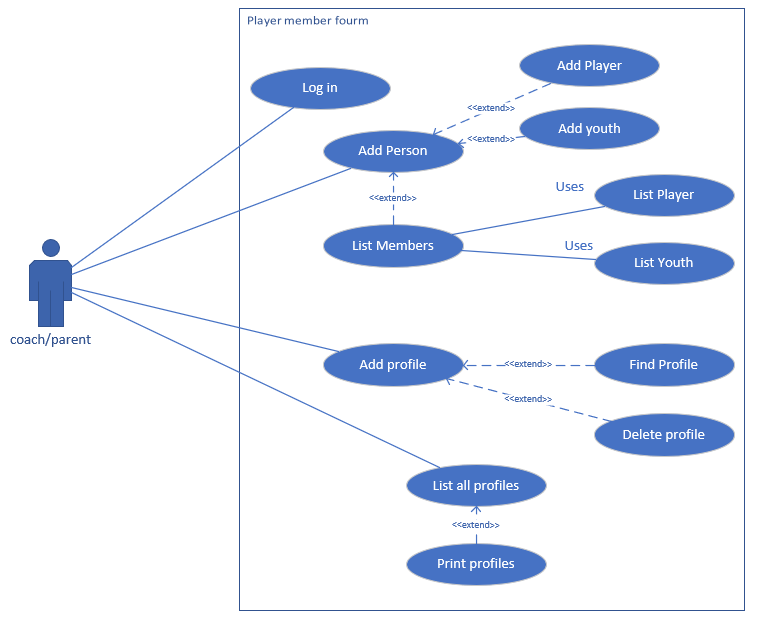
|  |  |
| --- | --- |
| Class name: Profiles | |
| Responsibilities:  Knows player Profile information | Collaborator(s):  SkillSet  MainFootball |

|  |  |
| --- | --- |
| Class name: SkillSet | |
| Responsibilities:  Knows players skills levels | Collaborator(s):  Profiles |

|  |  |
| --- | --- |
| Class name: MainFootball | |
| Responsibilities:  Adding players and youth players  Knows list people  Knows list profiles  To print profiles to file  Knows to delete a profile | Collaborator(s):  Person  Profile |

After doing this, I had created this top-level use case diagram for the business model.

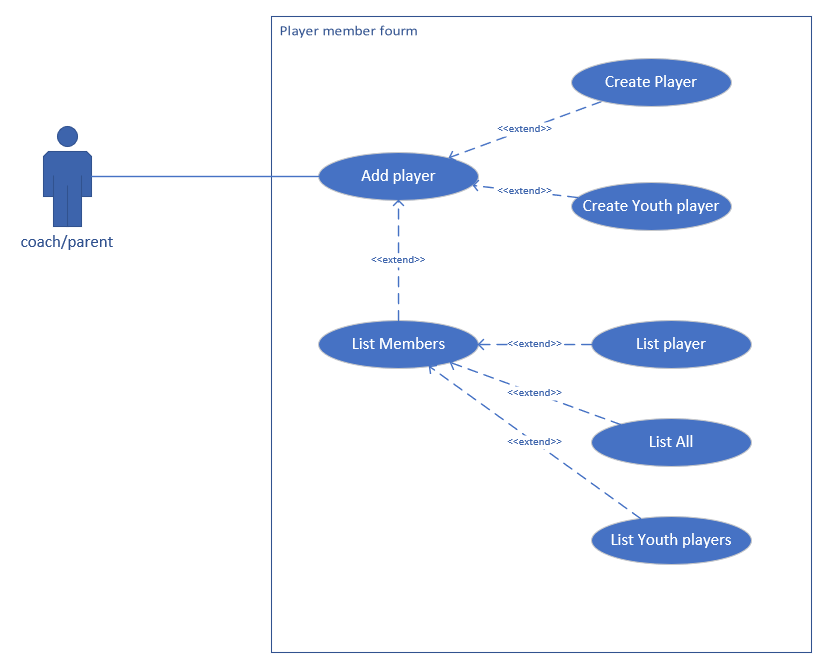
## Top Level Use Case



This shows the basic functionality for the coach/parent actor and what they would be allowed to access throughout the application.

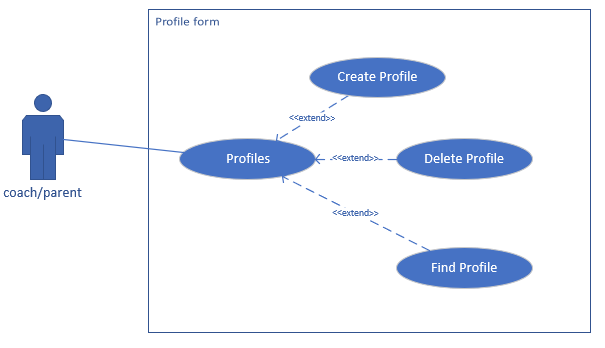
I had also made another couple of use case scenario and activity diagrams for the system.

## Use case Scenario



### Use case scenario 1

|  |
| --- |
| Use case: Coach adds a new player |
| Trigger event: New football player wants to join |
| Pre Condition: Player doesn’t already play for a football team |
| Post-Condition: Player is added to the team |
| Scenario: normal flow:   1. Coach requests to add player. 2. Coach chooses the “Player” option 3. Coach inserts players details into system. 4. New player is added to the team. |
| Alternative flow: Player is a youth player.  At step 2, coach chooses the “youth” option for the player.  Returns to step 3. |

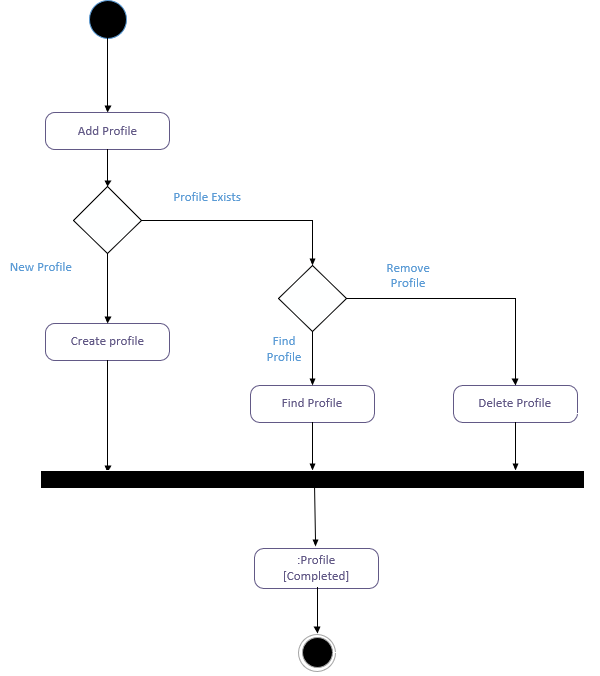


### Use case scenario 2

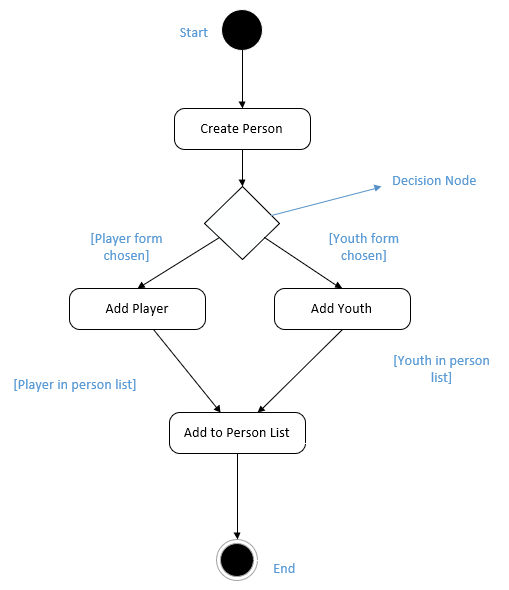
|  |
| --- |
| Use case: Coach adds a new profile |
| Trigger event: New profile for footballer |
| Pre Condition: There’s isn’t a profile for that player in that specific season |
| Post-Condition: Profile is added to the list |
| Scenario: normal flow:   1. Coach chooses profile. 2. Coach requests the “create profile”. 3. Coach inserts players profile details into system. 4. New profile is added to the list. |
| Alternative flow: Profile already exists  At step 3, profile already exists  Step 4 Return message saying profile exists and cannot add new profile. |

## Activity Models

### Add Profile diagram

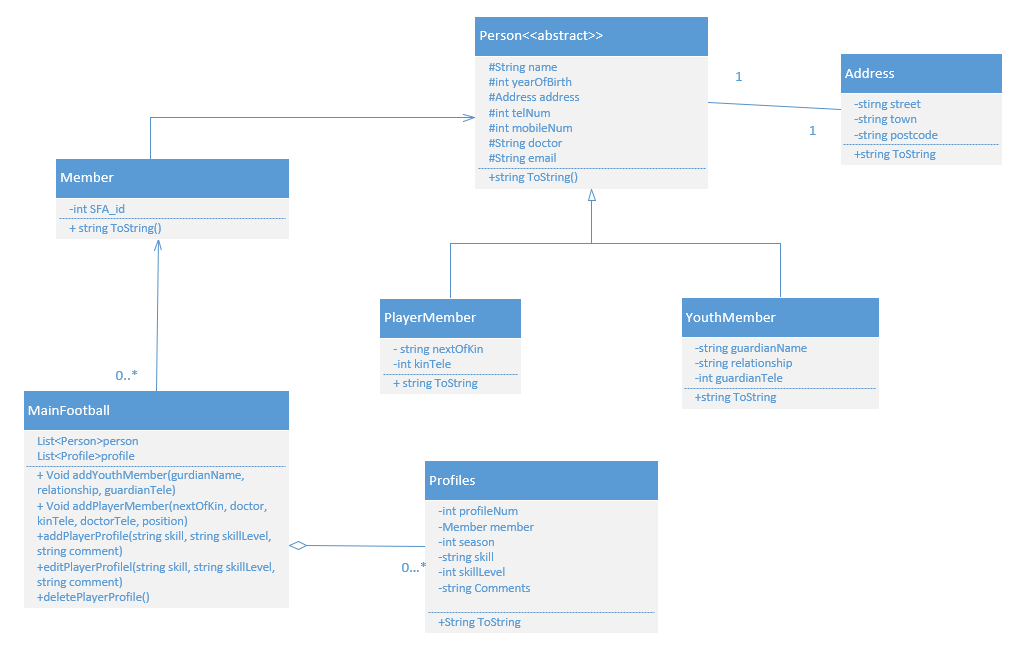
****

### Add Player Diagram

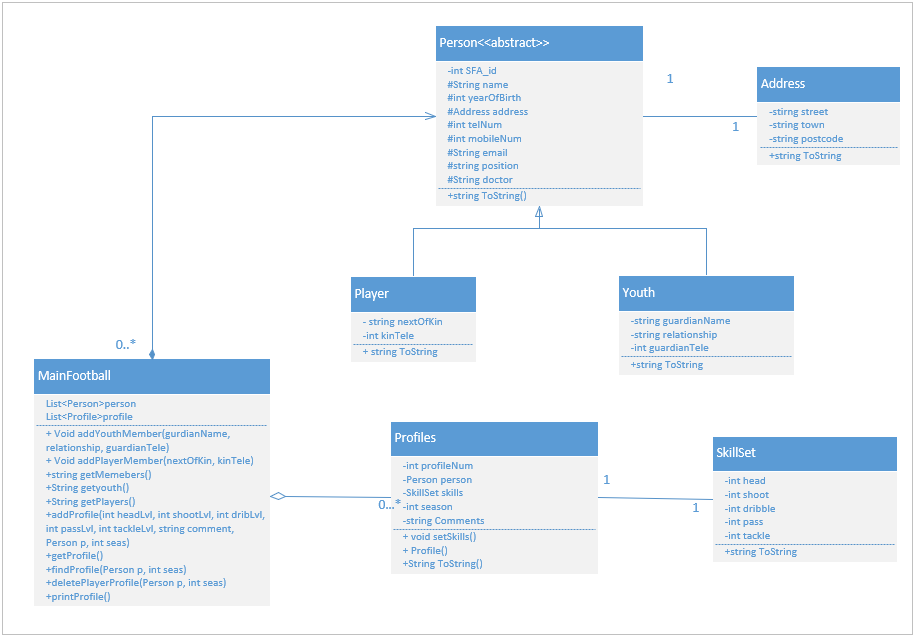
****

## Static model

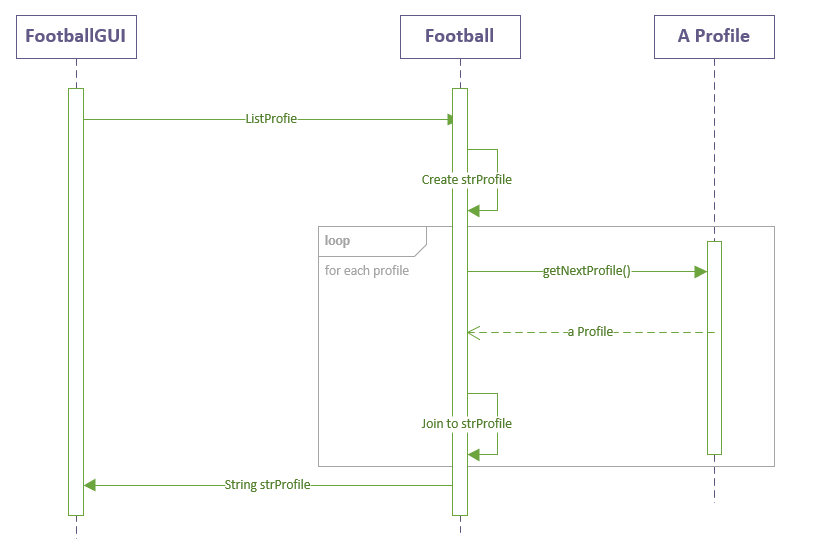
Upon designing this application, I came up with a class diagram that I thought would be the best way to tackle this system which looked like this.



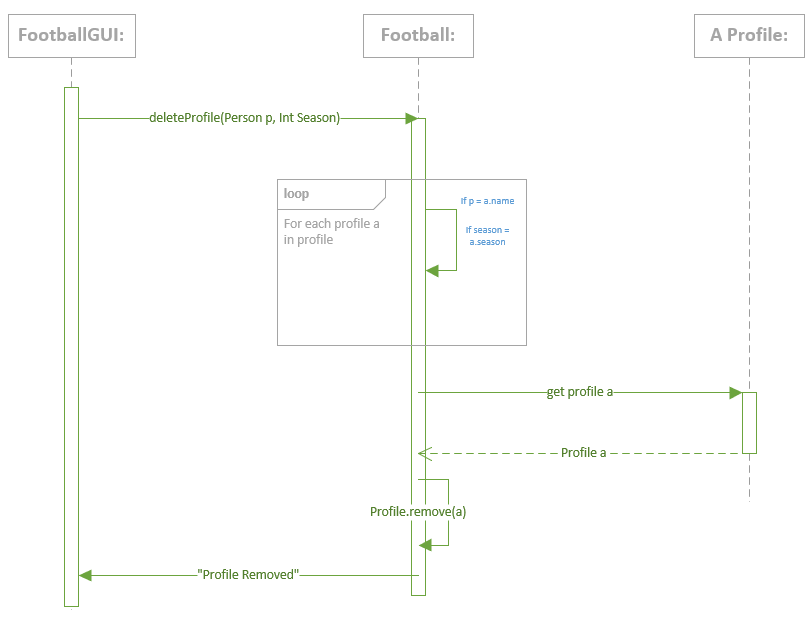
Although, upon coding this project, I realised this class diagram wasn’t good enough for this particular system. Realising this, I decided to modify it. One major change I made was to include a “SkillSet” class, which allowed me to hold many skills for the player profile since the previous diagram only allowed me to hold one skill at a time, along with the removal of the “Member” class and adding the “SFA\_id” into the “Person” class. The updated class diagram is as followed.



## Sequence Diagram for List Profiles



## Sequence Diagram for Delete Profile

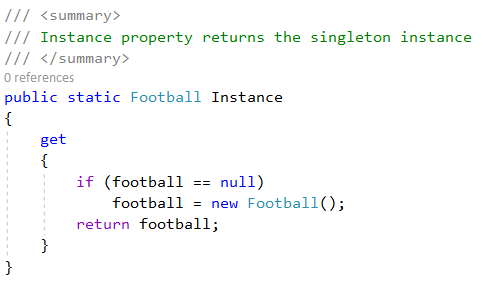


# View Model

When considering the user interface design for this application, I had to choose which front end technology I was going to be using, along with different analysis on the human computer interaction to ensure that the user of the system would have the easiest time navigating the system.

Firstly, since I had created the back end of this program in visual studios, using the c# programming language, it would make sense to use the visual studios WPF application software to create the front end. This would ensure the most fluent one-way data binding between the business and view models.

To create the binding between the WPF files and console application, I will be using a singleton orchestrating model to restrict a single instance of the football class. This will also be helpful since it can be accessed anywhere on the system to be used.

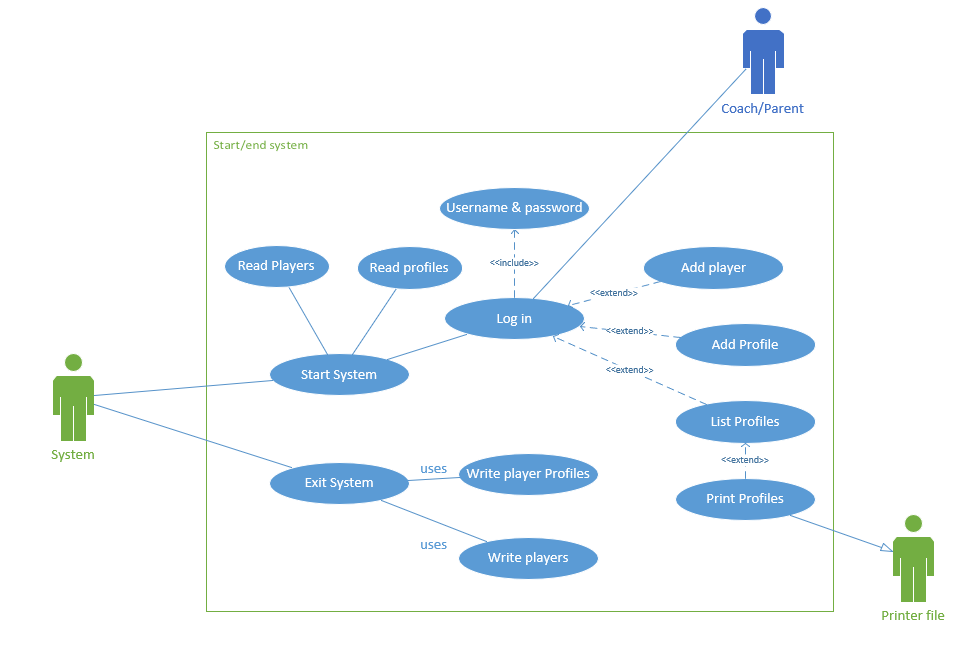


**“Ensuring system functionality and usability, providing effective user interaction support, and enhancing a pleasant user experience.”** (W3 Computing, n.d.)

When analysing the human computer interaction, I had to look at the ways in which the system could be made as simple as possible, so that the user could have the best possible experience navigating the application. Since this application would be used by a coach or a parent, they may not be very tech savvy, so when building the WPF pages, they would have to seamlessly interlink with each other to fulfil the users needs, in which ill explain later when showing the WPF I had made.

To design the WPF pages, I had to create a top-level use case for the system to understand how the different pages would connect with each other.

## Top Level Use Case



## User Interface Design

When creating these WPF below, I had implemented a few design principles to ensure I could make the best front end for this application as I could. One principle I used was to have good flow throughout the program, this was so that the navigation throughout the system was seamless and was easy too pick up on the first exposure to the programme.

This also leads into another principle I used which was DRY (Don’t Repeat Yourself), this is to certify that the system wasn’t overly complicated and to not have multiple connections to the same code where it wasn’t needed to maintain usability and maintainability. (Chauhan, 2019)

## WPF Pages

Now due to the current corona virus pandemic and the time constraints now put on this project, I didn’t have enough time to code and connect the WPF to the back end programme but I will showcase what I would’ve liked each page to be connected too.

### Log in Page

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **App Name**  **Simply Football** | **Designer**  **Connor McGuire** | **Page**  **1** | | **Version**  **1** |
| **Overview:**  Log in that requires a username and password. | | | | |
| **Screen orientation – Portrait** | | | **Interactions:**  **Username box:** To insert users’ username.  **Password:** To take in users’ password.  **Log in button:** If accepted, will take user into app. MainFootball() | |

### Mainframe

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **App Name**  **Simply Football** | **Designer**  **Connor McGuire** | **Page**  **2** | | **Version**  **1** |
| **Overview:**  Mainframe page with 4 buttons to navigate the app. | | | | |
| **Screen orientation – Landscape** | | | **Interactions:**  **Add Footballer button:** Takes to the add footballer page to add players.  **Add footballer profile:**  Will take user to a page to ad profiles.  **List all football profiles:**  Will list all profiles in the text box provided. getProfile()  **Print footballer profiles:**  Will print profiles to a file and display in the text box all profiles. printProfile()  **Text box:** Will display profiles for list all and print. | |

### Players page

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **App Name**  **Simply Football** | **Designer**  **Connor McGuire** | **Page**  **3** | | **Version**  **1** |
| **Overview:**  Add person page to add player and youth, also lists players. | | | | |
| **Screen orientation – Landscape** | | | **Interactions:**  **Add buttons:** Adds player/youth to person list. addPlayers ()  **Cancel buttons:** Cancels back to the mainframe.  **Info boxes:** To be filled out for the persons information.  **List Players:** Lists the “Players” in the persons list. getPlayers()  **List all:** lists all people in the club. getMembers()  **List youth:** Lists all the “youth” players. getYouth() | |

### Profile Page

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **App Name**  **Simply Football** | **Designer**  **Connor McGuire** | **Page**  **4** | | **Version**  **1** |
| **Overview:**  To add a profile for a player’s season | | | | |
| **Screen orientation – Portrait** | | | **Interactions:**  **Players name box:** Selects a certain player from the person list.  **Season box:** Selects from a number of seasons.  **Find profile:** Finds a profile from the players name and season. getProfile()  **Delete profile:** Deletes a profile from name and season.deleteProfile()  **Cancel:** Cancels back to mainframe.  **Save profile:** Save the player profile to the players name using skills and season.  Skills sliders: Used to determine the skill level of a particular skill from 1-5. addProfile()  **Comments box**: to add a comment about the profile  **Text box:** prints out profiles. | |

# Bibliography

(Chauhan, 2019) - https://www.w3computing.com/systemsanalysis/understanding-human-computer-interaction/

(W3 Computing, n.d.) - https://www.w3computing.com/systemsanalysis/understanding-human-computer-interaction/