Run With Friends

H446 - Programming Project
Joshua Graham
Candidate number: 8121
JFS: 10326

3.1

Analysis

3.1 Analysis of the problem (10 marks)

3.1.1 Problem identification

A. Describe and justify the features that make the problem solvable by computational methods.
B. Explain why the problem is amenable to a computational approach.

- Due to the current outbreak of Corona Virus, technology is being used more than ever before to connect friends and families from across the world. And also obviously due to isolation, me and my brothers were very bored not being able to see our friends. My brother, who is a regular runner, said to me that his runs have not been the same lately, as he doesn't have a companion to run with and motivate him.

- He thinks it would be great if he was able to connect online with a friend, whilst at the same time running with them. Therefore, my client (my brother) would like me to create a program, which would allow him to run with his friends.

- We first decided that it must be in the form of a mobile application, due to a phones mobility, it will allow the program to run in the background, whilst the user is running in real life - the user and the program can run simultaneously ;)

- Upon sitting down with my brother - we decided on the following features of the app:

  • Firstly, it must allow users to connect online to join a run together.

  • Users should be able to view their partners distance, and speed during the run, so that it feels like they are running together.

  • Get valuable feedback after a run, such as speed, total distance and calories burnt vs your partners data.

  • My client very interestingly insisted on the user choosing how long the run will be (e.g. 5km) instead of the race finishing when the user wants. This will give the user motivation to finish the initial set distance of run.

  • Finally, users should be able to share their achievements and runs with their friends, as according to my brother, this is what also motivates him to run more.

Why it is suited to a computational solution

- This problem is clearly suitable for a computer to solve. This is because, as of 30th of March 2020, you cannot meet friends in public, therefore you cannot go

on a jog with a friend. However, by connecting through an app, two or more people can run together, whilst viewing each others progress in the run, and after receive valuable and personal running data such as speed, distance and calories burnt. This is only possible through connecting together through computer networks, GPS and realtime databases.

- Here is a table showing the problems of going for a run with friends, and how a computer can solve these problems:

Problem:	**Solution:**
Physically going for a run with a friend.	Connect online through the use of realtime databases. You can see your friends live running data in realtime, so it is as if they were there with you.
The freedom of mobility whilst tracking all this data	The user cannot carry their laptop or desktop computer with them on a run. Using a mobile phone running a mobile application allows the user to run, whilst the program is running in the background.
The user analysing their running data.	Using GPS, their running data such as speed, distance and altitude can be observed. Furthermore, their calories burnt can be deduced.
Accounts for each user, so that data can be saved.	Use a database. I have decided to use Google Firebase as it can be integrated nicely with Swift 5 and Xcode IDLE. Users can register and then login on future visits to the app.
The ability to share their achievements and runs with their friends.	An app allows sharing through social medial, allowing the user to show off their hard work.
GUI	Using Xcode to create the GUI of the app (the user interface). The unique user interface is obviously what makes a mobile app an app.

Computational methods the solution lends itself to

Top-down design

- Overall, I will be using a top-down design to solve my problem as it involves breaking down the task into smaller tasks, which are broken down again and again, until each one is a small and manageable task. This involves problem recognition and problem decomposition:

Problem Recognition

- The overall problem is simultaneously updating the location of both users, and displaying this information on both of the users phone.

Problem Decomposition

- This problem can be broken down and represented by these steps:

    1) Connecting two users to a private 1v1 race where a custom distance is chosen (e.g. 100 metres)

    2) When the race starts, instantly start updating the location of both users using GPS

    3) Calculating the distance that each user has ran and writing this to a database

    4) Reading each users distance, and checking if either user has finished the race

    5) When both users have finished the race, display running data such as speed, and time.

- Each of these steps can be solved on their own and then combined into one modular program - divide and conquer.

3.1.2 Stakeholders

A. Identify and describe those who will have an interest in the solution explaining how the solution is appropriate to their needs (this may be named individuals, groups or persona that describes the target end user).

The main stakeholder: My brother - Aron

My main stakeholder is my brother Aron. As mentioned in the previous section (3.1.1 Problem identification), he is a passionate runner, however due to the corona virus outbreak, he can no longer run with his friends and he has lost his motivation and joy for running.

By having an app which will allow him to connect and run with his friends in realtime, will bring back his motivation to get outside and run.

My client has an iPhone 7, therefore the user interface will be created with this specific screen size in mind. Also, I also have the same phone so this is useful because I can use my phone for testing, instead of bugging my brother for the use of his phone.

Second stakeholder: My friend - David

My app will also target casual runners (the general population), in addition to passionate runners. This is as my app will hopefully motivate people to get outdoors and exercise together. Therefore, my stakeholder for the general healthy population will be my friend David, who is enjoys playing sports from time to time.

In the hopeful future, if I deploy my app onto the App Store, the public will also be a major stakeholder. They will be able to create their own account and then run with their own individual friends. They will also have a say in how the app progresses and is updated over time as any requests they have would be taken into consideration.

The target age group is for people all ages (given that they can run). Therefore, the design must be quite modern, consistent with clarity and aesthetic integrity (represents how well an app's appearance and behaviour integrate with its function), so that it is appealing to people of all ages.

It is appropriate that each user can make their own account. This is because security-wise, it is crucial that users can only run with their friends, as you do not want random people accessing your locating and personal data.

First interviews with my stakeholders

I have set the following questions to ask each of my two stakeholders. The questions will find out their opinions on the app, it's design and how they would like to use it. I have added a specific question which I will ask, in order to understand how the app will support the needs of my two different stakeholders. Here are my questions:

1) How often do you go running?
2) Do you use a running app?

    IF yes:

        3) What app is it?
        4) What do you like about that app?
        5) Are there any problems with that app?

    IF no:

        6) Have you ever thought about using a running app?

7) *Explain app idea*. So what do you think about running in realtime with a friend online?

8) What features would you like the app to have?

9) Specific question:

   - For my brother (passionate runner): What features would you like the app to have so that it suits your needs as a frequent runner?
   - For David (casual runner): What features would keep you and your friends motivated to go outside and run together?

10) Do you have anything else to add?

Questions 1 and 2, establish their history as a runner. This is important to understand how they would feel about an online and realtime running app.

Questions 3 to 5 investigates the running app which they currently use. I am able to understand the benefits and problems with these running apps, so I could use this information to enhance the stakeholders experience in my app.

Question 6 allows me to understand why they don't use a running app, which is helpful so that I can address this point in my own app.

Questions 7 and 8 lets me know what each stakeholder thinks about the idea of running online with someone, and gives me an understanding from a users perspective, of which features are needed for success

Question 9 is very important as it address each stakeholder specifically. Each stakeholder is a representative sample of that demographic, therefore it is crucial to address my target audience of both casual and frequent runners.

The final question allows them to mention any other points not discussed.

Response to first interview

Aron

1) How often do you go running?

"I tend to go running around 3-4 times per week. However, recently I haven't been going on as many runs as I would like to, because I can't run with my friends due to the current lockdown."

2) Do you use a running app?

"No"

3) Have you ever thought about using a running app?

"The idea of using a running app doesn't really entice me. I feel like they are a bit of a gimmick and I feel like they distract you from running."

7) *Explain app idea*. So what do you think about running in realtime with a friend online?

“This idea does sound quite unique. I would be quite interested in using this app. It would definitely be exciting to run with some of my friends and even run with my family who live outside of the UK.”

8) What features would you like the app to have?

“Firstly, I think it should be very easy to use. Also, it definitely shouldn’t take long to set up a race with a friend. I think you should be able to add friends on the app so you can quickly see who is online and invite them to race you.”

9) What features would like the app to have so that it suits your needs as a frequent runner?

“I would like to see accurate statistics about my running that I have never seen before, such as my top speeds, and average pace, so that I can analyse this and improve.”

10) Do you have anything else to add?

“Nope. Sounds great!”

David

1) How often do you go running?

“I try to do two 5km runs per week in order to stay fit for Sunday league football, however, usually I only end up running once per week.”

2) Do you use a running app?

“Yes, I have been using it for 6 months now”

3) What app is it?

“MapMyRun”

4) What do you like about that app?

“It is very simple and easy to use. It provides me with very cool and intriguing statistics which makes me feel like I am a serious runner.”

5) Are there any problems with that app?

“Not that I know of”

7) *Explain app idea*. So what do you think about running in realtime with a friend online?

“Sounds epic!! Thinking about it, it is exactly what I need to motivate myself to complete my goal of two 5km runs per week. I can just imagine using it with my teammates from my football team. It would be a brilliant way of maintaining our fitness.”

8) What features would you like the app to have?

“It definitely should be easy to use because I am not the most technologically advanced person in the world. I think it would be great if you could choose a distance for the race to take over, so that I could have the option to specifically choose a 5km race.”

9) What features would like the app to have so that it suits your needs as a frequent runner?

“It should send some notifications if I stop running for a few days, to make sure I don’t stop using it.”

10) Do you have anything else to add?

“One more thing, I think it is crucial to clearly display the statistics of both runners after a race has finished, in order to add to the competitive nature of the app.”

3.1.3 Research the problem

A. Research the problem and solutions to similar problems to identify and justify suitable approaches to a solution.

My clients problem: Not being able to run with friends, due to the corona virus outbreak, and this has lead to a loss of motivation for running. After the isolation terminates, he would also like to run with friends who don’t live locally, or even family in another country.

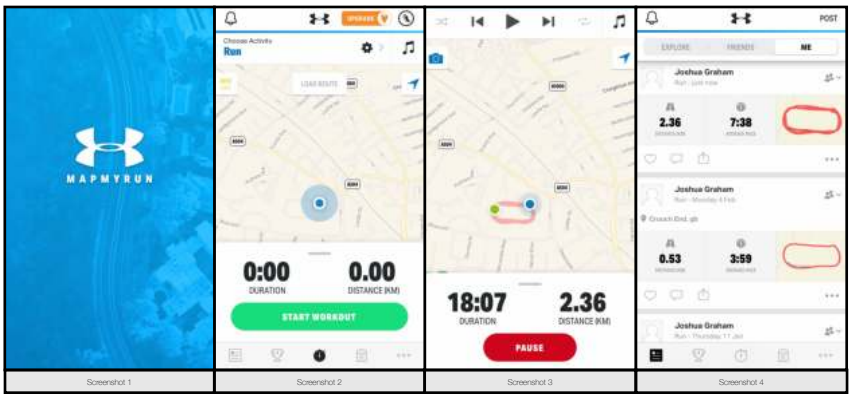
The problem with a majority of other running apps is that; although they allow the user to run, whilst tracking their distance and speed, and allowing them to share their achievements with their friends; they do not let you run at the same time as getting live updates on your friends running. This is the unique thing about my app. Personally, I think it would be incredible to run with a friend or family who are halfway across the globe, viewing their running statistics in realtime.

Other similar solutions:

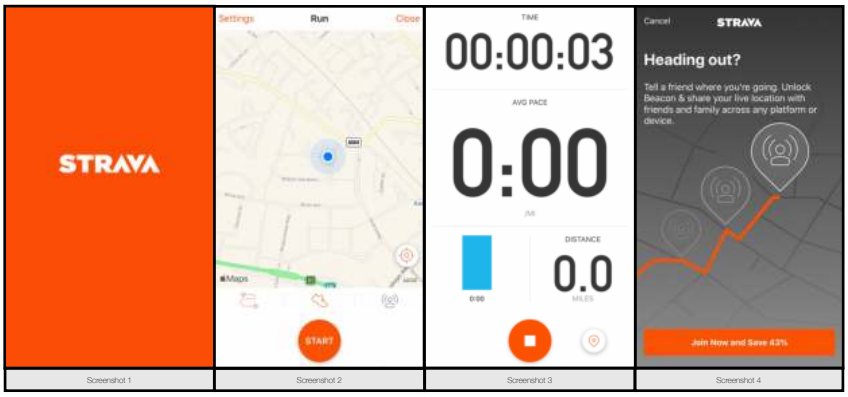
Strava

- Strava is a social-fitness network, primarily tracks cycling and running exercises, using GPS data. Strava is a free service, however there is a monthly subscription plan called Strava Summit.

- *Positives things about this solution:*
  - Very clean design and structure (Screenshot 2 & 3) - a user will find the app interface very inviting and in many ways quite exciting as you can see your progress on the map during your run.



  - Provides accurate and running data to the user: time, average pace, distance.
  - Its main feature is that it is very social network based:
    - A user can create/join a running club where users can all view each others recent runs and statistics (however, not live - only when they have completed a run).
    - The ability to share publicly loads of things such as: running challenges and running routes.
    - It even allows you the ability to follow other people so that their recent runs and statistic appear in the tab called feed.



- *Limitations about this solution:*
  - If a user has purchased the monthly subscription, they have the ability to share their live location during their run (Screenshot 4). However, this is a security feature for the runner so that their loved ones know that they are safe. However, the runner cannot see the location and statistics of another runner in realtime, whilst running - this is the main focus point of my project.
  - Continuing from the previous point, this app may solve the problem of my brother having no motivation to run, as it is quite cool to be able to see your own statistics as a runner, so you can monitor your progress over time. However, this app, along with many others, lack the ability to be able to view

friends live progress during a run which you both start at the same time, and finish together at the same time.

- My program will hopefully be as aesthetically pleasing, smooth and easy to use as Strava. However, this is quite unlikely as obviously Strava was a professionally developed app which most likely had a whole team behind developing it, whereas I am just one beginner in app development.

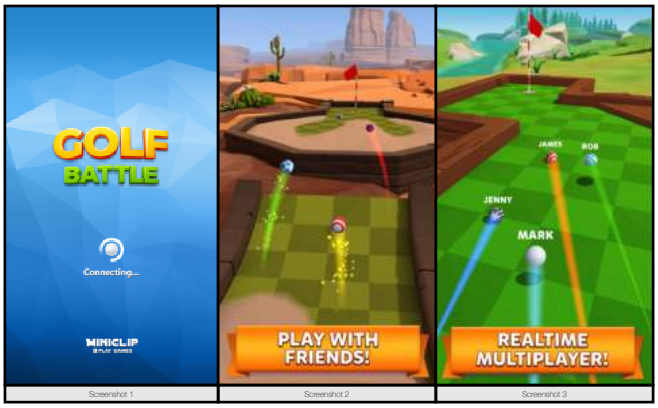
MapMyRun

- *Positives things about this solution:*
  - Again MapMyRun has a very clean design and structure (Screenshot 2 & 3) - a user will find the app interface very inviting and in many ways quite exciting as you can see your progress on the map during your run.
  - Also provides accurate and running data to the user: time, average pace, distance.
  - It also is slightly social media based:
    - You can add friends on the app and view their recent runs.
    - You can explore runs from the public on your feed.
    - You can view your recent runs and share them with your friends on the app, or on other social media platforms (Screenshot 4).
- *Limitations about this solution:*
  - Again this app is quite appealing and useful for runners who want to run individually. However, again this app lacks the ability to be able to view friends live progress during a run which you run together.

- In my opinion MapMyRun's interface is superior to the interface of Strava. However, Strava is more popular (more downloads) due to its social networking and sharing aspect - demonstrating that the ability to share runs with friends is key for a successful running app, and what better way to do it than realtime with a friend.
- Furthermore, a crucial point I would like to bring up about this MapMyRun and also Strava, is that these two apps do not allow the user to set the distance of the race/run, before it has started. However, my client has said that it would be a great idea for the user to be able to set the distance of the run when creating the online run with a friend. He said that it is important as it gives both runners a target to aim for.

Golf battle

- This online multiplayer game allows up to 6 players to connect and play together in real-time. Players compete against each other to finish the mini golf course in the fastest possible time.

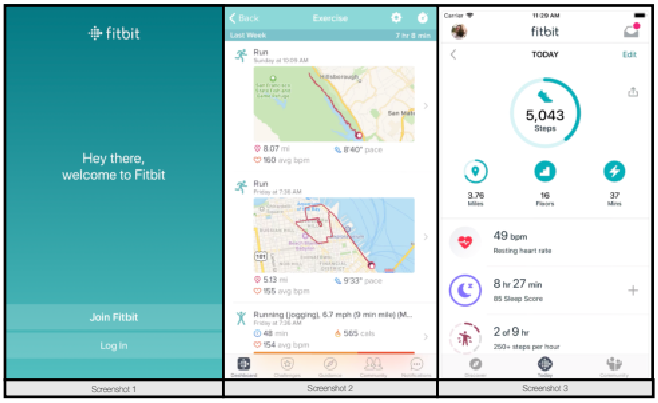


  - *Positives things about this solution:*
    - Competitive nature - makes it very enjoyable to play with friends.
    - Its realtime online gameplay is very smooth which is great for the user.
    - You can see the other players golf ball move in realtime (Screenshots 2 & 3), which really adds an element of excitement to the game.

- Obviously this mobile app does not track the users running. However, it is mainly the realtime online features which makes it an excellent example to follow and get inspiration from. I would love my app to be competitive, with a smooth realtime gameplay where the users can see their opponents distance in the race, to add to the excitement of the race.

Fitbit

- Fitbit is one of the leading companies in health and fitness, which is a slightly different to the purpose of my app (enjoyment and motivation). It is largely a product based company selling smart watches which can track data such as sleep, heart rate and nutrition. This app is therefore a mobile application which sidekicks the Fitbit smartwatches, allowing users to view the data recorded on their watches.



  - *Positives things about this solution:*
    - Very accurate tracking of all-day stats (such as steps, distance, calories burned, floors climbed), due to the fact that the app is connected to the smart watch, which has more accurate sensors than a mobile phone.

- This app also doesn't allow users to connect in realtime online. However, the focus of this app is its remarkable accuracy with tracking running statistics. This of course uses separate hardware, which will not be feasible in my solution.

B. Describe the essential features of a computational solution explaining these choices.

The essential features of my solution are:

1. A pleasant and intuitive user interface so that the user can easily navigate through the app and smoothly start a run with a friend.
2. The use of GPS, in order to track the users location, speed, distance, altitude, and other important data.
3. An efficient method of storing users accounts and saving their running data, so that it can be accessed next time they open the app, without being lost.
4. The ability to connect online to a friend, and get updates on their running data and location. This is the unique feature of my project.
5. The ability to share achievements and runs with their friends. This is clearly important as it is what keeps users motivated and engaged on the app, as they know they are being recognised for their hard physical work.
6. The ability for the user when creating the run, to be able to set the distance of the race/run. This is important as it gives both runners a target to aim for. This is not present in any other current app.

C. Explain the limitations of the proposed solution.

The limitations of my solution are:

1. It is most likely not going to be as aesthetically pleasing as the apps shown in section A. This is obviously because this is a one man project, whereas the apps have teams of developers who created them. However, the design of the app is very important to me, so I will make sure that the app will be easy to use and appealing, although, it may not be of the same level of a professional app.
2. This app will only be available for iOS (iPhones) and not android phones. This is because I will be using Xcode IDLE and programming in Swift 5, which only compiles apps for Apple devices. However, this will not affect the testing of the app, as me and my brother both have the same iPhone. This will reduce my target market, as only iPhone users will be able to download the app, however, this is still a very large number of people (around 900 million). Many businesses and companies have taken on this strategy of developing their app for iOS first, and then for Android if all is successful. This way, development is faster and cheaper as you don't need to worry about different processing specifications, touch gestures and sizes of the thousands of different Android phones.

3. Security of each user. Obviously, I am not an expert in cyber security, therefore the database may not be secure, so users data could be stolen. Also, I will need to access the users location whilst they are running, which also pose some safety risks.
4. The main limitation of my program is going to be the accuracy:
    1. Of using GPS to track users running data. Some areas of the country can have poor GPS (such as a tree-covered park), meaning that it will be very slow at updating the users location, distance and speed, which may lead to complications in the app. Also, their is a general delay with GPS, as you are communicating with satellites which are thousands of kilometres away (e.g. The delay when using a Satnav system). This may cause another delay in a user running and may reduce the accuracy of the app. I will definitely focus on this when I am testing my finished project.
    2. Of connecting multiple users online and updating each others location. It may be difficult to keep the data of another user you are running with updated in realtime, as this requires networking (which is expensive, complicated and time consuming). However, I will be using a realtime database to keep users connected in realtime. But this solution may mean that there is a delay between each user.

    - These two delays together, may cause a much bigger delay, which may hinder the joy and experience of the app.

3.1.4 Specify the proposed solution

A. Specify and justify the solution requirements including hardware and software configuration (if appropriate).

Hardware:

- I'll be using my MacOS laptop because I can download Xcode IDLE, as it is only available for apple devices. Hence, I will also be using the laptops standard peripherals: display, keyboard, mouse, in order to code my project.
- I'll be using my iPhone to test my app as I develop it, as you can only build the apps for apple devices. A may also use my brothers iPhone, in order to test if the online/connection features are working.
- As I have mentioned above, the users will need an iPhone (which runs iOS 10 or later) to run the app. During the development and testing process, you can send a download link to up to 10 testers. Hopefully in the future, I will be able to upload my app onto the iOS app store so that the any iPhone user in the world can download the app.

- Finally, it is important to note that every iPhone user will be able to download the app, however, not every iPhone user may be able to use the entirety of the app. This is because in order to actually use the main feature of the app (racing against another player online), users will need to have access to mobile data - which means that they will need a phone with a working sim card (which I have estimated needs to give them at least 3G internet speeds). This is because the race is online so an internet connection is required to establish a connection between the two racers and then also to allow them to communicate with the database. A WiFi connection would not be appropriate because the users will be running so they would lose connection to the WiFi once they are far enough from the router.
- Therefore, to summarise, the hardware which a user needs to use this app fully is an iPhone (running iOS 10 or later) with a sim card which provides speeds of 3G or higher (4G/LTE or 5G). The app will work best for people in areas with fast mobile internet speeds and with a good GPS signal - therefore it is more suited for people living in urban areas.

Software:

- My laptop must have macOS 10.13.6 or later to run the latest version of Xcode.

- I will be using Xcode IDLE to develop my app. Xcode is an integrated development environment for macOS which contains an array of software development tools for developing software for Apple devices. I am using Xcode due to a few reasons:
  - It is the only legitimate program you can use to make native Apple product applications - and I have an iPhone so it is convenient for testing my application.
  - You can test your iOS apps either in the simulator that comes with Xcode, or on your own physical iOS device
  - Swift 5 is a well documented language, with lots of resources all over the internet to learn the programming language. This is great if I need to learn something new in the language, or if I am stuck, there is a large community of Swift users out there who are ready to help.
- I first intended on using SQLite, as it is what we had been using in class with python. However, the integration of SQLite with Swift is much more difficult. Therefore, I will be using Google Firebase Database. This is great as at gives you a flexible way to scale up your database through the cloud, and the speed is realtime, which is exactly what is needed for my realtime online running app.

B. Identify and justify measurable success criteria for the proposed solution.

Measurable Success Criteria

To judge the measure of success of my solution, it will need to meet a set of criteria that my client and I outlined during our first discussion. These are a set of measurable criteria which we have decided to measure the projects success by:

Criteria	Evidence
No errors or bugs in the final product.	In testing and evaluation phase. I want any volunteers to find no bugs and there be no crashes when using the app in the testing phase.
The app is intuitive and easy to use and navigate.	Through final feedback questionnaire in testing phase.
Reduced lag time between players.	I would like there to be less than 0.2 seconds in lag time between updating distances on each persons phone. I will measure the time differences in the testing phase.
The app motivates the user to go running.	80% of users say the app motivates them to go on a run. In testing phase.
It is easy to set up and create a new run	90% of users say it is easy for to create a new run. In testing phase.
The final time is accurate	I will go to a local 400m running track. I will create a 400m race and walk the race whilst holding a stopwatch in my other hand. By walking it reduces the effect of human error. I will then compare the time on the stopwatch to the time on the app. The times should be the same. Ideally I would like the times to be no more than 0.5 seconds apart.
The average speed is accurate	I will walk a 100m race whilst also using a speedometer on another phone. I will record my speed from the speedometer at 10 constant intervals throughout the race (every 10m). I will then calculate the average speed by summing these values and then dividing by 10. My calculated average speed should be the same as the average speed given on the app. I would like the average speeds to be no more than 0.5 kilometres per hour apart.
The top speed is accurate	I will walk a 100m race whilst also using a speedometer on another phone. In the middle of the race I will run and record my peak speed. My recorded top speed should be the same as the app's top speed.
The shape of the distance-time graph is accurate	I will walk at varying speeds a 100m race. Before the race I will mark out 10m intervals. During the race I will record the time when I pass an interval. I will then plot a line graph using Excel and compare the two graphs. The graphs should look similar.

Other Success Points

I also have set myself some success points which I would also like to accomplish with my solution. By creating this set of success criteria, it will keep me on track throughout my project, because I recognise which feature are most important to my

app. I will be referring to this list of success criteria constantly throughout the rest of my project:

Success point	Evidence
Efficient code	Through time testing
Well documented code / well commented - anybody reading it will understand it.	Through screenshots
Receive realtime updates from a database	Screenshot of code and testing
Receive realtime GPS data	Screenshot of code and testing
Allows the user to create an account which is then stored in a database	Screenshot of code and testing
The user can join a run with a 4 digit code	Screenshot of code and testing
The user can select the distance of the run	Screenshot of code and testing
Displays the users location on a map	Screenshot of code and testing
The race will only start when both users are ready	Screenshot of code and testing
Shows both users progress throughout the race as a progress bar	Screenshot of code and testing
Displays the users current speed throughout the race	Screenshot of code and testing
The user has the option to change the speed units from miles to kilometres	Screenshot of code and testing
Calculates the users average speed	Screenshot of code and testing
Calculates the users top speed	Screenshot of code and testing
Displays a clear comparison between the running statistics of the two competitors	Screenshot of code and testing
Correctly shows the winner of the race and each players final time	Screenshot of code and testing
Shows a distance-speed graph	Screenshot of code and testing
Shows a distance-time graph	Screenshot of code and testing

3.2

Design

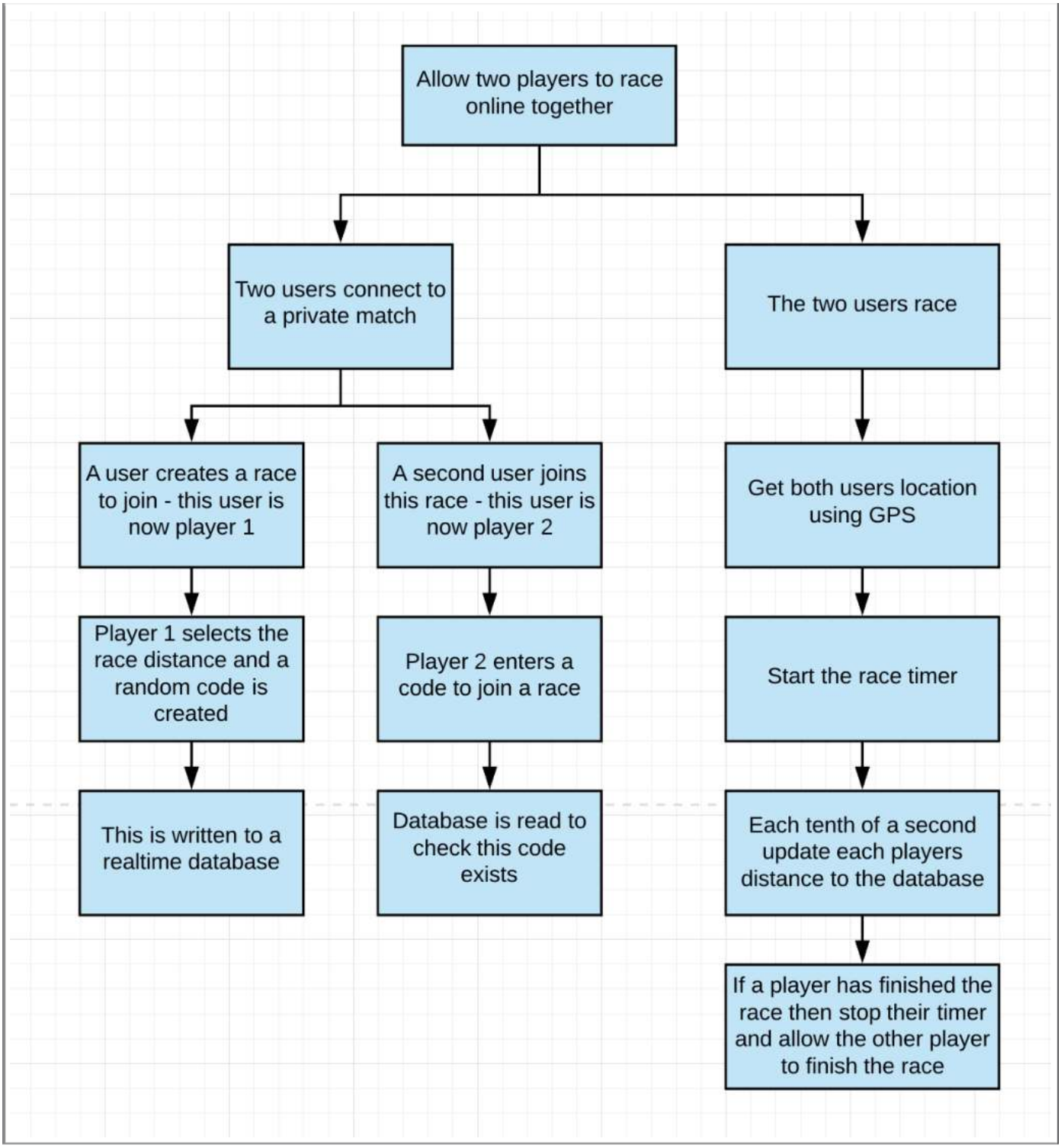
3.2.1 Decompose the problem

A. Break down the problem into smaller parts suitable for computational solutions justifying any decisions made.

Decomposing the flow of the app

To understand how my stakeholder's complex problem can be solved, I need to break down his problem into manageable parts.

Firstly, the main aim is to obviously allow to users to race against each other. This can be broken down into smaller problems:



This solution for my stakeholders can be separated into two main challenges, connecting two users to the same race (‘connection’) and then carrying out the actual running race (‘race’).

These two sections can be split in the following way:

Connection:

- Creating race (player 1):
  - Player 1 selects a distance
  - Create random 4-digit code
  - Create a new record in the database for this race and wait for player 2 to join
- Joining race (player 2):
  - Player 2 enters 4-digit code
  - Find this code in the database and if it exists notify player 1 that player 2 has joined

Race:

- Start updating the users location using GPS
- Wait until both users are ready to start
- Start the countdown timer (3, 2, 1, GO)
- Start the race timer
- Update the user’s total distance ran using GPS data
- Constantly update the users distance to the database
- If a player has finished the race then stop their timer and allow the other player to finish the race if they haven’t finished

Link to success points:
Receive realtime GPS data

Here is each step further broken down:

Connection

- **Creating race (player 1)**

- **Player 1 selects a distance**

  - Player 1 will select the distance of the race from a menu. The distances which can be selected will be: 50m, 100m, 200m, 400m, 1km, 2km, 5km, 10km

- **Create random 4-digit code**

  - A 4-digit code will be randomised. This step is important as it is the code which player 2 will use to join the race.

- **Create a new record in the database for this race and wait for player 2 to join**

  - Create a new record in the database. This record will contain a column for the 4-digit code and the selected distance of the race. It will also contain a boolean value which will be set to true when player 2 joins. This will allow player 1 to be notified when player 2 has joined.

- **Joining race (player 2):**

  - **Player 2 enters 4-digit code**

    - They will enter the code using an on-screen keyboard. Validation will be needed to ensure player 2 can only enter 4 digits, and these can only be numbers.

  - **Find this code in the database and if it exists notify player 1 that player 2 has joined**

    - Read the database to see if a record exists with this code. If this code exists update the boolean value, mentioned previously, to notify player 1 that player 2 has joined the race. If this code doesn't exist then alert the user that the code is incorrect and allow them to re-enter the code.

Race

- **Start updating the users location using GPS**

  - Privacy is very important in mobile applications, especially when accessing the user's location. Therefore, before the app can start updating the users location using GPS, it will need to make sure that I have permission to start using their location. If it does not have permission, then the app will request access to the

users location. Once GPS has been permitted, their location will be shown on a map view, very similar to google maps.

- **Wait until both users are ready to start**

  - When the user in the race screen, allow them to press a button which says ready. When they press ready update this in the database. When both players are ready, the race countdown can begin.

- **Start the countdown timer (3, 2, 1, GO)**

  - Only after the previous step (once both players are ready), this timer can begin. Using the built in timer function I will create a timer which will count down from 3 and will display the number on the screen.

- **Start the race timer**

  - Create another timer which will count up from 0 by 0.01 every 0.01 seconds. Therefore, 0.01 seconds the timer will update via the updateTimer function.

- **Update the user's total distance ran using GPS data**

  - Every time the user changes location (i.e, moves), the GPS location is updated. From this, the speed of the user can be calculated and therefore using the equation 'speed = distance/time', distance can be calculated (as the time is also known). This distance can then be stored in a variable and constantly updated over the course of the race.

- **Constantly update the users distance to the database**

  - The users distance can be written to the database constantly using the updateTimer function, which is being called every 0.01 seconds. This then ensures that both players distance are updated in the database in realtime.

- **If a player has finished the race then stop their timer and allow the other player to finish the race if they haven't finished**

  - If the user has exceeded the distance of the race, then their timer is stopped and their final time is written to the database, as well as their other statistics (like top speed and average speed). Then the database is read to check if the other player has finished the race. If they have finished the race then show the final results. If the other player hasn't finished the race then the user must wait until the other player has finished before viewing the final standings.

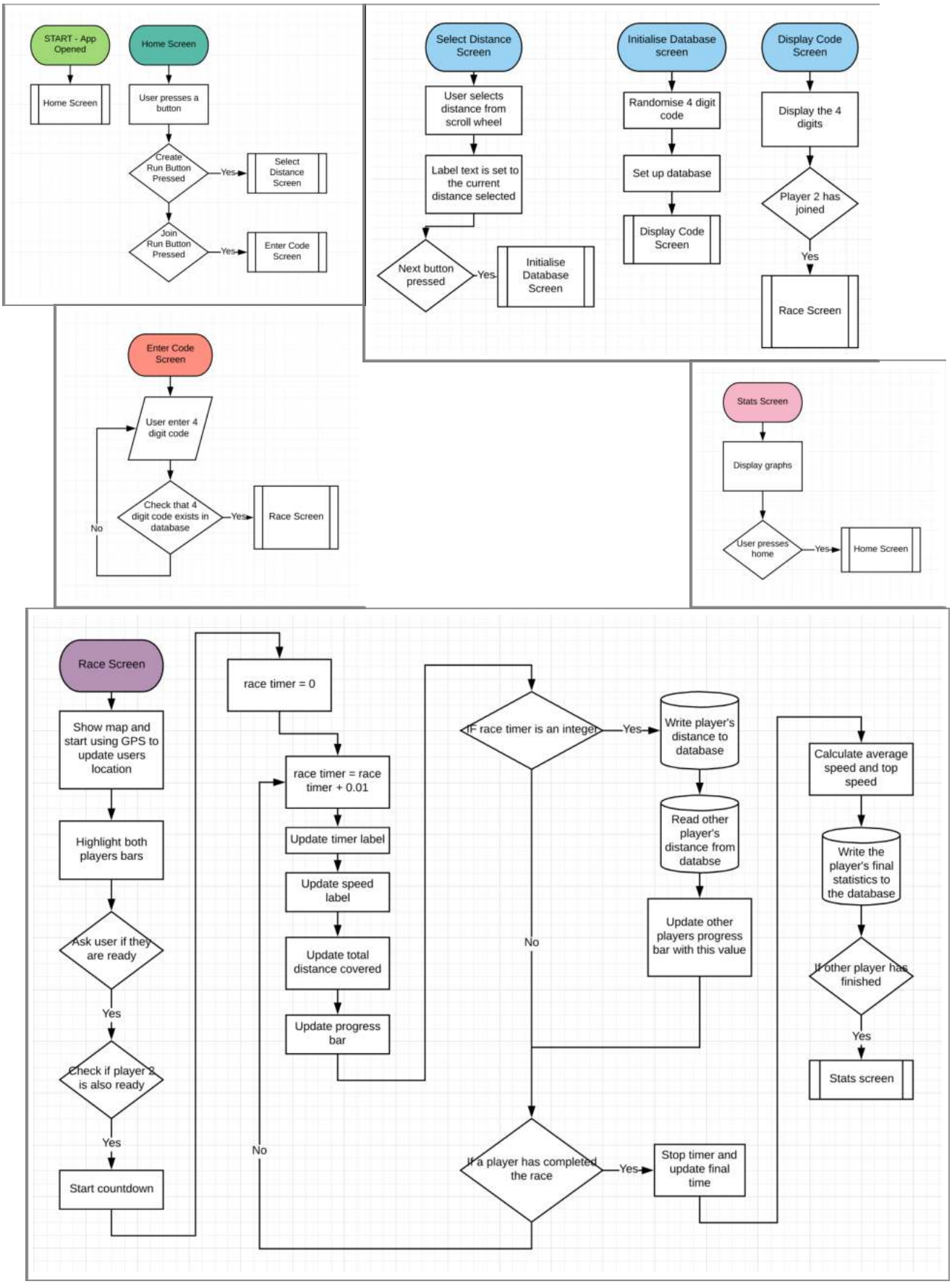
Note on user accounts

I have taken the decision to not implement user accounts in my solution. This is because it is not a key part of the solution and it would mean my project would have too big of scope, and due to the time restraints it would just not be possible.

It is more important to focus on perfecting the key and unique part of my solution - the race, i.e. allowing two users to race online.

Flow of screens

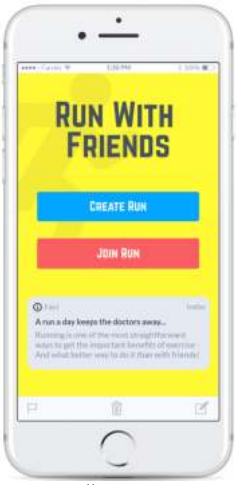
This image summarises how each of the above step relates to the flow of the screens.



User interface design

- Now that I understand the flow of the program, I can now design the user interface of my solution.

- I have designed my solution using the help of marvelapp.com - a website which is great for creating mobile application prototype for iOS or android. It even allows you to configure the flow of the program and then view your prototype on a model phone. I began by designing the home screen:



Home screen

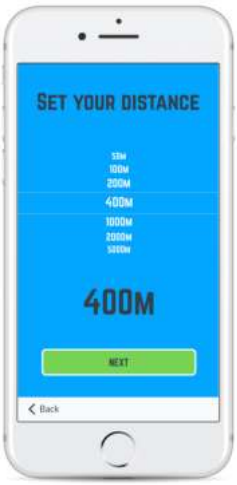
- ***Home screen***

  - It has a clear and well spaced-out layout.
  - It has two buttons: to either join or create a race.
  - It has a toolbar at the bottom which will allow the user to do various things such as share the app with friends.

- ***Select distance screen***

  - If the user selects the create run option, the user will be brought to this screen.
  - They are presented with a dropdown picker where they can select the distance of the race.
  - They can then press the button at the bottom to go to the next screen
  - There is also a toolbar at the bottom which contains a back button to allow the user to go back to the main screen.

Link to success points:
The user can select the distance of the run



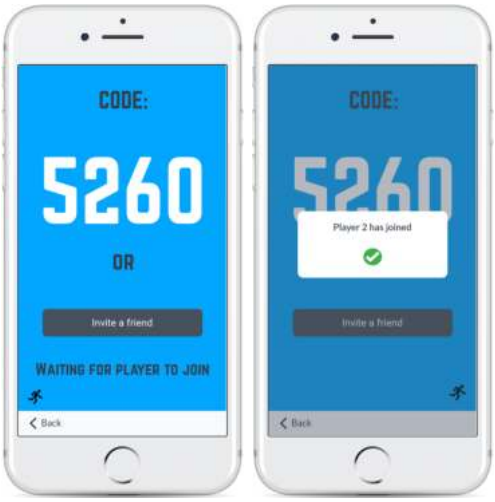
Set distance screen

- ***Initialise database screen***

  - This next screen will only display for a very short amount of time.
  - It will only display a spinning progress wheel, however, this is the screen where the user will initialise and connect to the realtime database.

Initialise database screen

- ***Display code screen***



Display code screen

- This screen will display the randomly generated unique 4 digit code to the user.
- The user will also have the option to invite a friend.
- When player 2 has entered the code and joined the race, the user will be shown an alert, and then the user will be taken to the race screen.

Link to success points: The user can join a run with a 4 digit code

- ***Enter code screen***

- If the user selects the 'join run' button from the home screen, they will be taken to this screen.
- The user will be presented with a keypad and they will be able to enter the 4 digit code.
- If it is successful, they will be shown an alert, and then will be taken to the race screen.
- If the code was incorrect they will also be shown an alert.

Enter code screen

- ***Race screen***

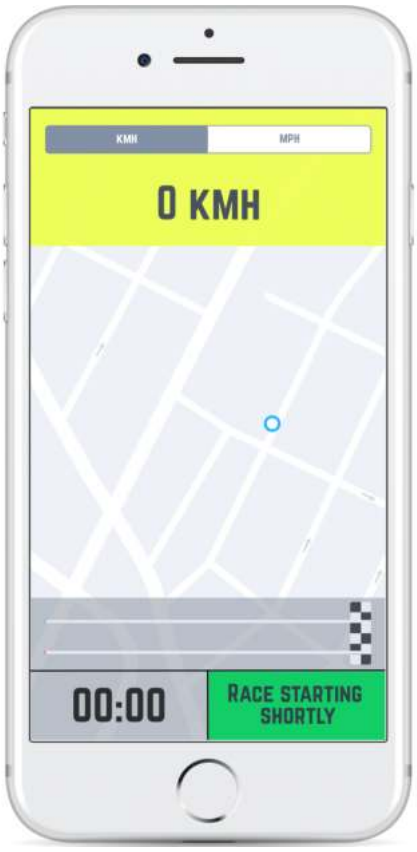
- When both users have connected to the same race they will be brought to the race screen. This is the most important screen as it is where both users will spend the most time on.
- The map view takes up the entire screen, however, with lightly opaque views in front.

Link to success points: Displays the users location on a map

- The top (yellow) view displays contains a label which displays the users current speed, where the units can be changed between kilometres per hour and miles per hour via a segmented control.

Link to success points: Displays the users current speed throughout the race & The user has the option to change the speed units from miles to kilometres

- The bottom (grey) view contains a timer label in the bottom left hand corner, an information label in the



Race screen

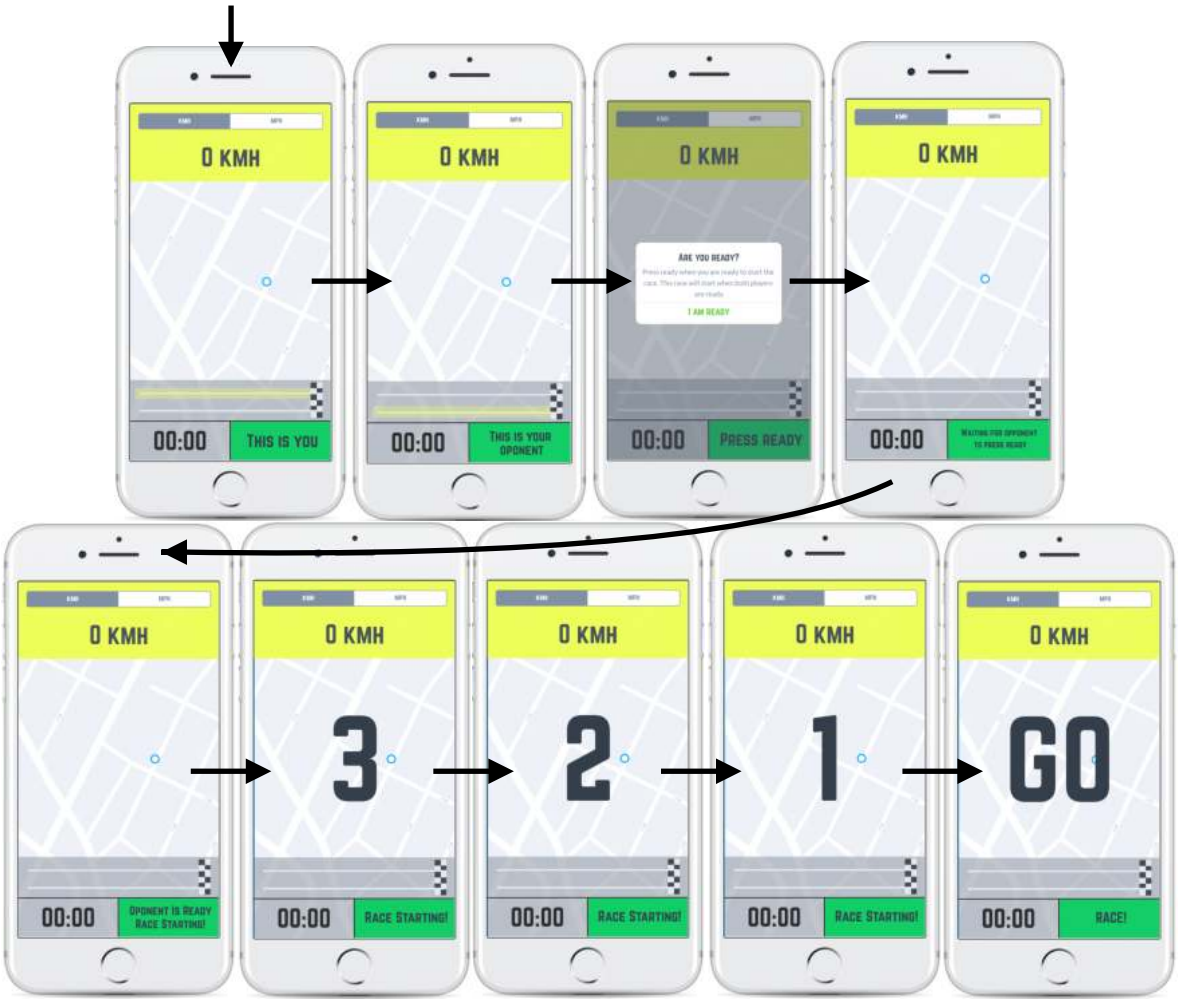
bottom right and another view container.

- The timer labels displays the time since the race has started
- The information label will display various text as information, such as ('Get ready' or 'race starting now')
- The view container contains two progress bars, and an image of a checkered finish line. Its purpose is to display the progress of both players in the race, i.e. how far they have run out of the total distance of the race.

Link to success points:
Shows both users progress throughout the race as a progress bar

- **Starting the race**
  - Before the race starts it is very important that the following are done in order:
    1) Highlights which progress bar is you and which is your opponents
    2) Allows the user to select a ready state through an alert containing a button
    3) When both players are ready, the race will start, displaying in big font 3,2,1 GO!

Link to success points:
The race will only start when both users are ready



Race screen series of events

Design feedback

I sent my designs to my stakeholders attached to the following email:

“Hi (name),

I have been working very hard on the prototype of the user interface of the app. Please find the attached diagrams of how the app would look. I have focused on creating an elegant but also user friendly design by using large text, large buttons and contrasting colours. I have also ensured that the structure and flow of the app is intuitive. I would love to know your thoughts about the design and flow of the app.

Thanks,

Josh”

I received the following replies:

Aron:

“I love this initial design. I think the colour scheme is very cool and in a way quite modern. The structure you have devised of either creating a run or joining a run from the home screen is very clear and simple. Finally, I think the use of a 4-digit code as the key to join a race is great as it easy to remember and is quite familiar. Overall, I love it and I can’t wait to use it soon!”

David:

“These designs look impressive. I love your colour choice it looks very clean. My favourite section is the design of the race screen - the layout is spot on. Firstly, I love how you will displaying valuable information such as the timer and the user’s current speed. Furthermore, the green information box in the bottom right of the screen is a great idea and I love how it is dynamic. I also like how you gave the user substantial flexibility being able to customise the units of the speed with the segmented control. Finally, the flow of this screen is excellent at the start of a race, it makes it very straightforward for a new user to understand.”

3.2.2 Describe the solution

C. Describe the parts of the solution using algorithms justifying how these algorithms form a complete solution to the problem.

D. Describe usability features to be included in the solution.

Deciding which stats to measure?

In order to determine what data I will need to collect and store in the database, I need to know which statistics the user will want to view following the race. Therefore, I thought that this was a great opportunity to ask my stakeholders what statistics they would like to be provided with after the race. I sent them the following email:

“Hi (name),

Just before I start designing the algorithms and the structure of the database, I would like to ask what statistics you would like to view after you have finished the race? Currently, I think it would be a good idea to present each user with their final time, average speed, and top speed. Maybe there are other statistics you would like to view? What format would you like the statistics to be presented in?

Thank you very much,

Josh”

I received the following replies:

Aron:

“I think showing the user their final time is obviously crucial and showing the user their average and top speed is a great idea. In terms of the format the statistics should be presented in, I think a graph would be a brilliant way of engaging frequent runners like myself. Distance-time graphs and speed-time graphs are simple graphs which would be suitable to present to the user. I think it would be a great idea if the graphs could show both the users data and also the opponents data on the same graph - it would really add to the competitive nature of the app.”

David:

“The stats you are measuring at the moment are simple to understand. I am assuming that you will also show the user their opponents final time, average speed and top speed. It would be amazing if the user could easily compare the stats from both the players. For example, a screen which compares these stats, showing each

player's stats side-to-side, displaying who came 1st and who came 2nd for each stat."

Design of stats screen

Reflecting on both these emails I designed the stats screen:

I have followed on with the colour scheme of blue and red to ensure that it is clear which stat belongs to which player. I will use a pageView to display different stats on each page.

The first page will display a comparison between the two players of their time, average speed and top speed. You can easily compare the stats between both players as requested by David.

Link to success points: Displays a clear comparison between the running statistics of the two competitors

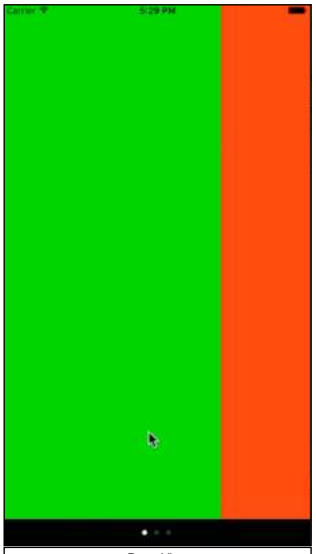
The second page will also displays a visual comparison between the two platers and incorporates a graph as requested by Aron. I also plan to have a page which will show a distance-time graph of the user.

Link to success points: Calculates the users average speed

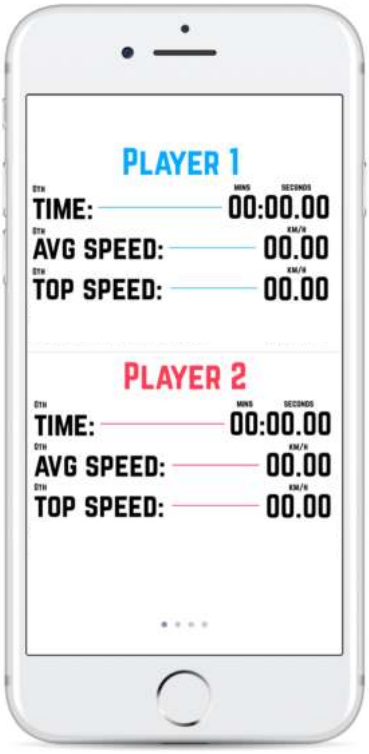
Link to success points: Calculates the users top speed

Link to success points: Correctly shows the winner of the race and each players final time

Link to success points: Shows a distance-time graph



Page View



Stats screen

Link to success points: Receive realtime updates from a database

Database design

For the database I will be using Google's Firebase database. One of Firebase's key features is the ability to create a realtime database, as well as app analytics and hosting.

A realtime database is defined as 'a system which uses real-time processing to handle workloads whose state is constantly changing'. It is necessary for my solution because it involves a running race, therefore the data for each user needs to be updated in realtime in order for the race to be fair. If a normal database was used, for example, player 1 may win the actual race, but because hasn't been updated immediately to the database, player 2 may be told that they won.

I have chosen to implement Google's realtime database because:

- Data is synchronised in realtime to every connected user - this is extremely useful for two users running simultaneously.
- The Firebase Realtime Database SDKs can be integrated into Swift.
- I can use Google's servers as opposed to having to run my own servers. Google is a trusted company.
- It is free.

The obvious solution, as opposed to a realtime database, for a large online multiplayer game would be to use HTTP requests and several servers. However, these are often programmed by specialists in this field and it can become very complicated. However, by using a realtime database it really simplifies my task of coding this solution.

This realtime database is called Cloud Firestore is a NoSQL, document-oriented database. As opposed to an SQL database, there are no tables or rows. Instead, you store data in 'documents', which are organised into 'collections'.

Structuring the database

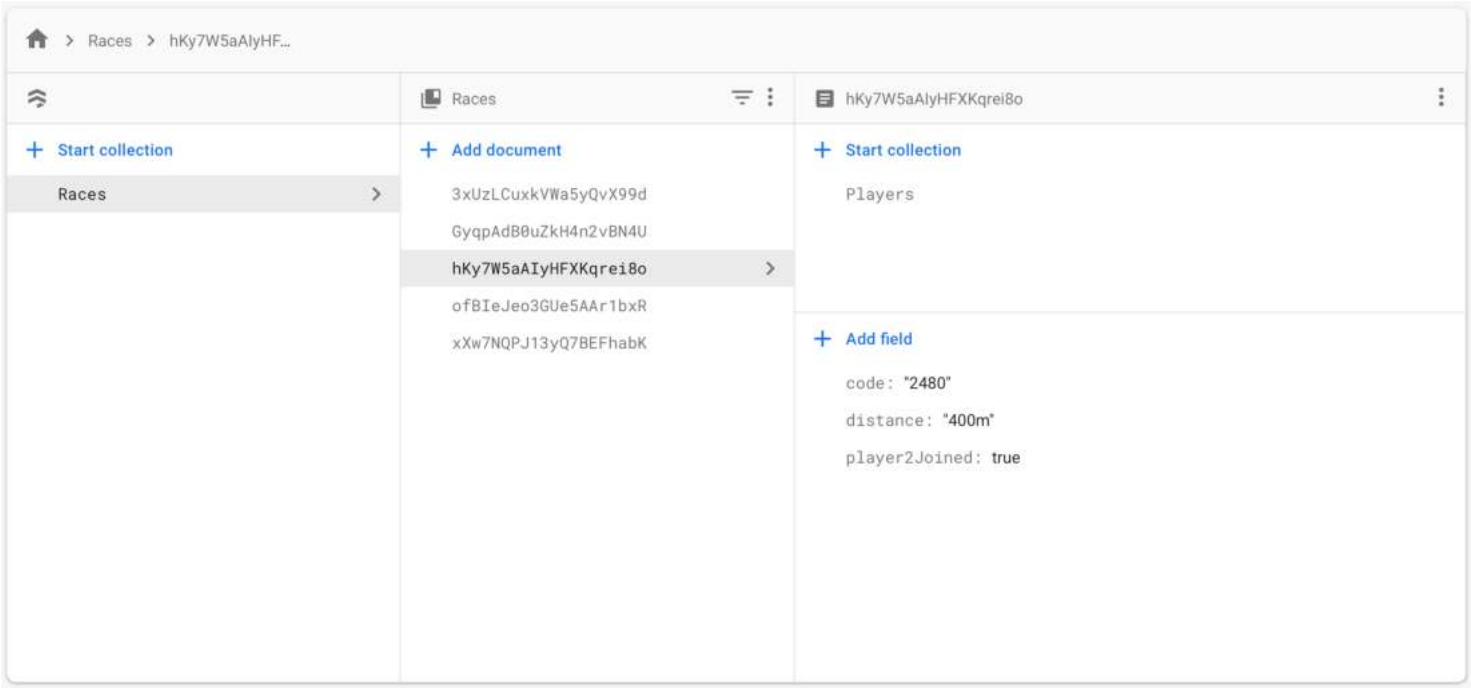
My first collection is called Races. This will contain all the data for every race.

The next layer inside the Races collection is the documents layer. Each document will represent one running race created2 by a user. It contains 3 fields:

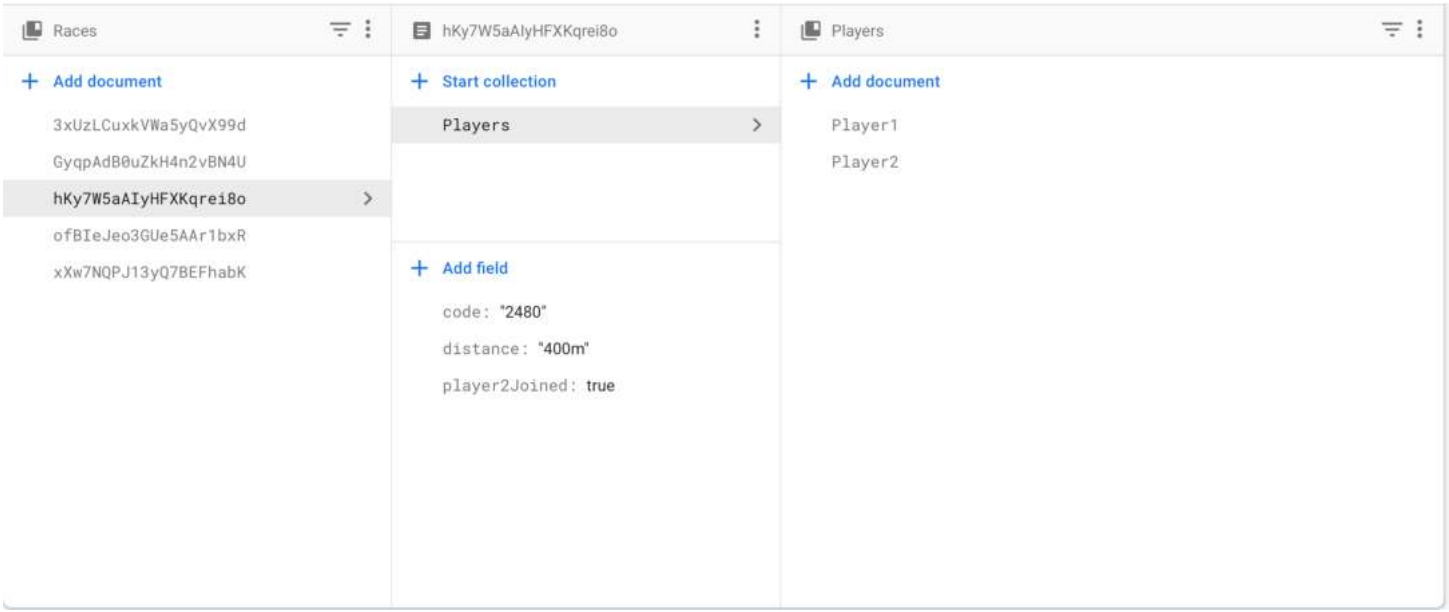
- code (string) - stores the 4-digit code used to join the race.

- distance (string) - stores the distance of the race.
- player2Joined (boolean) - is set to true when player 2 has entered the code for that race.

These document's ID's are set to 'auto-id' which means that when a new race is created by player1 a new document will be created with a random string of characters. It is done this way because no two documents can have the same name.

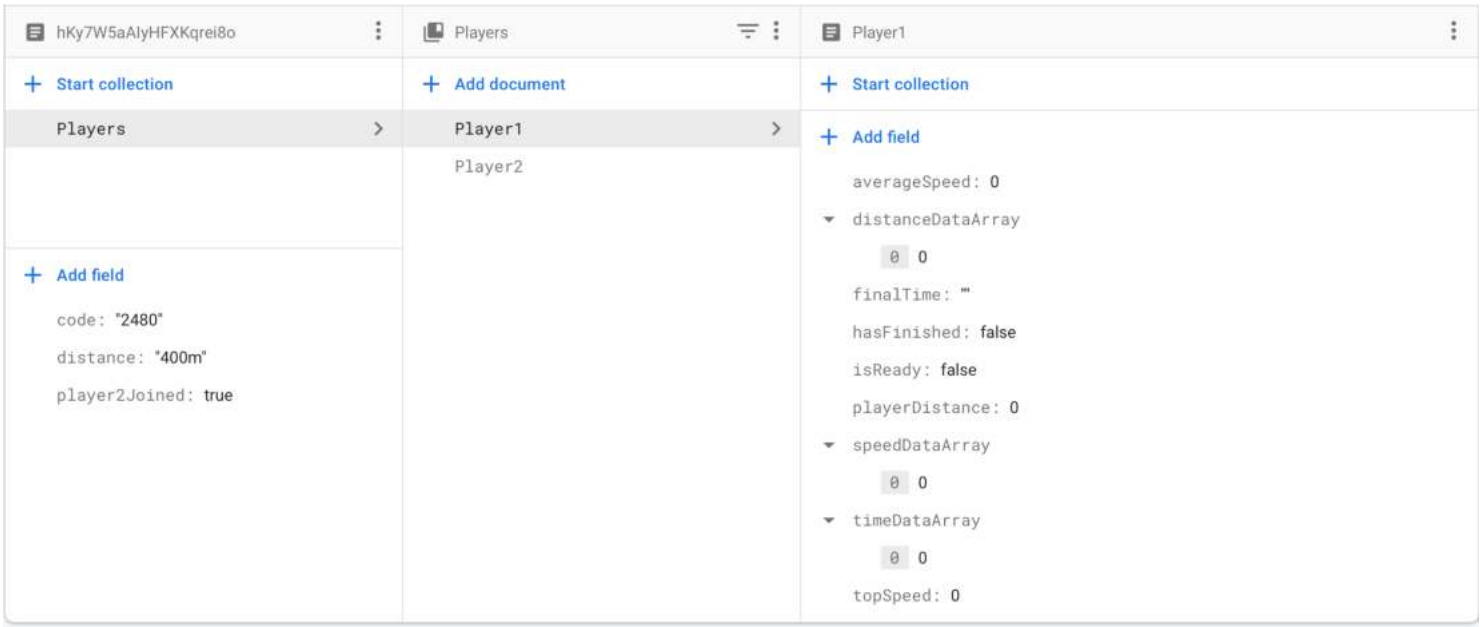


When each of these documents are created they will contain a collection called Players. This collection will contain two documents: Player1 and Player2.

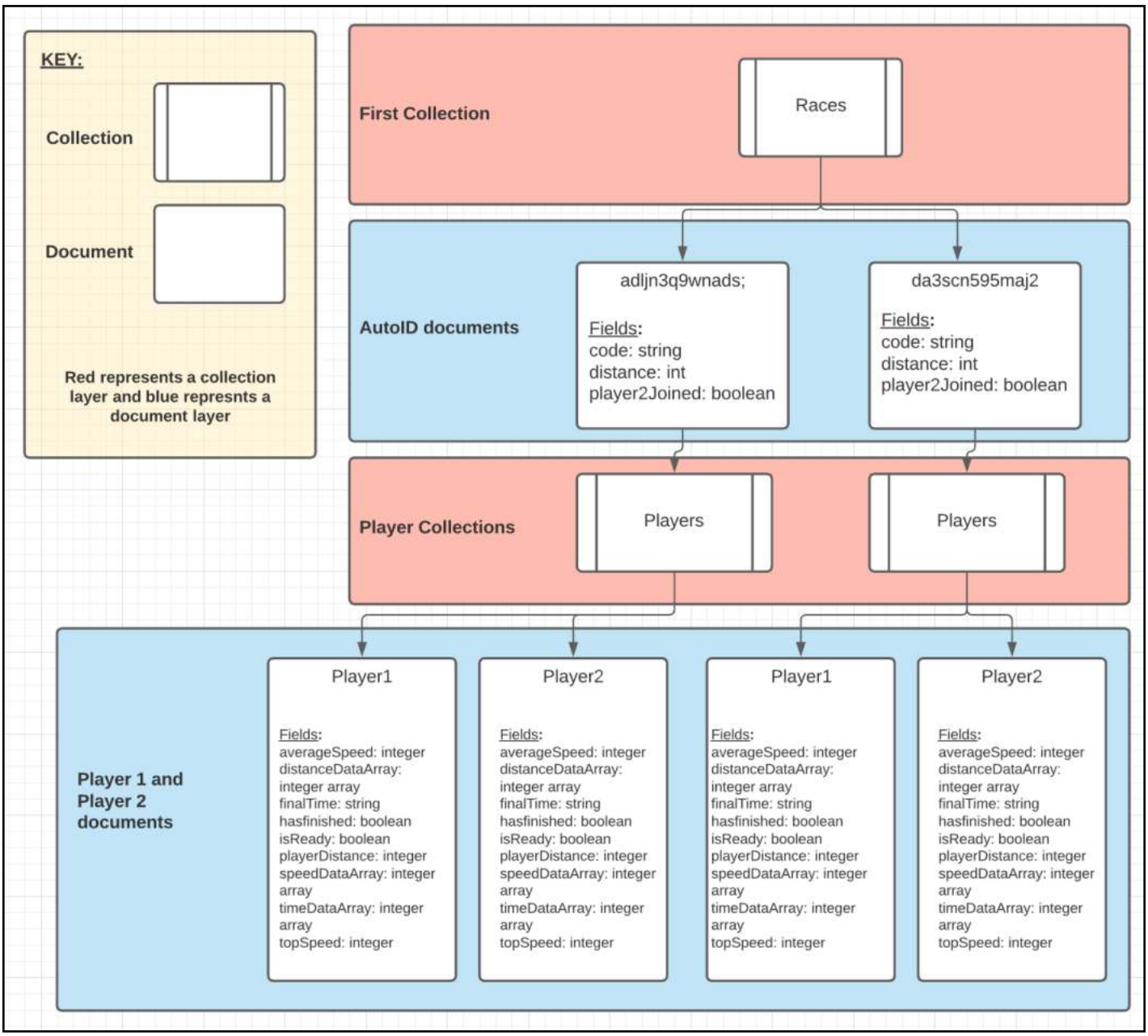


Both these documents will be created with the same fields:

- averageSpeed (Integer) - to store the the player's average speed.
- distanceDataArray (Array of Doubles)- to store the player's distance over the course of the race (useful for the graphs).
- finalTime (String) - to store the users final time.
- hasFinished (Boolean) - to determine whether that the player has finished. This will be set to true when the player finishes the race. This is useful because you can easily check if the other player has finished their race.
- isReady (Boolean) - to determine whether the player is ready to start the race. This will be set to true when both players press the ready button.
- playerDistance (Double) - to hold the current distance of the player. This is useful so the two progress bars can be set to this distance (as a percentage of the total distance of the race) to display the progression of each player over the course of the race.
- speedDataArray (Array of Doubles)- to store the player's speed over the course of the race (useful for the graphs).
- timeDataArray (Array of Doubles)- to store the player's time over the course of the race (again useful for the graphs).
- topSpeed (Double) - to store the players top speed.



Overall, this is a diagram of my database structure:



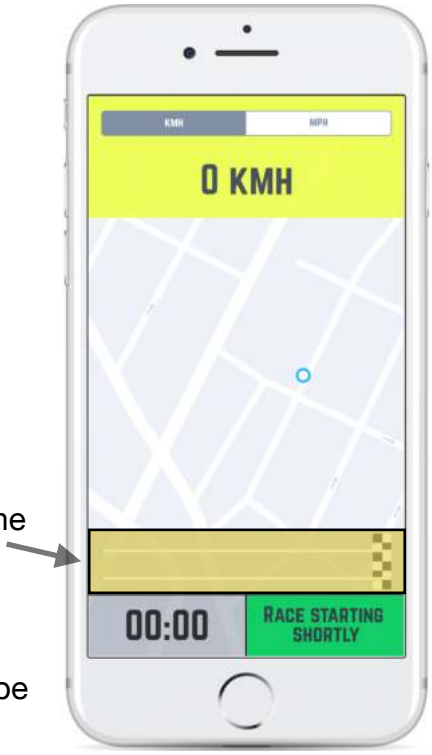
Algorithms/Pseudocode

My solution has 7 screens (as seen above from the flow of screens diagram). In Xcode and Swift, each screen needs there own file, therefore, I will have 7 files. In this section of my project, I will go through each file one-by-one and describe them using algorithms and explain how they form part of my complete solution.

In Swift each file is automatically created with the function called 'ViewDidLoad()', which is called every time that screen is loaded up. This is useful if you want to set something at the start of the screen loading up. I have used the special notation '*procedure' to represent that this function is a special function.

The Player1 and Player2 complication

Before I describe the pseudocode, I think it is very important to describe the difficulty with having two players:



- Users are tagged player1 or player2 depending on whether they choose to create the race or join a race.
- It is necessary to have a player1 and player2 because:
    - Each player has there different GPS data. So to separate them in the database the two players need to be distinguished.
    - The progress bars at the bottom of the race screen display the progress of each player throughout the race. The top one will always show player1's progress and the second one player2's progress.
    - For a user write to the database, they will need to know whether they are player1 or player2, so this data can then be read by the other player.
- The same app however is run for both these players. Therefore the app needs to differentiate whether the user is player1 or player2.
- The distinction between player1 and player2 can be seen in my algorithms below and the development section of this project.

Algorithms prior to the race screen:

The algorithms prior to the race screen are mainly database based. I have decided to not include the database related code in the pseudocode because it is very specific to the database language, and will be shown in the development stages. I will summarise the database code in basic english.

Home screen

This screen does not have any interesting algorithms. Its only purpose is to display the two buttons: ‘create a run’ and ‘join a run’. Therefore, I have not included any pseudocode for this screen.

Select Distance screen

This is the screen where the user selects the distance of the race. The user will choose the distance out of a selection from a scroll wheel. This distance is then returned and stored in a variable called ‘distance’, which will then be passed to the next screen.

```
// MARK: Variables
distance = 0

*procedure viewDidLoad()
    allow user to select distance from scroll wheel
    //The variable 'distance' stores the race distance so it can be passed to the other files to be used.
    distance = selectDistance(usersSelection)   //The argument 'usersSelection' is the user's selection from the scroll wheel.
endprocedure

function selectDistance(selection)
    if selection=="50m" then
        return 50
    elseif selection=="100m" then
        return 100
    elseif selection=="200m" then
        return 200
    elseif selection=="400m" then
        return 400
    elseif selection=="1000m" then
        return 1000
    elseif selection=="2000m" then
        return 2000
    elseif selection=="50000m" then
        return 50000
    elseif selection=="10000m" then
        return 100000
endfunction
```

Link to success points:
The user can select the distance of the run

Initialise Database screen

This screen is dedicated to setting up the database. In terms of the UI, it will only display a loading wheel - when this is screen is present, everything will be processed in the background. Here is the pseudocode for setting up the database. As mentioned above, the database code will be explained in very plain english in order to achieve simplicity.

Link to success points:
The user can join a run with a 4 digit code

```
// MARK: Variables
code = ""
raceDistance = distance selected in 'Select Distance' screen

*procedure viewDidLoad()
    randomiseCode()
    setUpDatabse()
endprocedure

procedure randomiseCode()    //sets the 'code' variable to a randomly generated 4-digit code
    int1 = random integer between 0 and 9
    int2 = random integer between 0 and 9
    int3 = random integer between 0 and 9
    int4 = random integer between 0 and 9
    code = str(int1) + str(int2) + str(int3) + str(int4)    //converts the 4-integers into strings and then concatenates them into the string variable called 'code'.
endprocedure

procedure setUpDatabse()
    //the following document is created with auto-id.
    add a document to the 'Races' collection with the following fields {
        "code": code,                   //field called 'code' which is set to the randomised 4-digit code
        "distance": raceDistance,       //field called 'distance' which is set to the distance selected by the user in the previous screen.
        "player2Joined": false          //field called 'player2Joined' which is set to false
    }
    add to this document a collection called 'Players'
    add to the 'Players' collection a document called "Player1" with the following fields {
        "playerDistance": 0,            //field called 'playerDistance' which is set to 0
        "isReady":false,                //field called 'isReady' which is set to false
        "finalTime":"",                 //field called 'finalTime' which is set to an empty string
        "averageSpeed":0.0,             //field called 'averageSpeed' which is set to 0.0
        "topSpeed":0.0,                 //field called 'topSpeed' which is set to 0.0
        "speedDataArray": [0.0],        //field called 'speedDataArray' which is set to a double array containing 0.0
        "distanceDataArray": [0.0],     //field called 'distanceDataArray' which is set to a double array containing 0.0
        "timeDataArray": [0.0],         //field called 'timeDataArray' which is set to a double array containing 0.0
        "hasFinished":false             //field called 'hasFinished' which is set to false
    }
endprocedure
```

Display Code screen

This screen will essentially display the randomly generated code to the user, so that a friend can join using this code.

It will then constantly check the database to see if player 2 has joined.

When transitioning to the Race screen it is very important to make sure it is known which player the user is, player1 or player2.

```
// MARK: Outlets
Outlet: codeLabel of type Label

// MARK: Variables
code = code generated from previous screen

*procedure viewDidLoad()
    codeLabel.text = code    //Sets the labels text to the code from the 'Initialise Database' screen
    checkForJoin()
endprocedure

procedure checkForJoin ()
    add a snapshot listener to the document where the field 'code' = code      //a snapshot listener will constantly check for a change in the database
    when the field in the document called 'player2Joined' is set to true then {
        player2Joined()
    }
endprocedure

procedure player2Joined ()
    show alert saying that player 2 has joined

    transition to raceScreenController {
        //this next bit of code is needed because it labels the user as player1 and the opponent as player2
        raceScreenController.mainPlayerString = "Player1"
        raceScreenController.mainPlayerInt = 1
        raceScreenController.opponentPlayerString = "Player2"
        raceScreenController.opponentPlayerInt = 2
    }
endprocedure
```

Enter Code screen

This screen will allow the user to enter the 4-digit code to join the race. They will enter the digits through an onscreen keypad. It will then check if this code exists in the database. If it doesn't the user will be allowed to re-enter the code. If the code does exist, then the database will be updated to accommodate for player2.

```
//MARK: Variables
code = ""

*procedure viewDidLoad()
    userInputsCode()
endprocedure

procedure userInputsCode()
    //In order to simplify things, I have not included the actual code for the user entering the 4-digit code in the keypad
    allow user to input 4 digit code
    let code = users input to keypad       //The 'code' variable now stores the 4 numbers the user has typed in.
    checkCode()
endprocedure


procedure checkCode()
        Find all the documents in the "Races" collection of the database where the "code" field == code
        if the number of documents found == 0 then
            //Incorrect code
            show alert to user saying that the code is incorrect
            userInputsCode()    //Allow the user to re-enter the code
        else
            //Correct code
            show alert to user saying that the code is code
            To this document update the following field {
                "player2Joined" = true          //sets the field called 'player2Joined' to true
            }
            In this document add to the 'Players' collection a document called "Player2" with the following fields {
                "playerDistance": 0,            //field called 'playerDistance' which is set to 0
                "isReady":false,                //field called 'isReady' which is set to false
                "finalTime":"",                 //field called 'finalTime' which is set to an empty string
                "averageSpeed":0.0,             //field called 'averageSpeed' which is set to 0.0
                "topSpeed":0.0,                 //field called 'topSpeed' which is set to 0.0
                "speedDataArray": [0.0],        //field called 'speedDataArray' which is set to a double array containing 0.0
                "distanceDataArray": [0.0],     //field called 'distanceDataArray' which is set to a double array containing 0.0
                "timeDataArray": [0.0],         //field called 'timeDataArray' which is set to a double array containing 0.0
                "hasFinished":false             //field called 'hasFinished' which is set to false
            }
            transition to raceScreenController {
                //this next bit of code is again needed because it labels the user as player2 and the opponent as player1
                destinationViewController.mainPlayerString = "Player2"
                destinationViewController.mainPlayerInt = 2
                destinationViewController.opponentPlayerString = "Player1"
                destinationViewController.opponentPlayerInt = 1
            }
        endif
endprocedure
```

Race screen

This is the main screen because this is where the actual race will be carried out - it is where the main algorithms exist. It is very important therefore to break down this screen and understand how the various algorithms will combine to allow the user to compete in a running race against a friend online.

Firstly, it is very important to identify the outlets which will be used in the following pseudocode. Here are the various outlets and what they will link to:

```
// - MARK: Outlets
Outlet: mapView of type MapView
Outlet: milesOrKilometresSegment of type SegmentedControl
Outlet: speedLabel of type Label
Outlet: timerLabel of type Label
Outlet: infoLabel of type Label
Outlet: player1ProgressBar of type ProgressBar
Outlet: player2ProgressBar of type ProgressBar
Outlet: countdownLabel of type Label
```

It is also very important to establish the key variables of this screen and their use:

```
// - MARK: Variables
// - Passed from previous view controllers
mainPlayerString = ""                   //will store "Player1" if the user is player1 or "Player2" if the user is player2
mainPlayerInt = 0                       //will store 1 if the user is player1 or 2 if the user is player2
opponentPlayerString = ""               //will store the opposite of the above to variables ("Player1" or "Player2")
opponentPlayerInt = 0                   //will store the opposite of the above to variables (1 or 2)

// - For RaceData class
// For the following three variables, the 0th index/1st array in the 2d array is used to store player 1's data...
// ... and the 1st index/ 2nd array is used to store player 2's data
speedDataArray = [[], []]               //will store both user's speed over the course of the race
distanceDataArray = [[], []]            //will store both user's distance over the course of the race
timeDataArray = [[], []]                //will store both user's time over the course of the race

// For the following three variables, the 0th index in the array is used to store player 1's data...
// ... and the 1st index is used to store player 2's data
finalTimeArray = ["", ""]               //will store both user's final time
averageSpeedArray = [0.0, 0.0]          //will store both user's average speed
topSpeedArray = [0.0, 0.0]              //will store both user's top speed

// - Race data
speedInMPS = 0.0                        //will hold the user's current speed received from the GPS (in metres per second)
totalSpeed = 0.0                        //this variable will keep a total of the user's speed over the course of the race
                                        // - it is needed to calculate the distance covered
distanceCoveredInMetres = 0.0           //holds the distance ran by the user (in metres)
p1Distance = 0.0                        //holds the distance ran by player1 so that it can be used to set the player's progress bar
p2Distance = 0.0                        //holds the distance ran by player2 so that it can be used to set the player's progress bar
raceDistance = 0.0                      //stores the value of the distance of the race

// - Timer variables
countdownTimer = Timer()                //This is the timer used to display the '3', '2', '1', 'GO'.
countdownTimerCount = 0.0               //This variable holds the current time when the countdownTimer is used.
raceTimer = Timer()                     //This is the timer used to time the running race.
raceTimerCount = 0.0                    //This variable holds the current time when the raceTimer is used.

// - Timer which will constantly check if the other player has finished
waitForOtherPlayerTimer = Timer()       //This is the timer used when waiting for the other player to complete the race.
otherPlayerHasFinished = false          //This boolean is set to true when the other player has finished.
```

The first section of allowing the users to race together is actually starting the race. This includes allowing both users to confirm they are ready to start the race and then only after this starting the countdown for the race.

When the screen loads the viewDidLoad() function is called. This will read the distance of the race from the database and store it in the variable raceDistance so that it can be used later on. The user will also be prompted to press a 'ready' button. When it is pressed, it will call the readyButtonPressed() function.

```
*procedure viewDidLoad()
    raceDistance = read 'distance' from the database
    display 'ready' button and allow the user to press it. If it is pressed call the 'readyButtonPressed()' function.
endprocedure
```

This function will then update the "isReady" field in the current user's document in the database to true. The program can identify whether the user is player1 or player2 using the 'mainPlayerString' variable, which was passed from the previous screen. The program can then use the 'opponentPlayerString' variable to read the other player's document and check if their "isReady" field is set to true. If it is set to true, the race countdown can begin. If the other user has not pressed the ready button, the program will constantly re-check until the opponent is ready.

```
procedure readyButtonPressed()
    To the mainPlayerString document update the following field {
        "isReady":true          //sets the field called 'isReady' to true
    }
    Read the "isReady" in the opponentPlayerString document {
        if the "isReady" field is set to true then
            startRaceCountdown()
        else
            wait 1 second
            readyButtonPressed()
        endif
    }
endprocedure
```

The race countdown function is very straightforward. It essentially creates a timer which will call the updateCountdownTime() function every time a second passes. After 2 seconds, the 'countdownLabel' will display the number "3" then one second later "2", and so on. Finally, after 3 seconds the 'countdownLabel' will display the word "GO" and the startRaceTimer() function is called, marking the start of the race.

Link to success points:
The race will only start when both users are ready

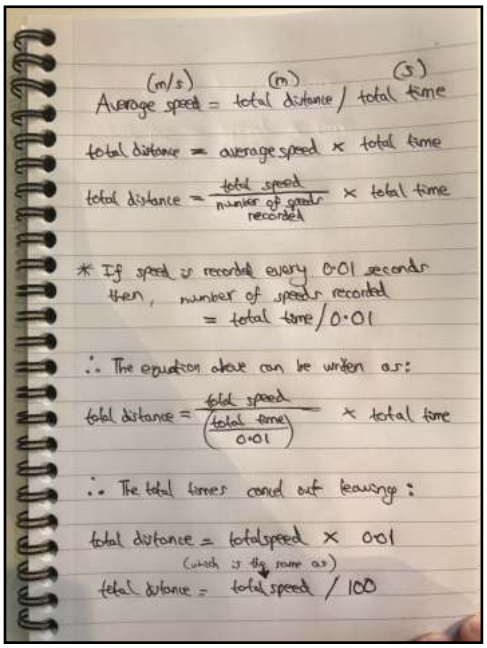
```
procedure startRaceCountdown()
    infoLabel.text = "Race Starting"
    countdownTimerCount = 0
    countdownTimer = Timer which calls the function updateCountdownTime() every 1 second
endprocedure

procedure updateCountdownTime()
    countdownTimerCount = countdownTimerCount + 1
    if countdownTimerCount == 2 then
        countdownLabel.text = "3"
        //the coundownLabel does not show 3 until the countdownTimerCount is equal to 2...
        //... because it gives the users a few seconds before the countdown starts.
    elseif countdownTimerCount == 3 then
        countdownLabel.text = "2"
    elseif countdownTimerCount == 4 then
        countdownLabel.text = "1"
    elseif countdownTimerCount == 5 then
        infoLabel.text = "Race!"
        countdownLabel.text = "GO"
        //when it has been 3 seconds the race can be start
        startRaceTimer()
    elseif countdownTimerCount == 6 then
       countdownLabel.text = ""
       stop countdownTimer
    endif
endprocedure
```

When the startRaceTimer() function is called, the raceTimer is created which will call the updateTime() function every 0.01 seconds.

```
procedure startRaceTimer()
    raceTimerCount = 0
    raceTimer = Timer which calls the function updateTime() every 0.01 seconds
endprocedure
```

The updateTime() function is the main function of the screen because it is involved in multiple calculations. Below, I have split up the function into 5 different sections. Here is what each section is responsible for:

```
procedure updateTime()
    // - Increment raceTimerCount
    raceTimerCount = raceTimerCount + 0.01                                                    (1)
    // - Update timer label
    if raceTimerCount < 10 then
        timerLabel.text = "0" + str(raceTimerCount)
    else
        timerLabel.text = str(raceTimerCount)
    endif

    // - Keeps tally of the speed in order to calculate the distance ran                      (2)
    totalSpeed = speedInMPS + totalSpeed    //speedInMPS is the speed received from GPS (metres per second)
    distanceCoveredInMetres = totalSpeed/100    //Uses distance = speed * time
                                                //The speed is in metres per second and the time is in hundredths of a second,
                                                //  therefore you all is needed is for the totalSpeed to be divided by 100
    if racetimer MOD 0.5 == 0 then                                                            (3)
        update2DArrays()
    endif

    // - MARK: Player 1s
    if mainPlayerString == "Player1" then
        p1Distance = distanceCoveredInMetres    //p1Distance holds the current distance ran by player 1
        player1ProgressBar.progress = float(p1Distance/raceDistance)    //Sets player 1's progress bar to the distance run out of the total race distance
        if raceTimerCount MOD 0.1 == 0 then
            write distance to database
            p2Distance = read player 2's distance from database
            player2ProgressBar.progress = float(p2Distance/raceDistance)    //Sets player 2's progress bar to the distance run out of the total race distance
        endif
        // PLAYER 1 HAS FINISHED
        if p1Distance >= raceDistance then                                                    (4a)
            raceFinished()
        endif
    // - MARK: Player 2
    elseif mainPlayerString == "Player2" then
        p2Distance = distanceCoveredInMetres    //p2Distance holds the current distance ran by player 2
        player2ProgressBar.progress = float(p2Distance/raceDistance)    //Sets player 2's progress bar to the distance run out of the total race distance
        if raceTimerCount MOD 0.1 == 0 then
            write distance to database
            p1Distance = read player 1's distance from database
            player1ProgressBar.progress = float(p1Distance/raceDistance)    //Sets player 1's progress bar to the distance run out of the total race distance
        endif
        // PLAYER 2 HAS FINISHED
        if p2Distance >= raceDistance then                                                    (4b)
            raceFinished()
        endif
    endif
endprocedure
```



1. The purpose of this section is to increment the raceTimerCount variable and then display the updated timerLabel.
2. The purpose of the next section is to calculate the distance ran by the user. The image on the right shows how I calculated the total distance covered to be equal to the total speed divided by 100.
3. This next section will every 0.5 seconds call the function update2DArrays(). This function (see below) will append: the current speed to the speedDataArray, the distance ran by the user to the distanceDataArray, and the current time to the timeDataArray. Again, the 'mainPlayerInt' variable is used so that the data is added to the 0th index of the 2d array if the user is player1, or the data is added to the 1st index of the 2d array if the user is player2. It is important that these arrays contain the user's speed, distance and time at every 0.5 second interval of the race so that a graph can be drawn using this data.

```
155 procedure update2DArrays()
156     speedDataArray[mainPlayerInt-1].append(speedInMPS)
157     distanceDataArray[mainPlayerInt-1].append(distanceCoveredInMetres)
158     timeDataArray[mainPlayerInt-1].append(raceTimerCount)
159 endprocedure
```

4. The purpose of section 4 is to update both player's progress bars and also check if the user has finished the race. The section is split into two parts because part a is only processed if the user is player1 and part b is processed if the user is player2. However, broadly what happens is as follows:

   Firstly, the users progress bar is updated to display the progress of the user. The progress bar can take the value of a float between 0 and 1. Therefore, its value is set to the distance ran by the user out of the total race distance.

   Then, every 0.1 seconds this distance is written to that player's document in the database. The other player's distance is read for the database so that their progress bar can be updated to show the correct distance ran.

   If the user's distance is greater than or equal to the raceDistance (they have finished the race) then the raceFinished() function is called:

Link to success points:
Shows both users progress throughout the race as a progress bar

```
procedure raceFinished()
    finalTimeArray[mainPlayerInt-1] = str(raceTimerCount)
    stop raceTimer
    calulateAverageSpeed()
    calculateTopSpeed()
    write final stats to the database
    if checkPlayerHasFinished() == false then
        add a snapshot listener which will do the following when the 'hasFinished' field in the Player2 document in the
            database is true{
            read other players final statistics from the database
            stop waitForOtherPlayerTimer
            transition screens to display the final times and statistics
        }
        waitForOtherPlayerTimer = Timer which calls the function updateTimeOtherPlayer() every 0.2 seconds
    else
        read other players final statistics from the database
        transition screens to display the final times and statistics
    endif
endprocedure
```

When the raceFinished() function is called (*shown above*), the following happens:

1. The final time is added to the user's index of the finalTimeArray.
2. The raceTimer is stopped
3. The average speed is calculated by calling the calculateAverageSpeed() function. The function works as follows:
   - Using the equation averageSpeed = total speed/number of speeds recorded
   - number of speeds recorded (count) = number of speeds recorded in the speedDataArray * 50.
   - The multiplication by 50 is necessary because in the updateTime() function above, the totalSpeed is updated every 0.01 seconds which is 50 times more frequent than the every 0.5 seconds the speedDataArray is updated.
   - The average speed is stored in the averageSpeedArray to the user's index for the user.

Link to success points: Calculates the users average speed

```
procedure calulateAverageSpeed ()
    count = speedDataArray[mainPlayerInt-1].count * 50
    averageSpeedArray[mainPlayerInt-1] = (totalSpeed / float(count)) rounded to 2.dp
endprocedure
```

4. The top speed is calculated by calling the calculateTopSpeed() function. It works as follows:
   - It gets the maximum speed from the speedDataArray @ the user's index and stores it in the topSpeedArray at the user's index.

Link to success points: Calculates the users top speed

```
procedure calculateTopSpeed ()
    topSpeedArray[mainPlayerInt-1] = speedDataArray[mainPlayerInt-1].maximumValue() rounded to 2.dp
endprocedure
```

5. After calculating the average and top speed the final statistics (average speed, top speed, final time and the 3 data arrays) are written to the correct fields of the user's document. This includes setting the "hasFinished" field to true.

6. Then the checkPlayerHasFinished() function is used in the condition of the if-statement to determine if the other player has finished the race.
    - The checkPlayerHasFinished() function will read the “hasFinished” field from the opponents document and return this Boolean.

```
function checkPlayerHasFinished ()
    otherPlayerHasFinished = read the "hasFinished" from the opponentPlayerString document
    return otherPlayerHasFinished
endfunction
```

A) If the opponent has NOT finished the race the following happens:

- A snapshot listener is added to the opponents document so that when the “hasFinished” field is set to true, the following happens:
    - the opponents final statistics are read from database (average speed, top speed, final time and the 3 data arrays)
    - the waitForOtherPlayerTimer is stopped - (the reason for this will become clear below)
    - The screen will transition to display the final times and statistics.
- After the snapshot listener is added, the waitForOtherPlayerTimer is created, which will call the updateTimeOtherPlayer() function every 0.2 seconds.
    - The updateTimeOtherPlayer() function will essentially ensure the opponent’s progress bar is still being updated, even though the race has finished for the user.
    - If the user is player1, then player2’s progress bar is updated, and vice versa.

```
procedure updateTimeOtherPlayer()
    if mainPlayerString == "Player1" then
        p2Distance = read player 2's distance from database
        player2ProgressBar.progress = float(p2Distance/raceDistance)    //Sets player 2's progress bar to the distance run out
            of the total race distance
    else
        p1Distance = read player 1's distance from database
        player1ProgressBar.progress = float(p1Distance/raceDistance)    //Sets player 1's progress bar to the distance run out
            of the total race distance
    endif
endprocedure
```

B) If the opponent has already finished the race:

- The opponents final statistics are read from the database.
- Then finally, the screen will transition to display the final times and statistics.

Link to success points: Efficient code

Note on the race screen algorithms

I would just like to talk about the purpose of the following variables:

```
// - Passed from previous view controllers
mainPlayerString = ""                   //will store "Player1" if the user is player1 or "Player2" if the user is player2
mainPlayerInt = 0                       //will store 1 if the user is player1 or 2 if the user is player2
opponentPlayerString = ""               //will store the opposite of the above to variables ("Player1" or "Player2")
opponentPlayerInt = 0                   //will store the opposite of the above to variables (1 or 2)
```

Essentially, as mentioned previously, the program needs to know whether the user is player1 or player2 for reading and writing to the database. Originally, when I was designing my algorithms, I had to use IF countless times to check whether the user is player1 or player2.

The strings (mainPlayerString and opponentPlayerString) are necessary for interacting with the database. The following code can be massively shortened.

Originally, I used an IF statement to check whether the user was player1 or player2. If the user was player1 I would write to the 'isReady' field of the 'Player1' document and then read the 'isReady' field of the 'Player2' document. I then repeated this code but for if the user is player2.
This is very inefficient because I am repeating the same database code twice in order to accommodate for the fact that the user could be player1 or player2.

```
procedure readyButtonPressed()
    if player == 1 then
        To the 'Player1' document update the following field {
            "isReady":true
        }
        Read the "isReady" in the 'Player2' document {
            if the "isReady" field is set to true then
                startRaceCountdown()
            else
                wait 1 second
                readyButtonPressed()
            endif
        }
    else
        To the 'Player2' document update the following field {
            "isReady":true
        }
        Read the "isReady" in the 'Player1' document {
           if the "isReady" field is set to true then
               startRaceCountdown()
           else
               wait 1 second
               readyButtonPressed()
           endif
        }
    endif
endprocedure
```

However, in the new code no IF statements are necessary. To explain this, let's assume that the user is player1. Therefore, the variable 'mainPlayerString' will hold 'Player1' and the variable 'opponentPlayerString' will hold 'Player2'. Therefore, to write to the user's (player1) document in the database, the mainPlayerString variable can be used and to read the other player's (player2) document, the opponentPlayerString variable can be used. Now if we assume the user is player2, the algorithm will still work. This is because the

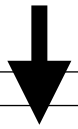
```
procedure readyButtonPressed()
    To the mainPlayerString document update the following field {
        "isReady":true          //sets the field called 'isReady' to true
    }
    Read the "isReady" in the opponentPlayerString document {
        if the "isReady" field is set to true then
            startRaceCountdown()
        else
            wait 1 second
            readyButtonPressed()
        endif
    }
endprocedure
```

variable ‘mainPlayerString’ will hold ‘Player2’ and the variable ‘opponentPlayerString’ will hold ‘Player1’.

We have just established that the string variables are useful for interacting with the database and can eliminate the need for IF statements. The use of the integers (mainPlayerInt and opponentPlayerInt) are used to access the correct index of the arrays. The following code can be again be shortened massively:

The above arrays are 2d arrays. The first array in the array is for player1 and the second array in the array is for player2.

```
155 procedure update2DArrays()
156     if player == 1 then
157         speedDataArray[0].append(speedInMPS)
158         distanceDataArray[0].append(distanceCoveredInMetres)
159         timeDataArray[0].append(raceTimerCount)
160     else
161         speedDataArray[1].append(speedInMPS)
162         distanceDataArray[1].append(distanceCoveredInMetres)
163         timeDataArray[1].append(raceTimerCount)
164     endif
165 endprocedure
```



```
155 procedure update2DArrays()
156     speedDataArray[mainPlayerInt-1].append(speedInMPS)
157     distanceDataArray[mainPlayerInt-1].append(distanceCoveredInMetres)
158     timeDataArray[mainPlayerInt-1].append(raceTimerCount)
159 endprocedure
```

Originally, I used an IF statement to check whether the user was player1 or player2. If the user was player1, I append to the 0th index of the each array the user’s specific data. I then repeated this code but for if the user is player2, appending to the 1st index of each array the user’s data. This again is inefficient because I am repeating the same code twice in order to accommodate for the fact that the user could be player1 or player2.

Now, in the new code no IF statements are used. Let’s assume that the user is player1. Therefore, the variable ‘mainPlayerString’ will hold ‘1’ as an integer. Therefore, the ‘mainPlayerInt’ subtract 1 is equal to 0; so the data will be appended to the 0th element of the 2d array. Now let’s assume that the user is player2. Therefore, the variable ‘mainPlayerString’ will hold ‘2’ as an integer and therefore, the ‘mainPlayerInt’ subtract 1 is equal to 1; so the data will be appended to the 1st element of the 2d array.

The principal is the same for the ‘opponentPlayerInt’ variable.

Race data class

Link to success points: Efficient code

In order to display the stats of both players of the race, I will use a use a static instance of a class, but it essentially is a data structure. It is very useful because during multiple points in my program I need to store and access the race data. By having this set class where it means I don't have to keep defining key names.

This will make it easier for me to access the running data from another screen, because it will be stored in one instance of the RaceData class.

```
class RaceData

    static sharedInstance = raceData()    //So that the class can be used over again.

    public speedDataArray = [[], []]     //2D arrays to hold each players speed over the course of the race
    public distanceDataArray = [[], []]    //2D arrays to hold each players distance over the course of the race
    public timeDataArray = [[], []]    //2D arrays to hold each players time over the course of the race

    public finalTimeArray = ["", ""]    //2D arrays to hold each players final time
    public averageSpeedArray = [0.0, 0.0]   //2D arrays to hold each players average speed
    public topSpeedArray = [0.0, 0.0]   //2D arrays to hold each players top speed

    public raceDistance = 0    //integer to hold the race distance
    public playerString = ""    //string to hold the current player: "player1" or "player2"
    public playerInt = 0    //integer to hold the current player: 1 or 2

endclass
```

When transitioning from the race screen to the stats screen. The following is set:

Here each field in the class is being set to the variable which temporarily stored that data during the race.

By doing this, all this data can be accessed from the stats screen.

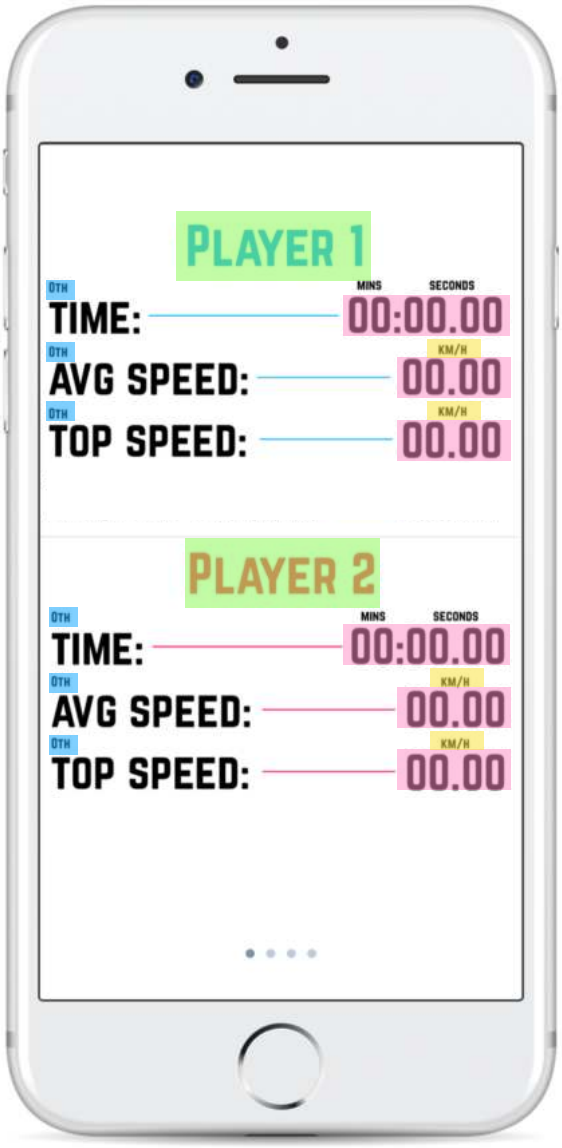
```
transition to statsScreen {
    raceData.sharedInstance.speedDataArray      = speedDataArray
    raceData.sharedInstance.distanceDataArray   = distanceDataArray
    raceData.sharedInstance.timeDataArray       = timeDataArray

    raceData.sharedInstance.finalTimeArray      = finalTimeArray
    raceData.sharedInstance.averageSpeedArray   = averageSpeedArray
    raceData.sharedInstance.topSpeedArray       = topSpeedArray

    raceData.sharedInstance.raceDistance        = int(raceDistance)
    raceData.sharedInstance.playerString        = mainPlayerString
    raceData.sharedInstance.playerInt           = mainPlayerInt
}
```

Link to success points: Displays a clear comparison between the running statistics of the two competitors

Stats screen

This screen will display the final statistics from the race. As mentioned previously, I will be using a page view to display different stats on each page. The screen is divided into 3 pages: AllStatsComparison, DistanceTimeGraph, SpeedTimeGraphComparison. Here is the pseudocode for each page:

AllStatsComparison page



Firstly, here are the outlets for this screen. When the screen is loaded the two green labels which read 'Player 1' and 'Player 2' will be set to show the respective text of 'You' or 'Opponent'. The pink labels will read the stats of the time, average speed and top speed. The yellow labels will either read 'km/h' or 'mph'. Finally, the (blue) position labels are there for comparison. They will be set to read '1st' alongside having a gold background if that stat was the best, or set to say '2nd' with a silver background.

```
// - MARK: Outlets
Outlet: player1Label of type Label
Outlet: player2Label of type Label

Outlet: p1TimeLabel of type Label
Outlet: p1AverageSpeedLabel of type Label
Outlet: p1TopSpeedLabel of type Label

Outlet: p2TimeLabel of type Label
Outlet: p2AverageSpeedLabel of type Label
Outlet: p2TopSpeedLabel of type Label

Outlet: unitIndicatorLabel1 of type Label
Outlet: unitIndicatorLabel2 of type Label
Outlet: unitIndicatorLabel3 of type Label
Outlet: unitIndicatorLabel4 of type Label

Outlet: p1TimePositionLabel of type Label
Outlet: p1AverageSpeedPositionLabel of type Label
Outlet: p1TopSpeedPositionLabel of type Label
Outlet: p2TimePositionLabel of type Label
Outlet: p2AverageSpeedPositionLabel of type Label
Outlet: p2TopSpeedPositionLabel of type Label
```

The purpose of this screen is to display the final stats of both users and compare them. Therefore, there will be no inputs and so the code can all be written from the viewDidLoad() function. However, to make my code efficient and avoid copy and pasting the same code, I will incorporate multiple functions. Essentially, the code should look like this:

```
*procedure viewDidLoad()
    if raceData.sharedInstance.playerInt == 1 then
        player1Label.text = "You"
        player2Label.text = "Opponent"
    else
        player2Label.text = "You"
        player1Label.text = "Opponent"
    endif

    p1TimeLabel.text = raceData.sharedInstance.finalTimeArray[0]
    p2TimeLabel.text = raceData.sharedInstance.finalTimeArray[1]

    p1AverageSpeedLabel.text = raceData.sharedInstance.averageSpeedArray[0]
    p1TopSpeedLabel.text = raceData.sharedInstance.topSpeedArray[0]
    p2AverageSpeedLabel.text = raceData.sharedInstance.averageSpeedArray[1]
    p2TopSpeedLabel.text = raceData.sharedInstance.topSpeedArray[1]

    highlightWinners()
endprocedure
```

Firstly, the ‘player1Label’ and ‘player2Label’ are set to appropriately distinguish which user is which. The raceData class is used here to check whether the user is player1 or player2. Next, each stat label is set to display that user’s stat from the raceData class. Finally, the highlightWinners() function is called.

```
procedure highlightWinners()
    //Time
    if Double(raceData.sharedInstance.finalTimeArray[0]) < Double(raceData.sharedInstance.finalTimeArray[1]) then
        playerWins(p1TimePositionLabel, p2TimePositionLabel)
    else
        playerWins(p2TimePositionLabel, p1TimePositionLabel)
    endif
    //AverageSpeed and TopSpeed
    compareTwoSpeeds(raceData.sharedInstance.averageSpeedArray, p1AverageSpeedPositionLabel, p2AverageSpeedPositionLabel)
    compareTwoSpeeds(raceData.sharedInstance.topSpeedArray, p1TopSpeedPositionLabel, p2TopSpeedPositionLabel)
endprocedure
```

PlayerWins function

Before understanding how the highlightWinners() function works, it is important to decompose the problem. Therefore, I have created a function called playerWins(). It takes in two Labels as inputs - the label of the player who had the superior stat and the label of the player who lost the duel of that stat. It then sets the winnerLabel background to gold and its text to read “1st” and losersLabel background to silver with its text to read “2nd”.

```
procedure playerWins(winnerLabel, loserLabel)
    winnerLabel.text = "1st"
    loserLabel.text = "2nd"
    winnerLabel background set to gold
    losersLabel background set to silver
endprocedure
```

CompareTwoSpeeds function

I needed a function which could efficiently compare two stats. The function has three parameters. The first is called ‘arr’ which will take in a speed array from the

RaceData class, e.g. ‘raceData.sharedInstance.averageSpeedArray’. The other two parameters are the position labels which will be passed into the playerWins() function (mentioned above). An IF statement is used to compare the 0th element (player1’s index) and 1st element (player2’s index) of the ‘arr’ parameter. According to which players stat was greater the playerWins function is called.

This function could not compare two final times because it compares to see if player1’s stat is greater than player2’s stat, i.e. if player1’s speed is greater than player2’s speed. Time is the opposite, with the aim being to have the lowest time. Since, I only will be comparing the two player’s times once (in comparison to comparing two different speeds), I have not created a separate function for comparing times, and I have just left the code for the HighlightWinners function.

```
66  procedure compareTwoSpeeds(arr, p1Label, p2Label)
67      if arr[0] > arr[1] then
68          //the player1's statistic is greater than player2's statistic
69          playerWins(p1Label, p2Label)      //player1's label is the 'winnerlabel'
70      else
71          //the player2's statistic is greater than player1's statistic
72          playerWins(p2Label, p1Label)      //player2's label is the 'winnerlabel'
73      endif
74  endprocedure
```

HighlightWinners function

Now finally going back to the highlightWinners() function which I mentioned above: This function is mainly for organisation purposes, i.e. this code could just be in the viewDidLoad() function. This function will first compare the times from both players and then call the playerWins() function with different arguments according to which player had the fastest and the lowest time.

This function will also call the compareTwoSpeeds() function which will compare the two speeds from both players and then highlight the appropriate position label.

Here is the highlightWinners function again just for reference. It is unchanged from the screenshot above:

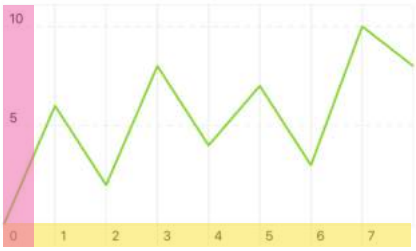
```
47  procedure highlightWinners()
48      //Time
49      if Double(raceData.sharedInstance.finalTimeArray[0]) < Double(raceData.sharedInstance.finalTimeArray[1]) then
50          playerWins(p1TimePositionLabel, p2TimePositionLabel)
51      else
52          playerWins(p2TimePositionLabel, p1TimePositionLabel)
53      endif
54      //AverageSpeed and TopSpeed
55      compareTwoSpeeds(raceData.sharedInstance.averageSpeedArray, p1AverageSpeedPositionLabel, p2AverageSpeedPositionLabel)
56      compareTwoSpeeds(raceData.sharedInstance.topSpeedArray, p1TopSpeedPositionLabel, p2TopSpeedPositionLabel)
57  endprocedure
```

Link to success points:
Correctly shows the winner of the race and each players final time

Link to success points:
Efficient code

DistanceTimeGraph page

I will be using the module 'SwiftChart' in order to draw my graph. You can add multiple series to a graph (i.e. multiple lines). The data for a series is inputted as an array of (x: Double, y: Double) tuples. For this screen I will be displaying a distance-time graph. Therefore, I need to create an array of tuples which is formatted to store the time as x and the distance as y. The values of x and y will be taken from the arrays (distanceDataArray and timeDataArray) from the raceData class.



Example SwiftChart graph

Therefore, in the race Data I have created a function called arrangeToData which has three parameters:

- player - Integer
- xArray - 2dArray of Doubles
- yArray - 2dArray of Doubles

Link to success points:
Shows a distance-time graph

A for loop is created which loops from 0 to the length of the xArray. Because all the data arrays are the same length, the length of any of the arrays can be used. In each iteration, a tuple of format (x: Double, y: Double) is added to the data array. The 'player' parameter is used to access the inputted players index of the 2dArray. Finally, the 'data' array is returned.

```
1  //Takes the two arrays of a specific palyer and formats each element into an array...
2  //... of tuples so it can be used to add the series to a graph.
3  function arrangeToData (player, xArray, yArray)
4      data = []   // of type [(x: Double, y: Double)]
5      for i=0 to lenth of xArray[player-1]
6          data.append((x: xArray[player-1][i], y: yArray[player-1][i]))
7      next i
8      return data
9  endfunction
```

This can then be utilised in the DistanceTimeGraph page.

For the player argument I have used the playerInt from the raceData class. Therefore, if the player is player1 it will get their stats from the 2d array or if the player is player2 it will get their stats. Because I would like the time to be on the x-axis and the distance to be on the y-axis, the timeDataArray is used as the xArray argument and the distanceDataArray is used as the yArray argument.

```
data = raceData.sharedInstance.arrangeToData(raceData.sharedInstance.playerInt, raceData.sharedInstance.timeDataArray,
    raceData.sharedInstance.distanceDataArray)
```

Here is the final pseudocode for the DistanceTimeGraph page:

```
// - MARK: Outlets
Outlet: chart of type SwiftChart

// - MARK: Variables
xLabelsArray = []
yLabelsArray = []

*procedure viewDidLoad()
    //*Formatting the axis*
    fillAxis()     //fills the xLabelsArray and yLabelsArray with the appropriate data for the axis.

    //Fills each axis with the labels.
    chart.xLabels = xLabelsArray
    chart.yLabels = yLabelsArray

    //To display the units of each axis
    chart.xLabelsFormatter = "s"    //adds a 's' character to the last label on the x-axis
    chart.yLabelsFormatter = "m"    //adds a 'm' character to the last label on the y-axis

    //*Formatting the data*
    data = raceData.sharedInstance.arrangeToData(raceData.sharedInstance.playerInt, raceData.sharedInstance.timeDataArray,
        raceData.sharedInstance.distanceDataArray)

    series = data as ChartSeries   //the series refers to the actual line of the graph.

    //Formats the colour of the line according to the repeated colour theme
    if raceData.sharedInstance.playerInt == 1 then
        series.color = blue
    else
        series.color = red
    endif

    chart.add(series)   //presents the series
endprocedure
```

Here is the fillAxis() function's code which is called at the beginning of the viewDidLoad():

```
procedure fillAxis ()
    //format x-axis data
    distance = raceData.sharedInstance.raceDistance
    interval = distance/5
    x = 0
    while x <= distance
        xLabelsArray.append(double(x))
        x = x + interval
    endwhile

    //format y-axis data
    time = raceData.sharedInstance.finalTimeArray[raceData.sharedInstance.playerInt-1]
    interval = time/4
    y = 0
    while y <= time
        yLabelsArray.append(double(y))
        y = y + interval
    endwhile
endprocedure
```

The function essentially fills the two arrays (xLabelsArray and yLabelsArray). Because I would like 5 labels on the x-axis, I divides the distance by 5 to get the interval. Then using a while loop it adds every increasing number from 0 to the race distance to the xLabelsArray. E.g. If the distance was 1000m, then the interval would be 200m and then the final xLabelsArray would look like: [0, 200, 400, 600, 800, 1000]. For the y-axis I would only like 4 labels, so instead the distance is divided by 4 to get the interval.

SpeedTimeGraphComparison page

The code for this screen is very similar to the previous DistanceTimeGraph page:

```
// - MARK: Outlets
Outlet: chart of type SwiftChart

// - MARK: Variables
xLabelsArray = []
yLabelsArray = []

*procedure viewDidLoad()
    //*Formatting the axis*
    fillAxis()    //fills the xLabelsArray and yLabelsArray with the appropriate data for the axis.

    //Fills each axis with the labels.
    chart.xLabels = xLabelsArray
    chart.yLabels = yLabelsArray

    //To display the units of each axis
    chart.xLabelsFormatter = "m"    //adds a 'm' character to the last label on the x-axis
    chart.yLabelsFormatter = "km/h"    //adds a 'km/h' character to the last label on the y-axis

    //*Formatting the data*
    data1 = raceData.sharedInstance.arrangeToData(1, raceData.sharedInstance.distanceDataArray,
        raceData.sharedInstance.speedDataArray)
    data2 = raceData.sharedInstance.arrangeToData(2, raceData.sharedInstance.distanceDataArray,
        raceData.sharedInstance.speedDataArray)

    //Adds a series to the chart containing player 1's data
    series1 = ChartSeries(data1)
    series1.color = blue
    chart.add(series1)

    //Adds a series containing player 2's data
    series2 = ChartSeries(data2)
    series2.color = red
    chart.add(series2)
endprocedure
```

The only difference is that two series need to be added. Therefore, I have created two arrays, data1 and data2 (in yellow). I again have used the arrangeToData function from the raceData class to format the arrays into an array of tuples. However, as the player argument I have used 1 for the data1 array and 2 for player2's data, so that each array contains that player's data. Also, as the x-axis in this graph is distance and the y-axis is speed, the next two arguments are the distanceDataArray and the speedDataArray.

Link to success points: Displays a clear comparison between the running statistics of the two competitors

Then, I can add the two series to the chart (in pink). The first series will represent player1's speed against time therefore I have used the data1 array as the data. It is also important that I make this series. The second series is player2's speed against time and I have made this line of the colour red.

Finally, the fillAxis() function is slightly different for this page. The strategy of creating the x-axis labels is the same as the way for the previous page's x-axis labels. However, the method for creating the y-axis labels (which will be speeds), is slightly different. The graph should display labels from 0 in

```
procedure fillAxis()
    //format x-axis data
    distance = raceData.sharedInstance.raceDistance
    interval = distance/5
    x = 0
    while x <= distance
        xLabelsArray.append(double(x))
        x = x + interval
    endwhile

    //format y-axis data
    maxSpeed = 0
    if raceData.sharedInstance.topSpeedArray[0] > raceData.sharedInstance.topSpeedArray[1] then
        maxSpeed = raceData.sharedInstance.topSpeedArray[0]
    else
        maxSpeed = raceData.sharedInstance.topSpeedArray[1]
    interval = maxSpeed/4
    y = 0
    while y <= maxSpeed
        yLabelsArray.append(double(y)*3.6)
        y = y + interval
    endwhile
endprocedure
```

increasing intervals up to the top speed of either player (you do not want the speed of one player going off the graph).

Therefore, using an if-statement I find the biggest top speed reached out of both the players, using the topSpeedArray from the raceData class to access both players top speed from the race. I then divide this top speed by 4 to get the interval of the y axis and then apply the same process, adding each successive incremented interval to the yLabelsArray.

I also multiply each label's value as it is added to the array by 3.6 to convert the speed from metres per second to kilometres per hour.

A. Explain and justify the structure of the solution.

Justifying the structure of the solution

My main client presented a problem to me. His solution requires:

- Users to connect online to join a run together.
- Users are able to choose the distance of the race.
- Users to be able to view how far their opponent has run during a race.
- Users can view statistics after a run, and easily compare these statistics with their opponent.

At first, this seemed liked an immense and complicated task. However, by decomposing this problem into separate stages, it will really benefit me in the development process as I will be able to more easily code each small step of my solution.

Let me just clarify how I have broken down this problem. Firstly, I broke down my main problem, of allowing two users to connect online and compete in a running race, into two separate problems:

1) Allowing two users to connect to a race.

2) Carrying out the actual race.

I then broke down both of these problems into smaller problems and then related them to the flow of the screens.

Next, with the UI prototype completed, I designed the database. By communicating with my stakeholders, I decided which stats to measure and then designed the database accordingly.

Finally, I designed the code for each screen, including the stats pages. I focused on the different variables and functions each screen would need. By breaking it down like this it ensures that the program is in separate modules so that they can communicate with one another according to the designed flow of the screens.

D. Identify key variables / data structures / classes justifying choices and any necessary validation.

Key Variables used in Race Screen:

Just to recap, these are the main variables that are used for my Race Screen:

Name	Data Type	How it is used
mainPlayerString	String	It is decided from the previous screen. IF the user came from the codeDisplayScreen then it will be set to “Player1”. IF the user came from the enterCodeScreen then it will be set to “Player2”.
mainPlayerInt	Int	Again the value of this variable is decided from the direction the user comes from. It will be set to either 1 or 2.
opponentPlayerString	String	This is the opposite of whatever the mainPlayerString is set to.
opponentPlayerInt	Int	This is the opposite of whatever the mainPlayerInt is set to.
speedDataArray	2d Array of doubles, e.g: [[13, 13,2, 13.3, 13.35], [12.1, 12.3, 12.4, 12.9]]	The first array in this 2d array will store player1’s speeds and the second array in this 2d array will store player2’s speeds. During the race, every 0.5 seconds, the users current speed is added to that players array. E.g. If the user is player2, then their current speed is appended to the second array in this 2d array.
distanceDataArray	2d Array of doubles	The structure of this variable is the same as the above 2d array. The first array in this 2d array will store player1’s distances and the second array in this 2d array will store player2’s distances. During the race, every 0.5 seconds, the distance that the user has run at that point in time is added to that players array.
timeDataArray	2d Array of doubles	Every 0.5 seconds during the race, the current time is appended to the players array (essentially it will hold a sequence of doubles which increment with an interval of 0.5). This variable is not completely necessary as it could be calculated from the final time, i.e, it isn’t a statistic which needs to be measured in realtime. However, it is easier to create the variable during the race so that it can be iterated through later when used for graphing.
finalTimeArray	Array of strings Length = 2: [“”, “”]	This array will hold both users final time. The first element in the array will hold player1’s final time and the second element in the array will hold player2’s final time.

Name	Data Type	How it is used
averageSpeedArray	Array of doubles Length = 2: [0.0, 0.0]	This array will store both user's average speed. The structure is the same as above.
topSpeedArray	Array of doubles Length = 2: [0.0, 0.0]	This array will store both user's top speed. The structure is again the same as above.
speedInMPS	Double	Will hold the user's current speed received from the GPS (in metres per second)
totalSpeed	Double	This variable will keep a total of the user's speed over the course of the race. It is needed to calculate the distance covered.
distanceCoveredInMetres	Double	Holds the distance ran by the user (in metres)
p1Distance	Double	Holds the distance ran by player1 so that it can be used to set the player's progress bar
p2Distance	Double	Holds the distance ran by player2 so that it can be used to set the player's progress bar
raceDistance	Double	Stores the value of the distance of the race

Key classes/data structures used in my solution:

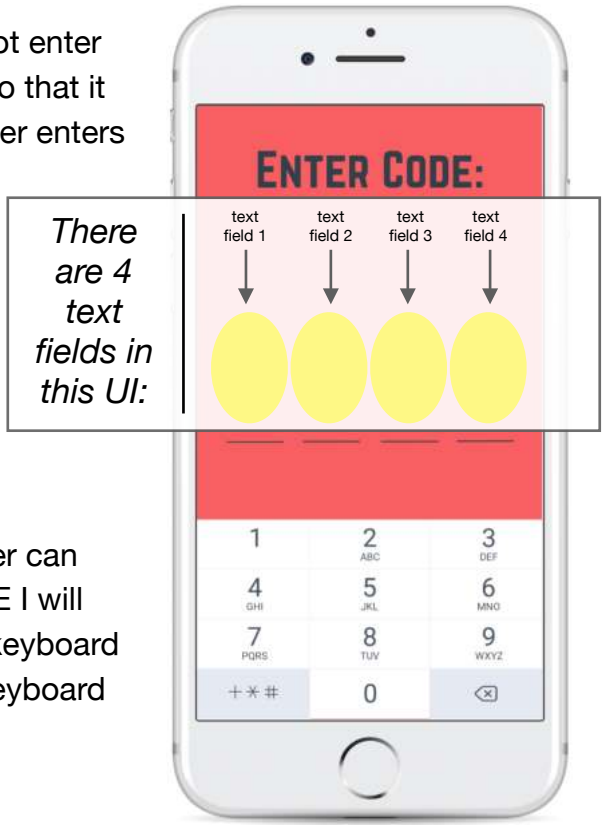
I have one main class which is used in my solution, called RaceData. As mentioned in detail previously, I have essentially created a data structure. Again, the reason for this choice is so that this class can be accessed from anywhere in my program. I can easily use this class, to access and display player1 or player2's stats such as in a graph format.

Validation

It is pivotal that I validate my solution to ensure that the user is inputting the right data and so cannot cause an error to occur in my program. In my program, there aren't many times when a user will need to enter data. The following are the two different ways users can input data:

- Buttons
    - There are multiple buttons in my solution. They are useful because they allow the user to make a choice. For example, on the home screen to allow the user to choose whether they would like to create a run or join a run. There is no validation needed as all the code for a button is handled by Xcode, as it is a pre-coded object.
- Text fields

- Text fields are only used once in my solution. This is in the Enter Code Screen to allow the user to type the 4 digit code.
- The first validation is to ensure that the user cannot enter more than 4 digits. Therefore, I will create the UI so that it will comprise of 4 individual text fields. When a user enters a number it will automatically move on to the next text field. This is repeated until a digit has been entered in the last text field. If the race with that 4 digit code is found in the database then it transitions to the Race Screen. However, if the 4 digit code is not found, then all 4 text fields are cleared and the user can start from the first text field entering the 4 numbers.
- The second validation is to make sure that the user can only enter digits. Therefore, in using the Xcode IDE I will change the settings of each text field so that the keyboard which pops up when they are clicked is a digits keyboard (as shown), as opposed to a regular keyboard.



Enter Code Screen

3.2.3 Describe the approach to testing

A. Identify the test data to be used during the iterative development and post development phases and justify the choice of this test data.

4 Testing methods:

The problem that my client has presented to me is a very unique problem because it requires the use of GPS to track the user of the app. The most significant problem which I broke down in my algorithms design is the actual race and therefore this feature needs to be tested extensively. It is important that I test how different factors affect the user's connection and GPS, such as: the altitude of runners, running under a bridge, using 4g vs 3g mobile data.

In order for my stakeholder to be satisfied with the product, it is important that it is tested extensively. I have devised a test plan which I will follow. Here are the four tests I plan on carrying out during the development and evaluation stages:

White Box Testing

- During the development phase, I will be carrying out various white box tests to inspect and adapt the internals of my functions. It is important to help me understand the flow of specific inputs through the code and their expected output. It will also help me optimise my code by finding hidden errors so by the time I finish the iterative development, because I know all code paths have been covered.

Black Box Testing

- During the evaluation phase I will follow the extensive test plan which I have created below. It is important to carry out black box testing as well as white box testing, to ensure that not just the logic of the code works, but the program fulfils the stakeholders requirements. I will carry out normal and boundary tests to ensure the buttons, flow and main purpose of the app works as planned. During the evaluation phase I will also carry out destructive testing. It is crucial, from a user's perspective, that my app does not crash. I will again follow a plan, in the form of a table, to test this.

Beta Testing

- Finally, during the evaluation phase I will download my app onto the phones of my two stakeholders and I will give them a few days to test the app. They will then fill out an evaluation form to provide me with feedback about how I may improve my solution.

Black-box test plans

Normal tests

Test ID	What am I testing	What data will I use? / what actions will I take?	Expected outcome
1	CreateRace button works on home screen	I will press the race button	It will transition to the selectDistance screen
2	JoinRace button works on home screen	I will press the join button	It will transition to the enterCode screen
3	User can select a distance from the scroll wheel on the selectDistance screen	I will select a distance from the scroll wheel	This distance will be selected
4	Next button works on SelectDistance screen	I will press the next button	It will transition to the initialiseDatabase screen
5	The users location is updated on the map when they move on Race screen	I will go outside and start walking	The location on the map moves
6	To see if the speed label on the Race screen is accurate	I will go outside and start running. I will compare the speed on my app with a speedometer.	The speeds are the same
7	The speed label on the Race screen is correct in kilometres units	I will go outside and start running. I will compare the speed on my app with a speedometer which is set to kilometres per hour.	The speeds are the same
8	The speed label on the Race screen is correct in miles units	I will go outside and start running. I will compare the speed on my app with a speedometer which is set to miles per hour.	The speeds are the same
9	Testing to see if the progress bar correctly shows the progress of each user throughout the race	I will go to a local 400m running track. I will create a 200m race. I will then run up to the 100m mark.	The progress bar should be at half way.
10	The race stops when both players finish the race	I will create a race with my brother, however, I will walk whilst my brother runs.	When I finish the race, it should stop and transition to the Stats screen.
11	The final time is accurate	I will go to a local 400m running track. I will create a 400m race and walk the race whilst holding a stopwatch in my other hand. By walking it reduces the effect of human error. I will then compare the time on the stopwatch to the time on the app.	The times should be the same. Ideally I would like the times to be no more than 0.5 seconds apart.

Test ID	What am I testing	What data will I use? / what actions will I take?	Expected outcome
12	The average speed is accurate	I will walk a 100m race whilst also using a speedometer on another phone. I will record my speed from the speedometer at 10 constant intervals throughout the race (every 10m). I will then calculate the average speed by summing these values and then dividing by 10.	My calculated average speed should be the same as the average speed given on the app. I would like the average speeds to be no more than 0.5 kilometres per hour apart.
13	The top speed is accurate	I will walk a 100m race whilst also using a speedometer on another phone. In the middle of the race I will run and record my peak speed.	My recorded top speed should be the same as the app's top speed.
14	The shape of the distance-time graph is accurate	I will walk at varying speeds a 100m race. Before the race I will mark out 10m intervals. During the race I will record the time when I pass an interval. I will then plot a line graph using Excel and compare the two graphs.	The graphs should look similar.

Boundary tests

Test ID	What am I testing	What data will I use? / what actions will I take?	Expected outcome
1	User selects shortest distance race on the selectDistance screen	Select the 50m option	A race is created of distance 50m
2	User selects longest distance race on the selectDistance screen	Select the 10,000m option	A race is created of distance 10,000m
3	If text fields on enterCode screen allow you to enter more than 4 digits	Enter 5 digits: e.g. 1, 2, 3, 4, 5	It should reset after 4 digits are inputted, and then the last digit is inputted to the first text field
4	A 4 digit code is created which already exists in the database	Temporarily add a line of code which sets the code to 4 digits which already exist in the database.	Should randomise a new code.
5	What happens if both players finish the race at the exact same time	I will go outside with mine and my brothers phone. I will create a 100m race and walk the distance whilst holding both phones next to side by side.	It should randomly give the win to one of the players according to who was ahead at that point according to the GPS.

Extreme testing

Test ID	What am I testing	What data will I use? / what actions will I take?	Expected outcome
1	Race starts with one player's phone on airplane mode (no mobile data in use)	Put one user's phone into airplane mode	Notifies the user that they have lost connection
2	Race starts with both players phone on airplane mode (no mobile data in use)	Put both user's phones into airplane mode	Notifies both users that they have lost connection
3	A player runs under a bridge/tunnel	I will run under a local bridge to simulate this	Connection may be lost temporarily but it should not affect the GPS signal completely. The connection to the database may also be lost temporarily but then the running statistics will be uploaded once connection is restored.
4	A player turns off their phone during a race	I will turn my phone off during a race	Race should still continue in the background
5	A player powers off their phone during a race	I will power off my phone during a race	I will be disconnected from the race
6	Player loses GPS signal during the race	This is very hard to recreate myself. The best, I can do is ask all the volunteers whether this happened to them during the race.	Notify the user that they have lost GPS connection and stop race.
7	A player tries racing indoors	I will race my brother whilst I stay indoors	Due to weak GPS it may not update my location. So I will not be able to complete the race.

Above are just a few scenarios which I could think of occurring during use of the app. When testing I very likely to run into/think of a new scenario to test and so I will include these in the future test table in the testing phase.

3.3

Development

3.3.1 Decompose the problem

A. Provide annotated evidence of each stage of the iterative development process justifying any decision made.
B. Provide annotated evidence of prototype solutions justifying any decision made.

Plan for development

- Version 1 - Creating the home screen
- Version 2 - Building the select distance screen
- Version 3 - Building the initialise database screen
- Version 4 - Building the display code screen
- Version 5 - Building the enter code screen
- Version 6 - Creating the race code screen
- Version 7 - Stats screen

Each Version will be broken down further into point versions (e.g. Version 1.1).

Note on how Xcode/Swift development works

Using Xcode you design the user interface using a drag and drop interface, in a file called the Main Storyboard.

Then to control how each UI elements can be used or manipulated, a file is created for each screen (having a 1-to-1 relationship), and then UI elements are linked to the Swift (the programming language) file.

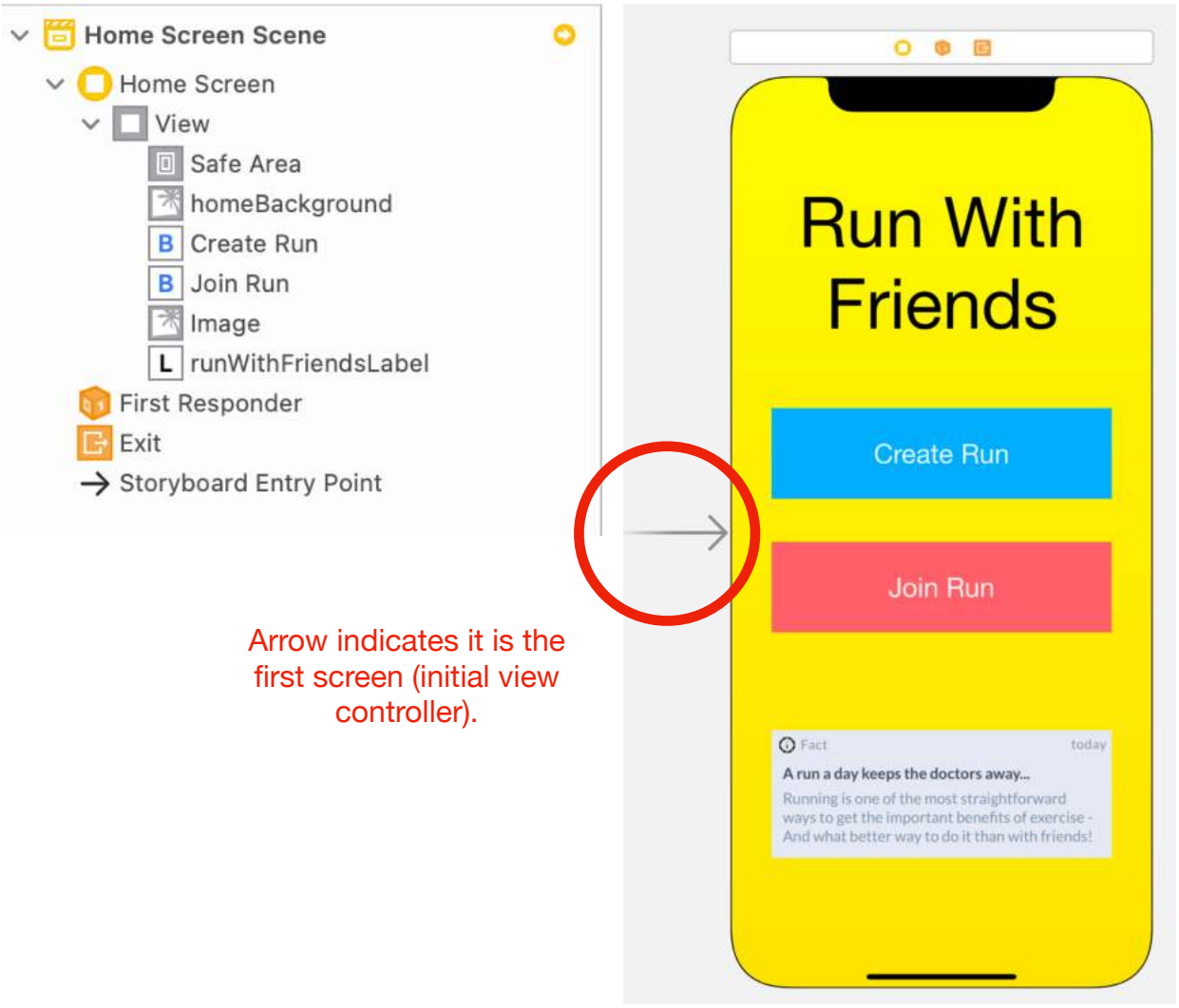
My app has 7 screens. Therefore, I will have 7 screens in my Main Storyboard file and then 7 Swift files.

In Swift, elements are linked to their corresponding files via a reference called a UIOutlet. From this variable, attributes of the UI element can be changed. Additionally, if you want an action to take place as a result of interacting with a UI element (such as pressing a button), a UI element needs to be linked to its corresponding file via a UIEvent function.

Version 1 - Creating the home screen

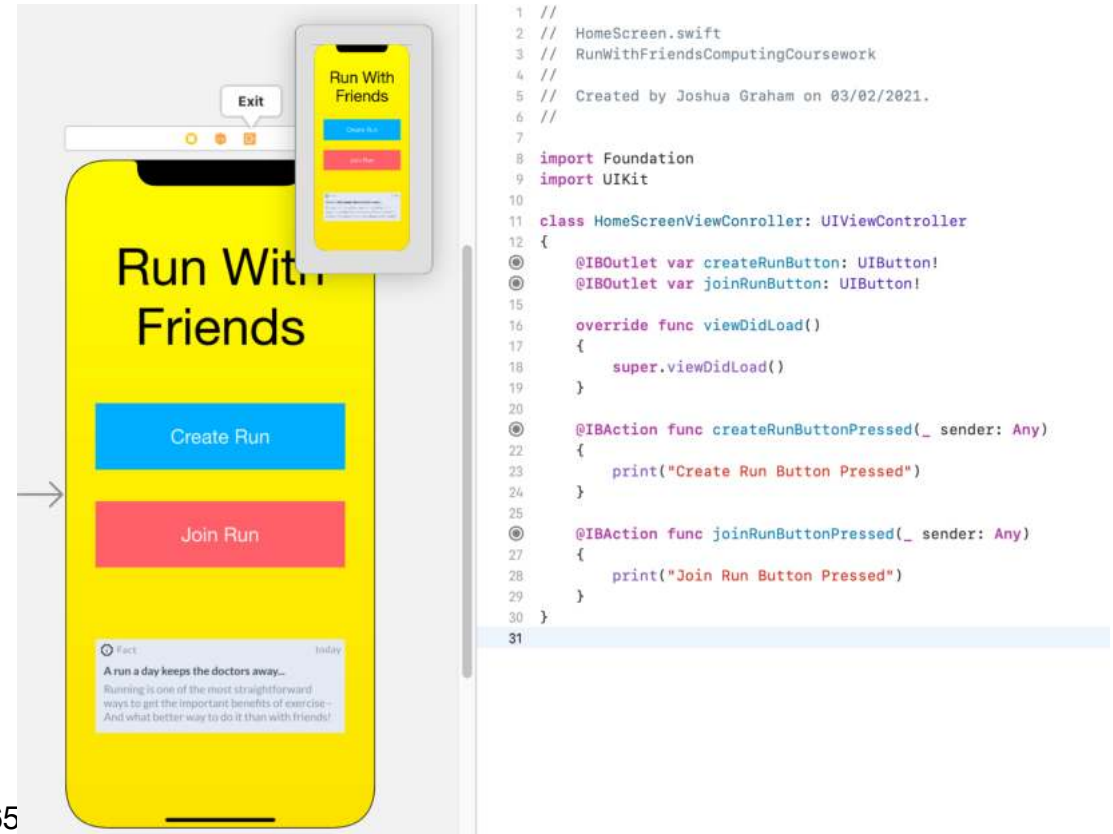
Version 1.1 - Creating the UI on main storyboard

Using the drag and drop feature on Xcode, I put together a UI screen which follows the same design as my prototype design:



Arrow indicates it is the first screen (initial view controller).

Version 1.2 - Connecting the UI to the swift view controller class




```swift
//
//  HomeScreen.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 03/02/2021.
//

import Foundation
import UIKit

class HomeScreenViewConroller: UIViewController
{
    @IBOutlet var createRunButton: UIButton!
    @IBOutlet var joinRunButton: UIButton!

    override func viewDidLoad()
    {
        super.viewDidLoad()
    }

    @IBAction func createRunButtonPressed(_ sender: Any)
    {
        print("Create Run Button Pressed")
    }

    @IBAction func joinRunButtonPressed(_ sender: Any)
    {
        print("Join Run Button Pressed")
    }
}

```

I first created a new swift file called HomeScreenViewController. Then, using the help of the assistant layout view, I connected the UI elements to the file, by creating the necessary references. As you can see above, I created two 'Action' functions for when each button is pressed. For now I have just written a print statement, but when I have made the next few screens, I will be able to program the transitions to these screens in these two functions.

I have also created two 'Outlets', one for each button in the screen. This is because I would like to give the buttons rounded edges instead of sharp edges so that my user interface looks more modern. This can only be done programmatically.

I have also added the viewDidLoad function, with its required line of code 'super.viewDidLoad()'. It is in this function where I will change the appearance of the buttons corners (because this function is called before the screen appears).

Both the 'Run With Friends' label and the image of the 'iOS notification' have not been referenced/connected to the swift file, because they both do not need to be changed or interacted with in any way.

Version 1.3 - Button corners

I added the following code into the 'viewDidLoad' function. As you can see I have used the 'Outlet' variables as references.

```
16      override func viewDidLoad()
17      {
18          super.viewDidLoad()
19
20          createRunButton.layer.cornerRadius = 6
21          joinRunButton.layer.cornerRadius = 6
22      }
```

Version 1.4 - Testing

I ran my code using an iPhone simulator (an Xcode built-in feature). I was happy with the appearance as it looks very similar to my prototype (so I know my stakeholders will be satisfied with the home screen design).

Furthermore, I pressed both the buttons a few times to test that the action is are linked properly. This was successful.

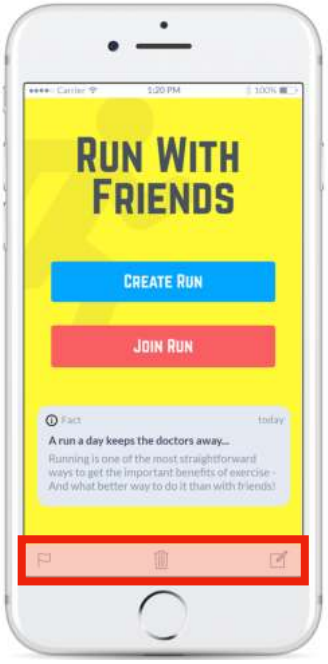
At this point, the home screen is finished. As I have mentioned, I can only program the screen transition for when a button is pressed, later in my development, when I have created the screens for which the buttons will transition to.

```
Create Run Button Pressed
Join Run Button Pressed
Create Run Button Pressed
Create Run Button Pressed
Join Run Button Pressed
Join Run Button Pressed
Create Run Button Pressed
Join Run Button Pressed
```

Version 1 - Review

What has been done

I have created the home screen which looks almost identical to the prototyped design. It contains two buttons. Currently, when the buttons are pressed, they do not transition to the next screen. I will add this when I create the screens which they transition to.



HomeScreen from prototype

What has changed from the original design

It doesn't contain the toolbar (highlighted) which was in the original design. This is because it is just an unnecessary part of my design.

What has been tested

I have tested that both buttons work.

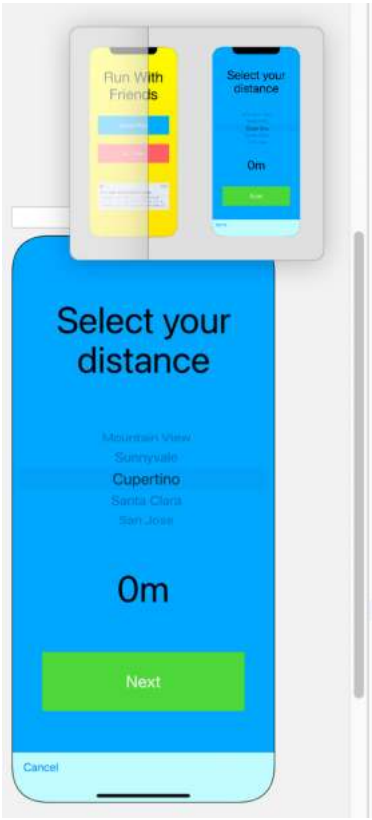
Version 2 - Building the select distance screen

Version 2.1 - Creating the UI on main storyboard

If again first created a UI screen on main storyboard file. It again follows my prototype very closely.

Version 2.2 - Connecting the UI to the swift view controller class

Again, I created a new swift file, called selectDistanceViewController. I added the Outlet references for the pickerView, button, and the pickerLabel. I also added an Action function for when the button is pressed (which is empty for now).




```
//
//  selectDistanceViewController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 05/02/2021.
//

import Foundation
import UIKit

class selectDistanceViewController: UIViewController
{
    @IBOutlet var pickerView: UIPickerView!
    @IBOutlet var pickerLabel: UILabel!
    @IBOutlet var nextButton: UIButton!

    override func viewDidLoad()
    {
        super.viewDidLoad()

        nextButton.layer.cornerRadius = 7
    }

    @IBAction func nextButtonPressed(_ sender: Any)
    {
        print("next button pressed")
    }


}
```

Version 2.3 - Configuring the Picker View

To tell my class that it is in charge of controlling the PickerView, it needs to inherit from the following methods:

```
class selectDistanceViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource
```

And I need to add this code to the viewDidLoad function:

```
override func viewDidLoad()
{
    super.viewDidLoad()

    pickerView.delegate = self
    pickerView.dataSource = self

    nextButton.layer.cornerRadius = 7
}
```

Next I created an array containing the distances to display in the picker. The other variable selectedDistance will be set to the item the user selects from the pickerView.

You may have noticed a difference in the way I initialised these two variables. The 'let' keyword means that the value of the variable can never change after creation (immutable) and is therefore more memory efficient. The 'var' keyword means that the variable can change after creation (mutable).

```
let distances = ["50m","100m","200m","400m","800m","1000m","2000m","5000m","10000m"]
var selectedDistance = ""
```

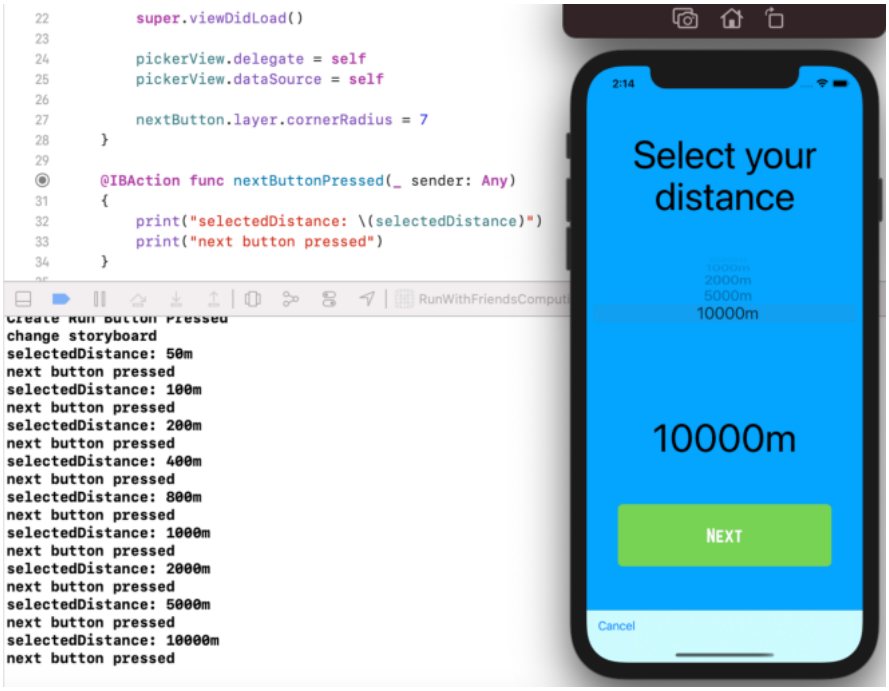
I then added the following functions:

```
func numberOfComponents(in pickerView: UIPickerView) -> Int
{
    //'components' = columns
    //there is 1 column in the picker view
    return 1
}

func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int
{
    //the number of rows in the picker view is set to the length of the distances array
    return distances.count
}

func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String?
{
    //'row' is the user selected item in the picker view.
    selectedDistance = distances[row]
    pickerLabel.text = selectedDistance

    //returns the string in the distance array which the
    return distances[row]
}
```

As you can see, in the final function I have set the selectDistance variable to the item selected in the picker view. I then set the pickerLabel text to this value.

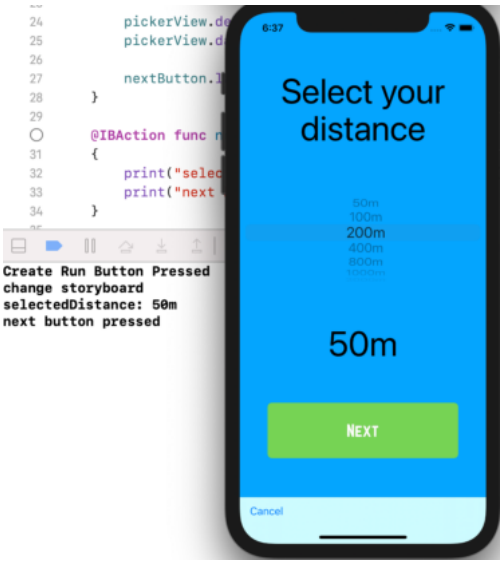
I then proceeded to test that my picker view works.

```
@IBAction func nextButtonPressed(_ sender: Any)
{
    print("selectedDistance: \(selectedDistance)")
    print("next button pressed")
}
```

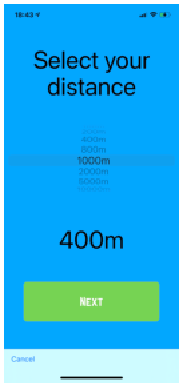


As you can see, I tested each selection, all of which were successful.

However, I found the picker wheel to be very finicky and difficult to use. For example, when I scrolled the wheel to 200m, the picker label showed 50m and this also showed in the console when I pressed the next button. Then when I moved the picker ever so slightly, the picker label and the 200m distance matched.



I first thought this was problem was occurring because I was using the iPhone simulator so maybe scrolling with my trackpad my cause the scroll view to lose calibration. So I installed the app onto my phone and tested the scroll wheel again but I ran into the same problem.



Screenshot from my phone

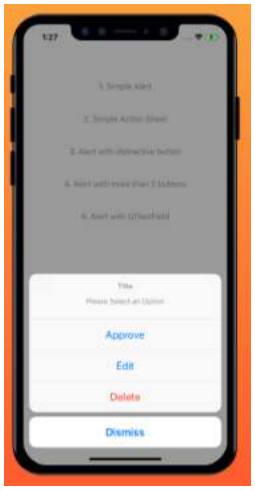
It clearly must be a built in flaw with the picker view Swift object.

One of my aims is to make my app as easy as possible to use. Therefore, I am going to find a new way implement the selection of the race distance. But firstly, it is important that I let my

stakeholders know that I am making this change to get their inputs. I downloaded the current version of the app onto their phones and they both agreed that I should replace the scroll wheel.

Version 2.4 - Designing the new user selection UI

I have decided that the best option to implement would be an action view.



On the right is an example of an alert view. I will have all the distances instead of the shown ‘approve’, ‘edit’ and ‘delete’. The bottom ‘dismiss’ button will show cancel and will bring the user back to the home screen.

The advantage of using an action view is that it is very easy to view all the available race options and then select an option.

Marvel prototype

I then created a prototype design of the screen on marvel.com (the same website which I used to create my prototype designs back in the design section). Obviously, when I program it, all the available distances will be available to be selected.

Version 2.5 - Programming the Action View

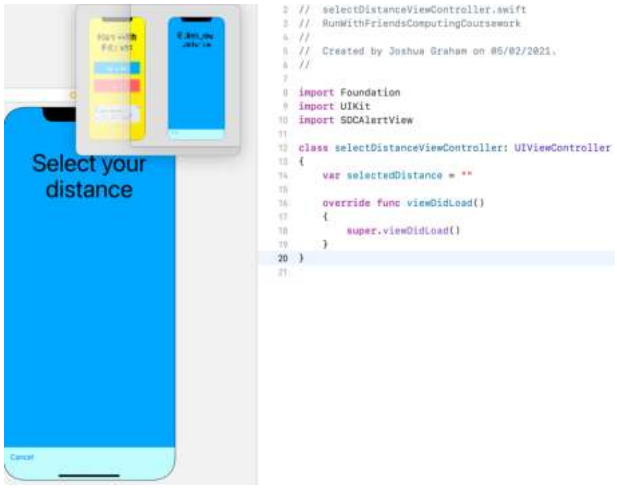
I will be using a 3rd party library from GitHub called SDCAlertView which is very similar to the inbuilt Swift UIAlertView, however, it is more customisable.

After using the terminal to install the library, I was ready to start programming the action view.

I firstly deleted the picker view, the picker label and the next button from the MainStoryboard UI design and all its related code from the corresponding swift file.

I also imported the new library ‘SDCAlertView’.

My Swift file and storyboard file now looks as follows:



Here is the code I added for the actionView.

```
override func viewDidAppear(_ animated: Bool)
{
    let alert = AlertController(title: "Please select your selectedDistance", message: "", preferredStyle: .actionSheet)
    alert.addAction(AlertAction(title: "Cancel", style: .destructive) { (action) in
        self.dismiss(animated: true, completion: nil)
        })
    alert.addAction(AlertAction(title: "100m", style: .normal) { (action) in
        self.selectedDistance = 100
        self.changeStoryboard()
    })
    alert.addAction(AlertAction(title: "200m", style: .normal) { (action) in
        self.selectedDistance = 200
        self.changeStoryboard()
    })
    alert.addAction(AlertAction(title: "400m", style: .normal) { (action) in
        self.selectedDistance = 400
        self.changeStoryboard()
    })
    alert.addAction(AlertAction(title: "1000m", style: .normal) { (action) in
        self.selectedDistance = 1000
        self.changeStoryboard()
    })
    alert.addAction(AlertAction(title: "2000m", style: .normal) { (action) in
        self.selectedDistance = 2000
        self.changeStoryboard()
    })
    alert.addAction(AlertAction(title: "5000m", style: .normal) { (action) in
        self.selectedDistance = 5000
        self.changeStoryboard()
    })
    alert.addAction(AlertAction(title: "10000m", style: .normal) { (action) in
        self.selectedDistance = 10000
        self.changeStoryboard()
    })

    alert.present()
}

func changeStoryboard()
{
    print("change screen")
}
```

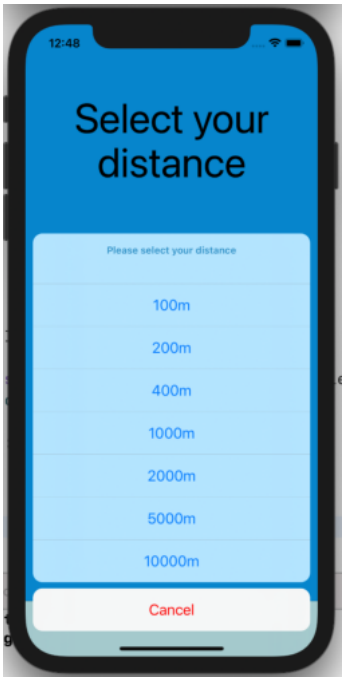
Essentially, when a button is pressed, the selectedDistance variable is set to that value and the app will transition to the next screen. If the cancel button is pressed, the actionView is dismissed and the user is returned to the home screen.

Version 2.6 - Testing

Before I can start testing I need to be able to transition to this ‘selectDistanceScreen’ from the homeScreen. Therefore, I have added this line to the homeScreen swift file, so that when the create run button is pressed this function is called:

```
func changeStoryboard()
{
    let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
    print("change storyboard")
    if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier:
        "selectDistanceViewController") as? selectDistanceViewController
    {
        self.present(destinationViewController, animated: false, completion: nil)
    }
}
```

I tested that the transition was working and all was successful. As soon as the new screen is appears the action pops up from the bottom of the screen.



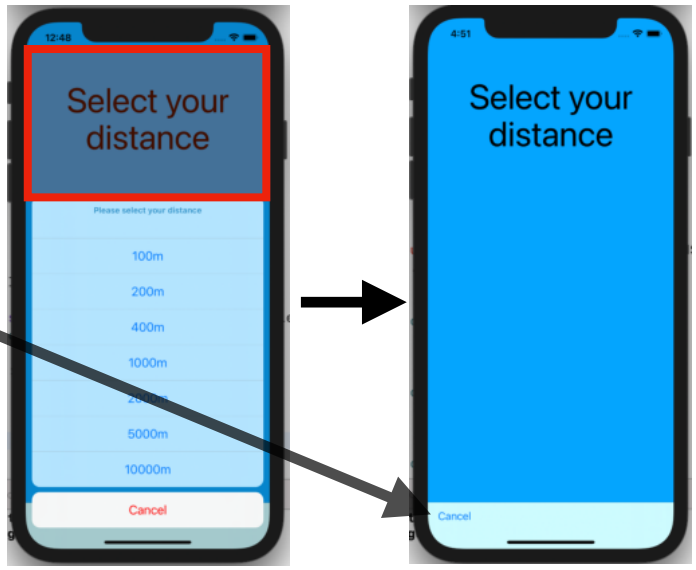
selectDistance screen as soon as it is opened

To test the action view was fully sound, I added the following line to the selectDistance screen 'changeStoryboard' function:

```
func changeStoryboard()
{
    print("change screen")
    print("selectedDistance: \(selectedDistance)")
}
```

Everything worked as intended. Personally, I found this implementation much easier to use compared to the picker. I again downloaded the updated version of the app onto my stakeholders phones and they both agreed that this implementation is an improvement.

Version 2.7 - Tapping outside the actionView area



IF user presses in highlighted area

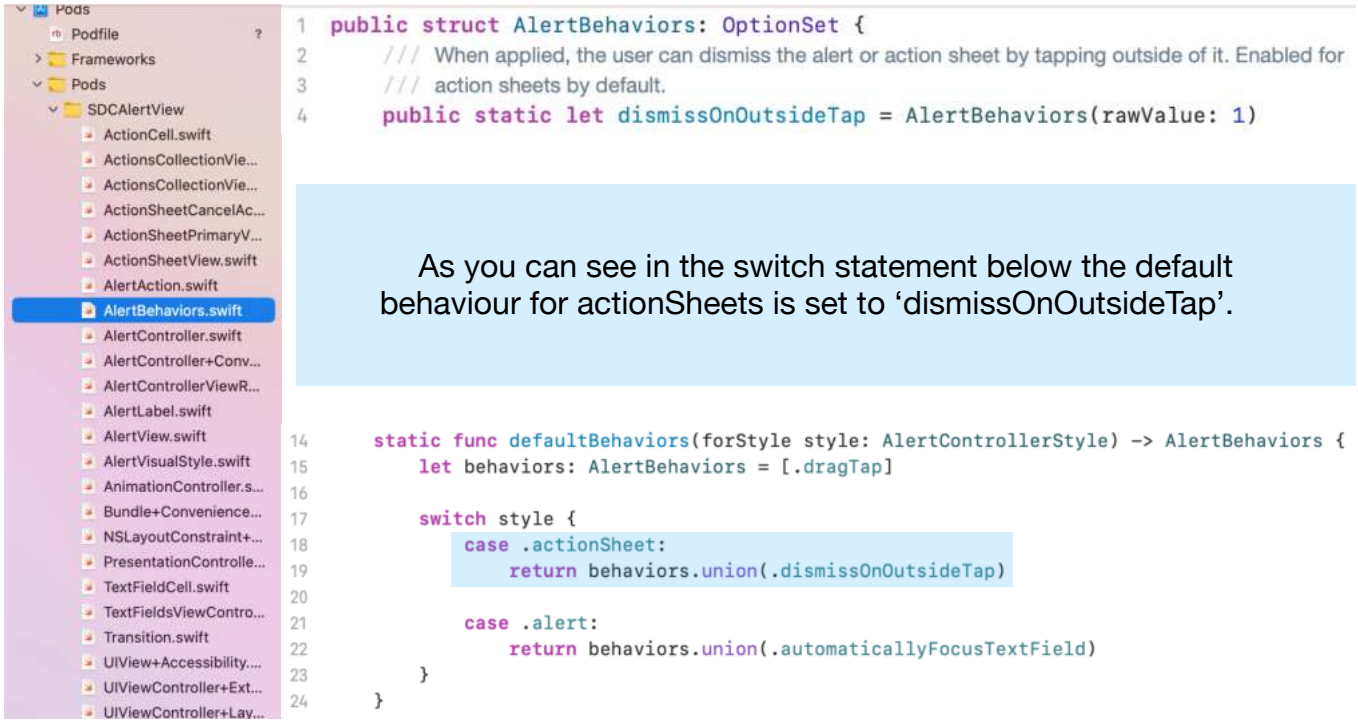
The ActionView hides

If the user presses in the area highlighted in the image on the right, the actionView is dismissed and the app is left in a permanent state (the user cannot interact or change screens). It is important to note that at this point I have't setup the 'cancel' button which is in the tool bar.

I have two options, either to configure the cancel button so that when it is pressed, it transitions back to the home screen. However, this would be a little unnecessary, because at this point (if the actionView is dismissed), the user's only choice is to press the cancel button and go back to the home screen. Therefore, it's just an extra unnecessary step to go back to the home screen.

If the user wanted to return to the home screen, they should press the cancel button from the alert view. Therefore, the second option is to prevent the user from being able to press in the red highlighted area above the alertView.

To do this I found the attribute in the SDCAlertView library which is responsible for dismissing the alert view when the user presses outside of it. I could see that it was enabled by default.



I therefore changed that line of code so the default behaviour was set to 'automaticallyFocusTextField'.

```
switch style {
    case .actionSheet:
        return behaviors.union(.automaticallyFocusTextField)
```

Now when testing the app, it is impossible to dismiss the actionView by tapping outside it. Therefore the user must either select a distance for the race, or press the cancel button to return to the home screen.

I can now delete the toolbar from the UI design in the MainStoryboard file (including the cancel button)

Version 2 - Review

What has been done

I have created the select distance screen. The user can select the distance for the race from an actionView. I have also created the transition from the homeScreen to this selectDistanceScreen. I have also perfected (in terms of functionality and convenience) for the way the user can return to the home screen. I did this by not allowing the user to press outside the actionView to dismiss it so they have to select an option from the actionView and they cannot accidentally dismiss it.

What has changed from the original design

In the original design I used a pickerView. The stakeholders and I agreed that it wasn't the optimum way of allowing the user to select a distance due to its inconsistencies. Therefore, I have used an actionView which is a lot simpler and in the end easier to use.

What has been tested

I originally tested the pickerView. I have also tested every that every option in the actionView gives the correct distance and I tested what happens if the user presses outside the area of the actionView.

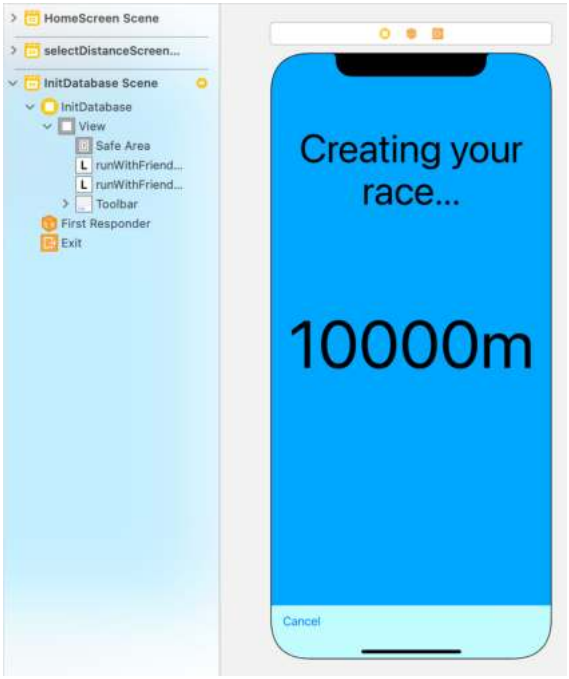
Version 3 - Building the initialise database screen

There will be no interaction between the user and this screen. Therefore, the user interface is not the important part of this screen. What is important, is what happens in the background (initialising the database).

Version 3.1 - Creating the UI on main storyboard

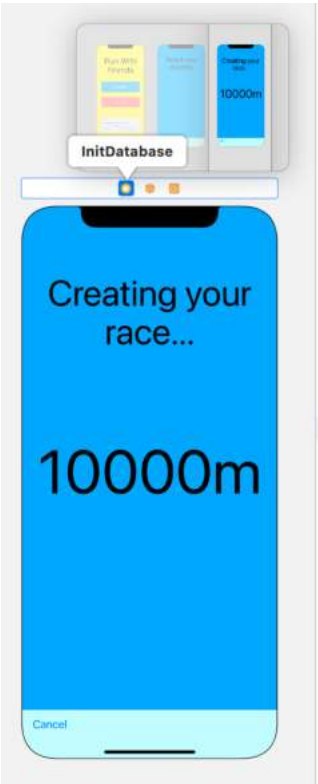
As you can see my UI interface has changed slightly from the one I designed originally. Instead of having just showing a loading wheel, I have rather decided it would be more useful to inform the user what is happening in the background (creating the race for them) and also to show them the distance of the race they selected.

Initialise database screen from initial design



Version 3.2 - Connecting the UI to the swift view controller class

This screen is very simple to set up. I created a new swift file, called InitDatabaseScreenController. I then added the Outlet references for the distanceLabel.




```swift
//
//  InitDatabaseScreenController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 06/03/2021.
//

import Foundation
import UIKit

class InitDatabaseScreenController: UIViewController
{
    @IBOutlet weak var distanceLabel: UILabel!

    override func viewDidLoad()
    {
        super.viewDidLoad()
    }
}
```

Version 3.3 - Transitioning to the screen

I added the transition from the selectDistanceScreen to this screen by adding this code to the changeStoryboard function of the selectDistanceScreen swift file.

```
60      //transitions from the SelectDistance screen to the InitDatabase screen
61      func changeStoryboard()
62      {
63          let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
64          print("change storyboard")
65          if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier: "InitDatabaseScreenController") as?
                InitDatabaseScreenController
66          {
67              self.present(destinationViewController, animated: false, completion: nil)
68          }
69      }
70  }
```

Version 3.4 - Passing over the race distance variable

When the app transitions to this screen, the distanceLabel should show the distance which the user selected from the previous screen (currently it is set by default to show 10000m). Therefore, I need a way to be able to access the value of the selectedDistance variable in the selectDistanceViewController class (highlighted below) from this current class.

```
12  class SelectDistanceViewController: UIViewController
13  {
14      var selectedDistance = 0
```

In Swift it is not as simple as simply writing *SelectDistanceViewController.selectedDistance*. You have to pass the value of the variable into another variable when transitioning ViewControllers.

Therefore, what I did was create a variable (which is an Integer) in the InitDatabaseScreenController class called distanceObject.

```
var distanceObject: Int?
```

(Quick note: the question mark means that this variable is an option variable - it can either have a value or it can be empty).

Then, I added this highlighted line to the changeStoryboard function in the selectDistanceScreen which sets the distanceObject variable to the selectedDistance variable:

```
60      //transitions from the SelectDistance screen to the InitDatabase screen
61      func changeStoryboard()
62      {
63          let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
64          print("change storyboard")
65          if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier: "InitDatabaseScreenController") as?
                InitDatabaseScreenController
66          {
67              destinationViewController.distanceObject = selectedDistance //passes the selectedDistance value to the next screen
68              self.present(destinationViewController, animated: false, completion: nil)
69          }
70      }
71  }
```

Then in the viewDidLoad function of the InitDatabaseScreenController class I set the distanceLabel to read the value of the distanceObject variable.

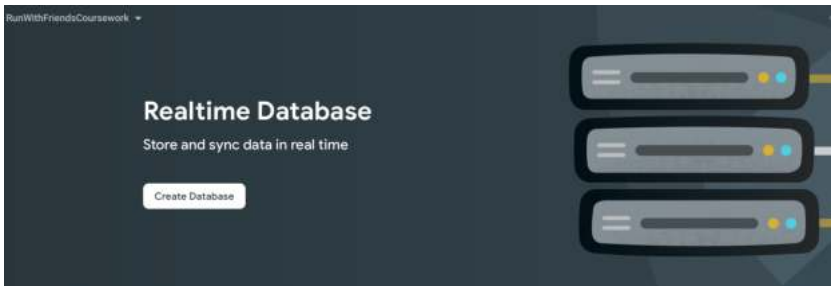
After testing, the race distance was passed to this viewController and it successfully showed the distance the user selected.

```
override func viewDidLoad()
{
    super.viewDidLoad()

    distanceLabel.text = String(distanceObject!) + "m"
}
```

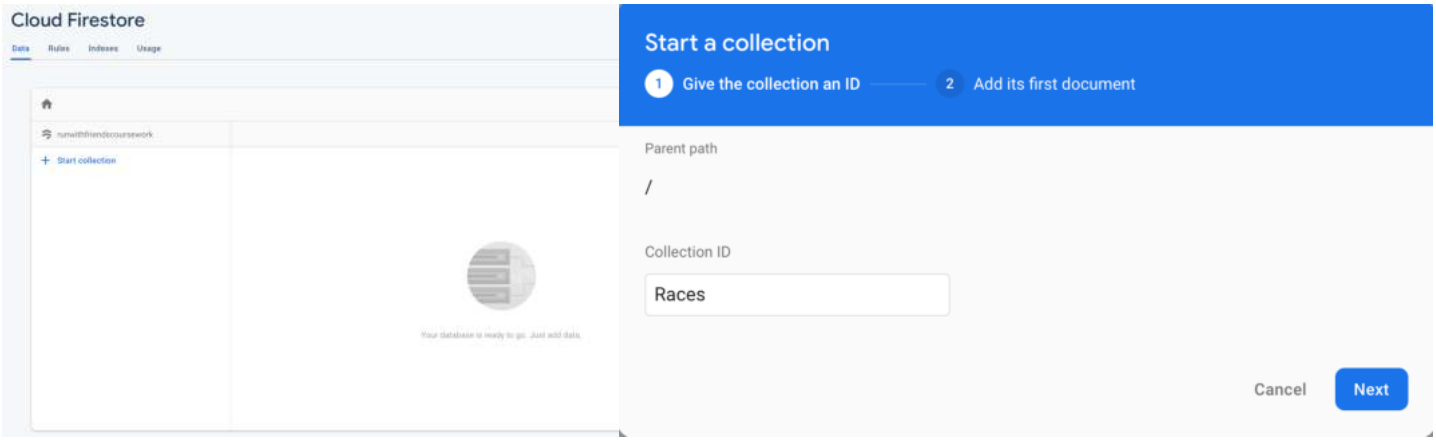
Version 3.5 - Setting up the database

First things first, I need to install the Google Firebase SDK libraries.

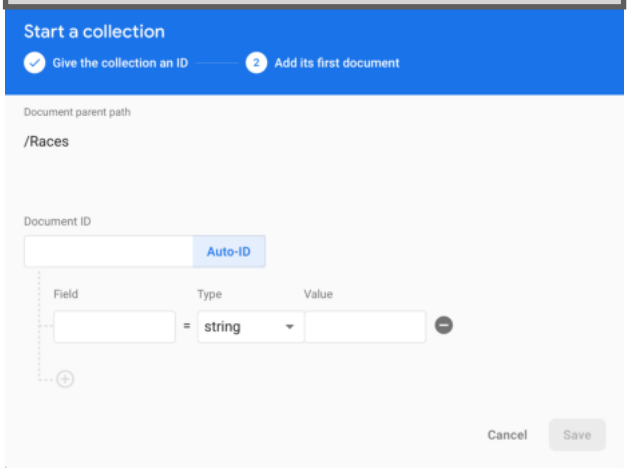
Now that it is all set up, I will create the database.



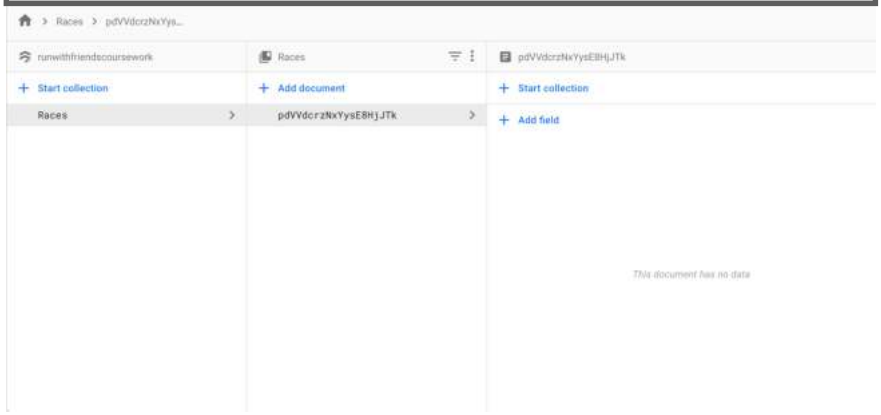
I will start by creating a collection called ‘Races’:



I want the documents to have an auto-ID



This is what my database now looks like



Now the database is all set up.

Version 3.6 - Initialising the database in the swift file

Now I can begin working on the main problem of this screen, which is initialising the database with the users selected distance.

To start, after importing Firebase to the Swift file class, I created the following required variables.

```
var ref: DocumentReference? = nil
var db: Firestore!
```

I then created a function called setUpDatabase.

```
func setUpDatabase()
{
    print("setting up db")

    //creates a new document (of auto-id) in the Races collection with the distance field set to the distance selected by the user
    ref = db.collection("Races").addDocument(data: ["code": 1234, "distance": distanceObject!, "player2Joined": false])

    //creates a new collection in that document called "Players"...
    //...and inside that collection a new document is created with the following fields.
    db.collection("Races").document(ref!.documentID).collection("Players").document("Player1").setData([
        "playerDistance": 0,
        "isReady":false,
        "finalTime":"",
        "averageSpeed":0.0,
        "topSpeed":0.0,
        "speedDataArray": [0.0],
        "distanceDataArray": [0.0],
        "timeDataArray": [0.0],
        "hasFinished":false
    ])
}
```

The purpose of this function is to firstly create the auto-id document with the race distance chosen by the user and with the randomised code. Secondly, inside this document, because the user who is creating the race is assigned ‘player1’ the Player1 document in the Players collection is created with all its necessary fields (see design section of project).

As you can see from the code above, the code is not randomised at this point (it has set to be 1234). This is the next stage of development for this screen.

Version 3.7 - Randomising the code

I firstly created a global variable called code. It is very important that this variable is a string because the first digit may be randomised to a '0'. If it was an integer the first digit would be made redundant.

```
var code = ""
```

I then created a function called randomiseCode. In it, four integers are created with their values being random number from 0 to 9. The code variable is set to the concatenation of those integers converted to strings.

```
func randomiseCode ()
{
    let int1 = Int.random(in: 0 ... 9)
    let int2 = Int.random(in: 0 ... 9)
    let int3 = Int.random(in: 0 ... 9)
    let int4 = Int.random(in: 0 ... 9)
    code = String(int1) + String(int2) + String(int3) + String(int4)
}
```

I can then use replace the 1234 in the setUpDatabase function with the code variable.

```
ref = db.collection("Races").addDocument(data: ["code": code, "distance": distanceObject!, "player2Joined": false])
```

It was important that I used a variable to store the randomised code, as opposed to having the randomiseCode function return the value, because I will need to use the 4 digit code in the next screen when I display the code.

Version 3.8 - Finishing touches

I have added the following code to the viewDidLoad function and I have also created a new function called viewDidAppear().

```
24      override func viewDidLoad()
25      {
26          super.viewDidLoad()
27          //randomise the code before setting up the database
28          randomiseCode()
29
30          db = Firestore.firestore() //required for setting connecting to database
31          setUpDatabase()
32
33          distanceLabel.text = String(distanceObject!) + "m"
34      }
35
36      override func viewDidAppear(_ animated: Bool)
37      {
38          changeStoryboard()
39      }
```

Let me explain what this code means:

In Swift, when transitioning to a new screen, the viewDidLoad function is called first (whilst the screen is loading), and then when the screen appears, another function called ‘viewDidAppear’.

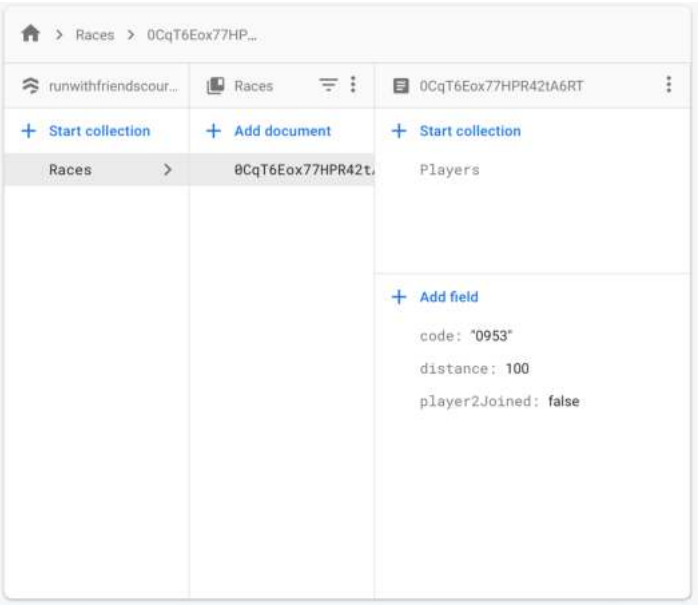
Therefore, looking at my code, when the screen loads in the background, the viewDidLoad function is called. So now looking at my viewDidLoad function, firstly, the 4 digit code will be randomised and then the database is set up. Now, because this code will take minimal time to compute, when the screen appears to the user, the viewDidAppear function is triggered and the app will transition to the next screen (because of the changeStoryboard function).

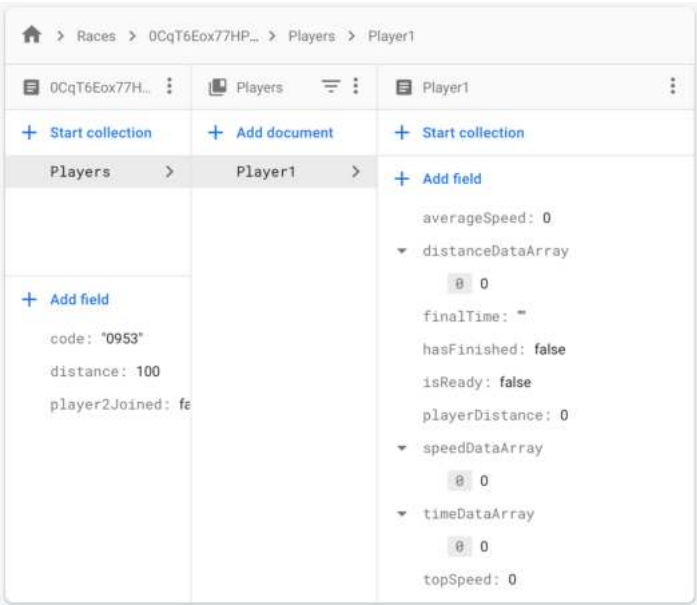
Version 3.9 - Testing

I ran my app and selected a 100 metre race. At the same time, I also had the Google Firebase website open in the background displaying my database (which was empty at the time).

As soon as I selected the 100m distance and transitioned to the initDatabaseScreen, the database had updated: A new document was successfully added to the Races collection with a random code and the correct distance and inside this document, a new collection had been created with the name ‘Player1’ containing all the fields set.

I tested this again with the different race distances and all was successful and also all the codes where seemingly random.





Version 3 - Review

What has been done

I have created the initialise database screen. The design is very simple because the user does not interact at all with this screen. However, in the background, a lot is going on. The 4-digit code is randomised and then a new document in the table is created with the required fields.

What has changed from the original design

As explained already I have changed the look of the UI from the design. Instead of it showing a spinning wheel, in my opinion it is now more informative because it shows a the distance the user selected.

Apart from this nothing else has changed.

What has been tested

I firstly have tested to see if the race distance which the user has selected can be passed successfully from the previous screen.

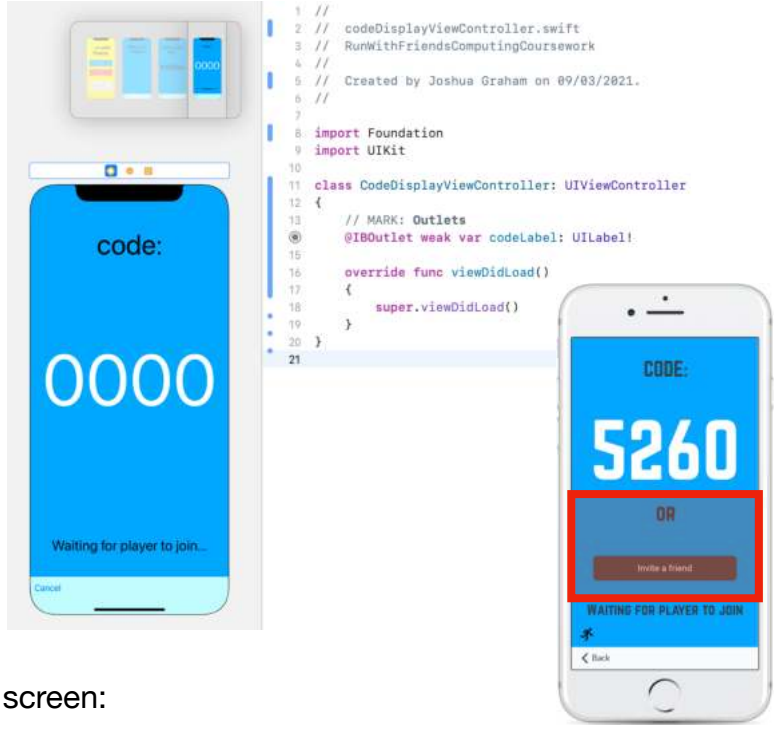
I have also tested that my app can communicate with the database.

During this testing I also observed the codes to see if they were successfully being randomised - and they were.

Version 4 - Building the display code screen

Version 4.1 - Creating the UI on main storyboard

Here is my UI screen design. I have already connected the main storyboard screen to the swift class.



Display code screen from initial design

My screen has changed slightly from my initial design. As you can see I have not implemented the invite a friend button (highlighted below). This is because I have taken the decision to not implement user accounts, so hence there would be no way to implement an invite button.

Version 4.2 - Passing data from previous screen

There are two variables which I will need to access from the previous screen in this screen:

1) The 4-digit code - so it can be shown to the user
2) The ID of the auto-id document so I can access that specific race in the database.

Therefore, using the same method I used to pass data between the selectDistanceViewController and the initDatabaseViewController, I created two new optional variables.

I then updated the changeStoryboard function in the initDatabaseViewController class:

```
// MARK: Variables
var codeObject: String?
var documentObject: String?
```

```
func changeStoryboard()
{
    print("change storyboard")
    let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
    if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier:
        "CodeDisplayViewController") as? CodeDisplayViewController
    {
        destinationViewController.codeObject = code
        destinationViewController.documentObject = ref!.documentID
        //ref!.documentID holds the ID of the document of the specific race being established.
        self.present(destinationViewController, animated: true, completion: nil)
    }
}
```

Now going back to the CodeDisplayViewController, I added the following line to the viewDidLoad function so that when the screen appears the codeLabel shows the 4 digit code:

```
codeLabel.text = codeObject!
```

As you can see the code now can be seen by the user →

Version 4.3 - Checking to see if the opponent has joined

As soon as the view loads, I want the app to constantly check if the other player has joined (by checking if the player2HasJoined field changes to read true). Therefore, I added the following (highlighted) code to the class:

```
21  var ref: DocumentReference? = nil
22  var db: Firestore!
23
24  override func viewDidLoad()
25  {
26      super.viewDidLoad()
27
28      codeLabel.text = codeObject!
29
30      db = Firestore.firestore() //line required for database
31      checkForJoin() //calls the checkForJoin function
32  }
33
34  func checkForJoin()
35  {
36
37  }
```

Above, I added the necessary variables for the database, then added the checkForJoin function at the end of the viewDidLoad() function and then I created the checkForJoin function (which is empty for now).

So now the next step is to write the checkForJoin function. The aim of it is to essentially check to see if the player2Joined field has been changed from false to true. This has to be checked constantly because it needs to update to the other player joining in real time.

One way to approach this would be to create a loop and constantly check every second or so to see if the field has changed to true. I have not implemented this because firstly, it would be a very inefficient way of tackling this problem. Secondly, using the Firebase library, there is a simple way to

The player2JoinedField

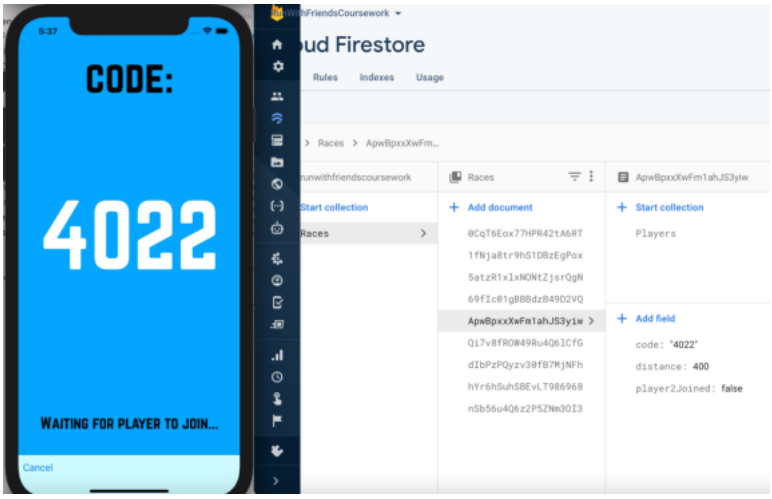
approach this problem (as you can see below):

```
34      func checkForJoin ()
35      {
36          //Adds a listener to the document where the code field is the same as the code/codeObject
37          db.collection("Races").whereField("code", isEqualTo: codeObject!)
38              .addSnapshotListener { querySnapshot, error in
39                  guard let snapshot = querySnapshot else {
40                      print("Error fetching snapshots: \(error!)") //in case of error
41                      return
42                  }
43                  snapshot.documentChanges.forEach { diff in
44                      if (diff.type == .modified) //when a field in the document is 'modified' this is triggered
45                                                  //i.e. when the player2HasJoined field is changed to true
46                      {
47                          //sets a timer for 0.8 seconds which when finished calls the player2Joined function
48                          Timer.scheduledTimer(timeInterval: 0.8, target: self, selector:
                                #selector(self.player2Joined), userInfo: nil, repeats: false)
49                      }
50                  }
51          }
52      }
53
54      @objc func player2Joined ()
55      {
56          print("Player 2 Joined")
57      }
```

To explain the code above, firstly, the document with the matching 4-digit code is found from the database.

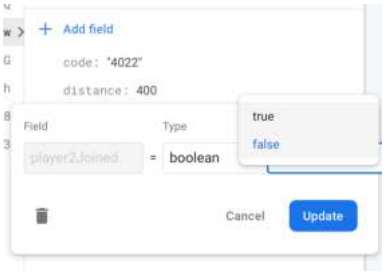
Then, what I have done is added a ‘snapshot listener’, which essentially checks the player2Joined field in the background. As soon as player 2 joins and the player2HasJoined field changes from true to false, the state becomes ‘modified’ and the snapshot listener is triggered. When this happens, I have set my program to call the player2Joined function (after a 0.8 second wait because I wanted to add a small delay).

As I have not coded the joining part of the app I needed a way to test if my code was working. I got around this however by editing the player2Joined field to true on the firebase website:



I ran my app and created a new race. I then located the document of the race I had just created in my database (as you can see in the image on the left the codes are matching).

I then modified the player2Joined field from false to true. →

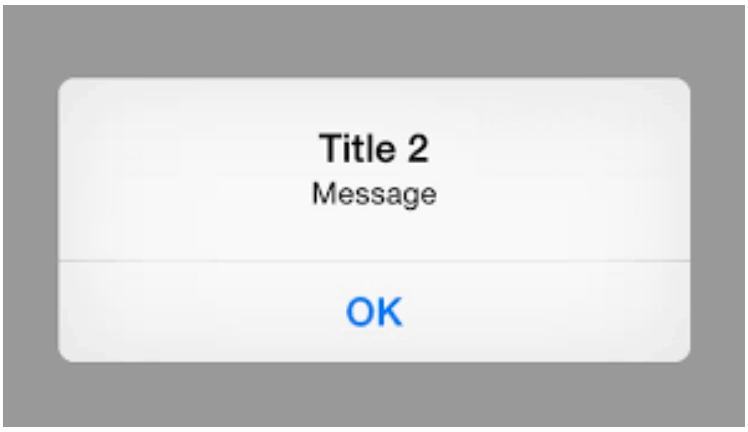


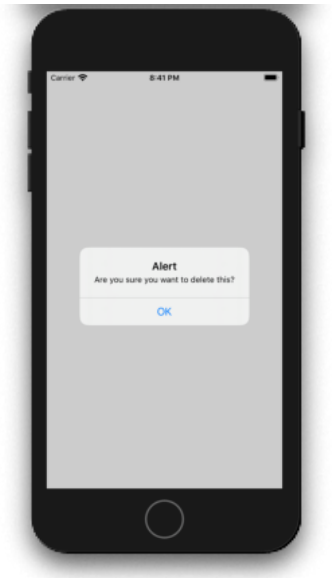
As expected, the console printed “Player 2 Joined” almost instantly.

Version 4.4 - Alerting the user the opponent has joined

Once the other player has entered the code and joined the race I need notify the user that the other player has successfully joined.

I could do this by presenting an alert to the user which would look something similar to the following two images:



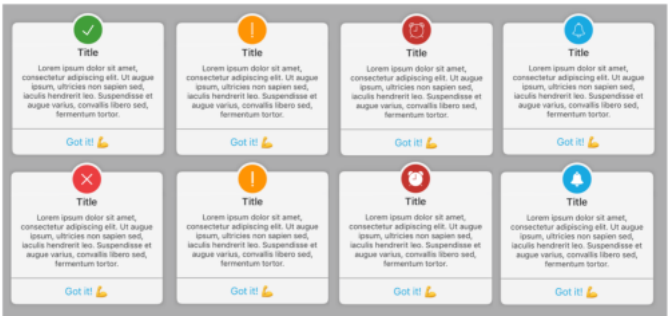


However, I would not like to implement for two reasons:

Firstly, it doesn’t match my colour scheme so it would seem out of place. Secondly, it shouldn’t be necessary for the user to press the ‘OK’ button to dismiss the alert; it should automatically dismiss after a short amount of time.

Instead, I found a library on Github called ‘CDAlertView’, which is a highly customisable alert popup. It’s also great because its usage is very similar to the built in Swift AlertView (above).

I am going to create one similar to the top left example (the one with the green success tip), however without the button.



CDAlertView example images

First things first, I installed the library and then imported the library into the CodeDisplayViewController.

This code then displays the alert and vibrates the users phone to notify them.

```
@objc func player2Joined ()
{
    print("Player 2 Joined")

    //Creates the alert, of type 'success' which means it will have a green tick
    //.show() means the alert is shown
    CDAlertView(title: "Success", message: "Player 2 has joined", type: .success).show()

    //Vibrates the phone to notify the user
    let generator = UINotificationFeedbackGenerator()
    generator.notificationOccurred(.success)

    changeStoryboard()
}

func changeStoryboard()
{
    print("change storyboard")
}
```

Version 4.5 - The cancel button

There arises a situation if the second player doesn't join the race. If this happens, currently, the user is stuck on the screen and is forced to quit and restart the app if they want to go back to the home screen.

Therefore, the cancel button is very important because it allows the user to return to the home screen.

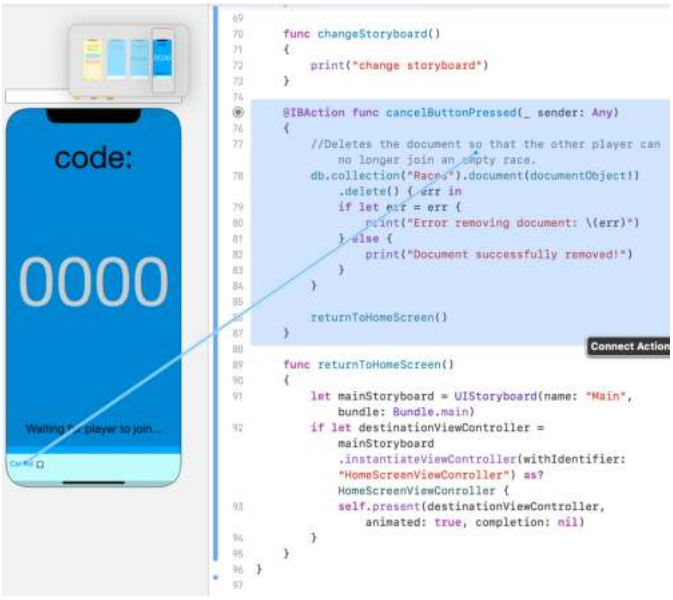


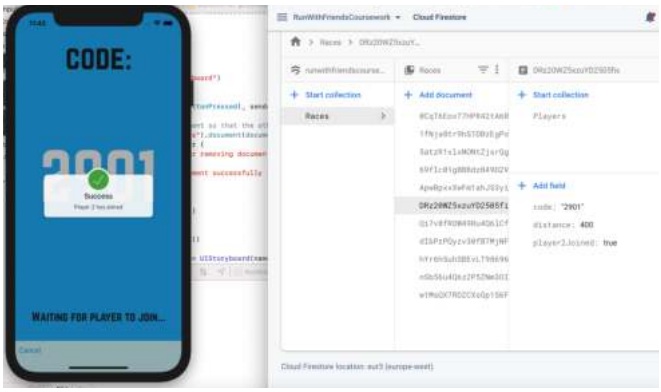
Image of me connecting the cancel button to the Action Function

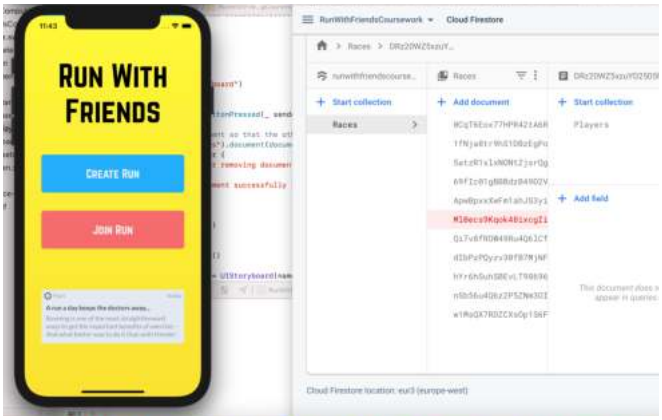
Now with the above code added, when the the cancel button is pressed the document is deleted from the database so that the other user cannot join a race with no one else in. As you can see the auto-id of the document is referenced with the ‘documentObject’ variable which was passed from the previous screen.

Once this document is deleted from the collection, the user is returned to the home screen.

Version 4.7 - Testing

I have created a small test plan just to check that this screen of the app is working as expected with no errors. Again if I needed to imitate the other user joining, I changed the player2HasJoined field on the Google Firebase website:

Test	Outcome	Evidence
Test that when the other player joins the success alert shows.	The alert shows and can be dismissed by tapping anywhere on the screen.	
The user is returned to the home screen when the cancel button is pressed.	The user is returned to the home screen.	*This is the screen after pressing the cancel button:* 

Test	Outcome	Evidence
The document is deleted when the user presses the cancel button.	The document is deleted in the database.	*As you can see the red shows that that document has been deleted:* 

Version 4 - Review

What has been done

I have created the screen which displays the 4-digit code needed to connect with an opponent. The user will wait on this screen until the other player joins the race. My program constantly checks, using a 'snapshot listener', whether the other user has joined by checking the field in the database. If it changes, then the user is notified with a custom success Alert view. The user will then be taken to the next screen. If the user wants to go back to the home screen they can press the cancel button and the record/document of the race is deleted from the database.

What has changed from the original design

As explained already, due to my decision to not have user accounts, I cannot implement the 'invite a user' function.

What has been tested

I firstly have tested to see if the code and document-id is being passed successfully from the previous screen.

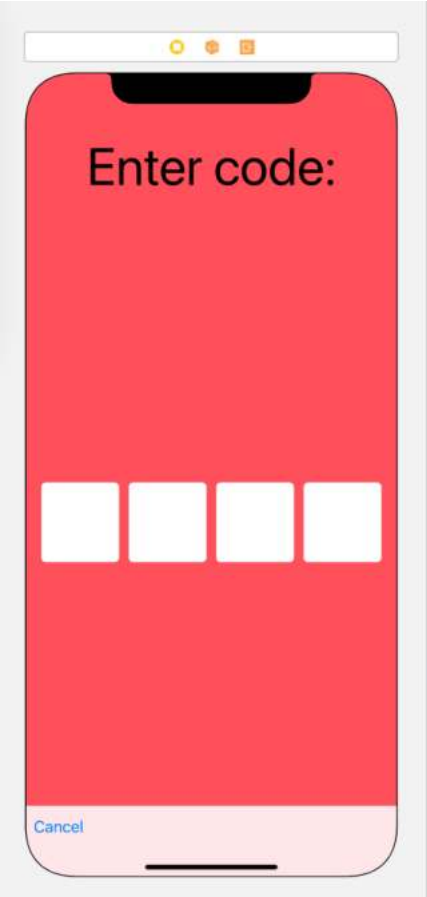
I have also tested that the snapshot listener works so that when the player2HasJoined field changes it is triggered.

I then also tested the success alert and also the functionality as a result of pressing the cancel button.

Version 5 - Building the enter code screen

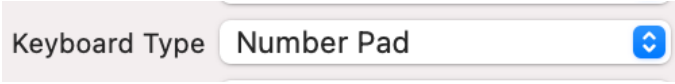
Version 5.1 - Creating the UI on main storyboard

As per normal I created a new screen in the main storyboard. Just like my prototype, I added 4 text fields to the screen (one text field for each digit).




```
3  //  RunWithFriendsComputingCoursework
4  //
5  //  Created by Joshua Graham on 11/03/2021.
6  //
7
8  import Foundation
9  import UIKit
10
11 class EnterCodeViewController: UIViewController
12 {
13     //MARK: IBOutlets
14     @IBOutlet weak var tf1: UITextField!
15     @IBOutlet weak var tf2: UITextField!
16     @IBOutlet weak var tf3: UITextField!
17     @IBOutlet weak var tf4: UITextField!
18
19     override func viewDidLoad()
20     {
21         super.viewDidLoad()
22     }
23 }
24
```

I also set the keyboard type of each text field to 'number pad', so that the user can only enter numbers.



Version 5.2 - Programming the text fields

It is important to set up the text fields in a way so it provides the user with an easy experience. Therefore, I have devised the following aims for which the text fields should follow:

1. When the screen appears the user is prompted to enter the first digit - the keyboard pops up.
2. Each text field only holds one digit -> The user is automatically moved to the next text field (i.e. when a digit is entered in the first text field field, the next digit inputted will be put in the second text field.)

3. When the user has inputted 4 digits, the code is checked:
    - if it is incorrect the text fields are cleared and the user is prompted to enter the first digit again.
4. The text cursor is not shown
5. If the user taps outside the text field area, the keyboard is dismissed

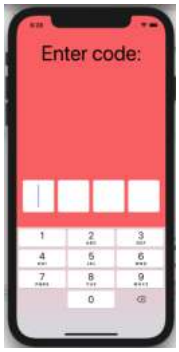
I will now go through how I programmed all these rules. Each numbered rule will have a corresponding sub version number.

I will also use iterative testing here: I will test each version as I go along.

Version 5.2.1

To make the keyboard pop up for the first text field, I very simply added the following line to the viewDidLoad function:

```
override func viewDidLoad()
{
    super.viewDidLoad()

    //activates text field 1:
    tf1.becomeFirstResponder()
}
```



Just before tested this, I added the code to transition to this screen after pressing the 'Join A Run' button on the home screen.

As you can see on the right the keyboard appears as soon as the screen appears:

Version 5.2.2

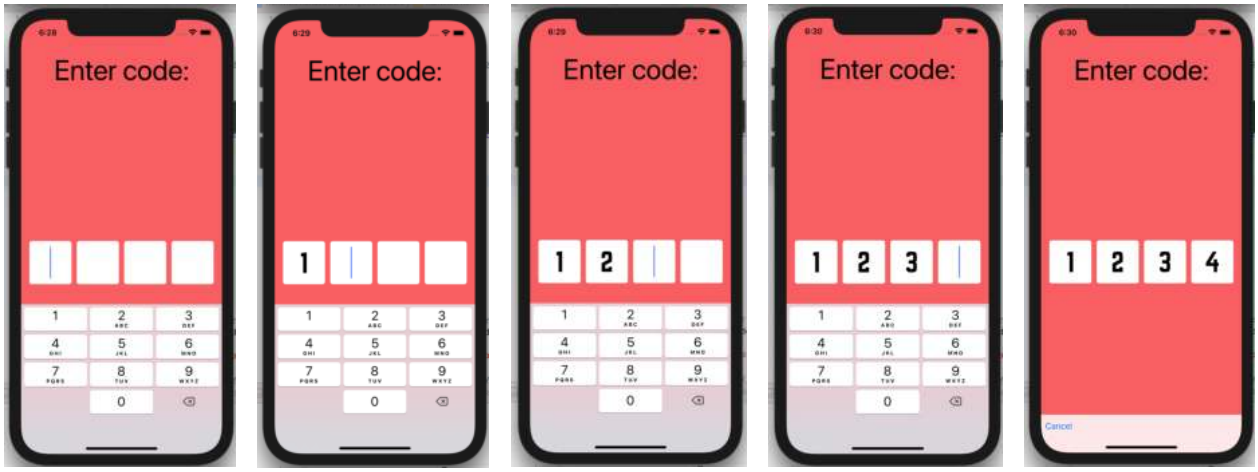
```
27      //function is triggered when the user types in any of the four text fields
29      @IBAction func textFieldEdited(_ sender: UITextField)
30      {
31          print("textEditChanged has been pressed")
32
33          //vibrates the phone very lightly
34          let generator = UIImpactFeedbackGenerator(style: .light)
35          generator.impactOccurred()
36
37          //counts the number of characters/digits in that text field
38          let count = sender.text?.count
39
40          //if the number of digits is 1, then move to the next text field:
41          if count == 1
42          {
43              switch sender {
44              case tf1:   //if text field 1 now contains a digit, then text field 2 is activated
45                  tf2.becomeFirstResponder()
46              case tf2:   //if text field 2 now contains a digit, then text field 3 is activated
47                  tf3.becomeFirstResponder()
48              case tf3:   //if text field 3 now contains a digit, then text field 4 is activated
49                  tf4.becomeFirstResponder()
50              case tf4:    //if text field 4 now contains a digit, then it is disabled and dismissed...
51                           //... so that the user cannot enter another digit.
52                  tf4.isEnabled = false
53              default:
54                  print("default")
55              }
56          }
      }
```

I created an action function called ‘textFieldEdited’, which is connected to all 4 text fields (as you can see on the right), and is triggered whenever the user types on the keyboard.

The code above ensures that each text field can only holds one digit. When the user enters a digit into the first keyboard, this function is triggered, and the user is moved automatically to the next text field.

The input of the function, ’*sender*’ is very important. It holds whichever text field was edited. Then you apply a switch case to the variable so a different instruction can be carried out depending on which one was edited.

I then proceeded to test this function. The first digit entered goes into the first text field. The next digit is entered into the second text field and the next will go into the third field. Therefore, it is impossible to enter two digits into the same text field. When four digit have been entered the keyboard is successfully dismissed.



Version 5.2.3

In the function above, in the switch statement, if the program enters ‘*case tf4*’, it means that all 4 digits have been entered. Therefore, it is at this point in the program that the code needs to be checked.

I will not be writing the code which actually checks the code at this point because I will be dealing with that part of the problem in the next version (version 5.3).

Instead, I will create a function which will be called at this point in the program, named ‘checkCode’, which will be responsible for checking the code. However, I will not write the database code, only the code which is to relevant to the text fields, as a sort of blueprint:

```
61      func checkCode()
62      {
63          //the users inputted code is stored in a variable
64          let code = tf1.text! + tf2.text! + tf3.text! + tf4.text!
65          print ("code: \(code)")
66
67          //Variable which will be set to whether or not the code has been found in the database
68          var codeIsCorrect = false
69
70          //************************
71          // * CHECK THE CODE HERE *
72          //************************
73
74          if (codeIsCorrect == false)
75          {
76              //clears all the text fields
77              tf1.text = ""
78              tf2.text = ""
79              tf3.text = ""
80              tf4.text = ""
81              tf1.becomeFirstResponder()  //prompts the user to enter in the first digit
82              tf4.isEnabled = true    //re-enables the last text field
83
84              //Creates the alert, of type 'error' which means it will have a red x
85              let alert = CDAlertView(title: "Invalid", message: "Wrong code", type: .error)
86              alert.autoHideTime = 3 // This will hide alert box after 3 second
87              alert.show()
88
89              //Vibrates the phone in a pattern which represents an error
90              let generator = UINotificationFeedbackGenerator()
91              generator.notificationOccurred(.error)
92          }
93          else
94          {
95              //Creates the alert, of type 'success' which means it will have a green tick
96              CDAlertView(title: "Success", message: "You have joined the race", type: .success).show()
97
98              //Vibrates the phone to notify the user of the success
99              let generator = UINotificationFeedbackGenerator()
100             generator.notificationOccurred(.success)
101         }
```

This is where the function is called from:

```
case tf3:   //if text field 3 now contains a digit, then text field 4 is activated
    tf4.becomeFirstResponder()
case tf4:    //if text field 4 now contains a digit, then it is disabled and dismissed
             //... so that the user cannot enter another digit.
    tf4.isEnabled = false
    checkCode()
default:
    print("default")
}
```

As you can see I have added the code for the alert pop ups using the same CDAlertView library as used for the alerts in the previous screen. If the code is incorrect, a red alert is shown, but if the code is correct, a green success alert is shown. It is important that the red alert is automatically dismisses after 3 seconds because the user will need to enter the code again, whereas with the green success alert, the user will be taken to the next screen so there is no need for it to automatically dismiss after a period of time.

To test this function, I first ran my application as it is (simulating an incorrect code entry because line 68 sets the codeIsCorrect variable to false) ->

Successfully, the error alert view shows and the text fields are cleared. The user keyboard is also bought up, prompting the user to enter a number into the first text field.

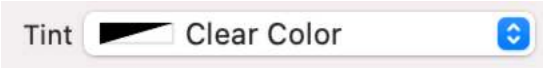
I then ran my program again, but this time changing line 68 so that my codeIsCorrect variable is set to true.

As expected, the success alert view is shown ->

At this point in the program, the user would be transitioned to the raceScreen. Therefore, I created the changeStoryboard function (which is empty for now) and have referenced it from the checkCode function.

Version 5.2.4

To stop the cursor from being seen, I changed the following colour property in the main storyboard of each text field from blue to clear:



However, when testing this change, I found it was quite difficult to know which text field you were typing into. Therefore, I added the following code:

```
override func viewDidLoad()
{
    super.viewDidLoad()

    //activates text field 1:
    tf1.becomeFirstResponder()
    tf1.layer.borderWidth = 2

    tf1.layer.borderColor = UIColor.darkGray.cgColor
    tf2.layer.borderColor = UIColor.darkGray.cgColor
    tf3.layer.borderColor = UIColor.darkGray.cgColor
    tf4.layer.borderColor = UIColor.darkGray.cgColor

    tf1.layer.cornerRadius = 6
    tf2.layer.cornerRadius = 6
    tf3.layer.cornerRadius = 6
    tf4.layer.cornerRadius = 6
}
```

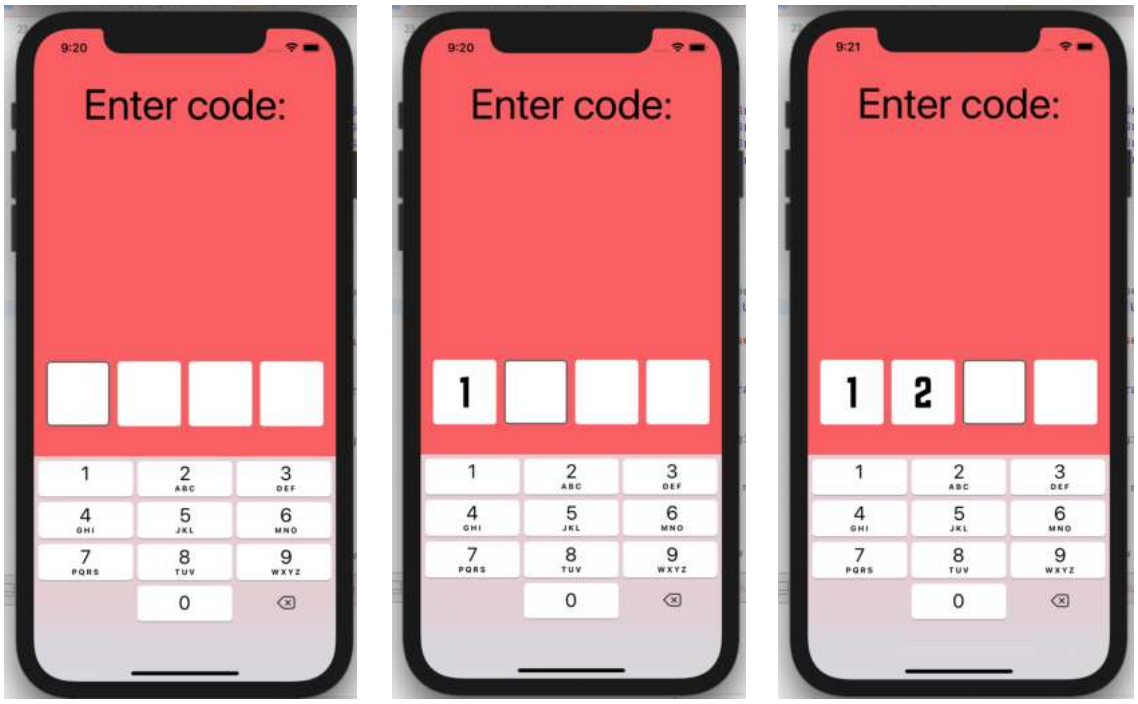
```
if count == 1
{
    switch sender {
    case tf1:   //if text field 1 now contains a digit, then text field 2 is activated
        tf2.becomeFirstResponder()
        tf1.layer.borderWidth = 0
        tf2.layer.borderWidth = 2
    case tf2:   //if text field 2 now contains a digit, then text field 3 is activated
        tf3.becomeFirstResponder()
        tf2.layer.borderWidth = 0
        tf3.layer.borderWidth = 2
    case tf3:   //if text field 3 now contains a digit, then text field 4 is activated
        tf4.becomeFirstResponder()
        tf3.layer.borderWidth = 0
        tf4.layer.borderWidth = 2
    case tf4:    //if text field 4 now contains a digit, then it is disabled and dismissed...
                 //... so that the user cannot enter another digit.
        tf4.isEnabled = false
        tf4.layer.borderWidth = 0
        checkCode()
    default:
        print("default")
    }
}
```

This highlighted lines of code has been added to the textFieldEdited action function

In the viewDidLoad function I set all the text fields a dark grey border (with all the widths being set to 0 by default) and with a slightly curved edge (of radius 6).

When a text field becomes active, I then set its border width to 2. When the view loads, the first text fields border width is set to 2. But when the user types the first character, the first text field's border width is set to 0 whilst the second text field's border width is set to 2.

Here is what it looks like:



Version 5.2.5

To dismiss the keyboard when the user taps outside the view, I need to add a gesture recogniser. It is essentially a listener which is triggered when the gesture occurs, in my case, the user tapping the background (in swift terms, the view).

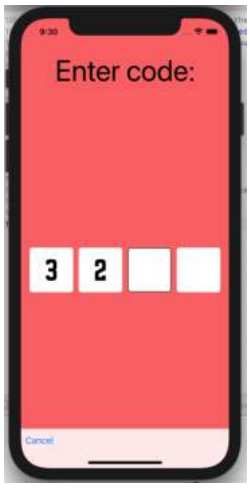
I added the following line to the viewDidLoad function which calls the function 'hideKeyboard' when the view is tapped:

```
let tapGesture = UITapGestureRecognizer(target: self, action: #selector(hideKeyboard))
view.addGestureRecognizer(tapGesture)
```

The hideKeyboard function hides the keyboard:

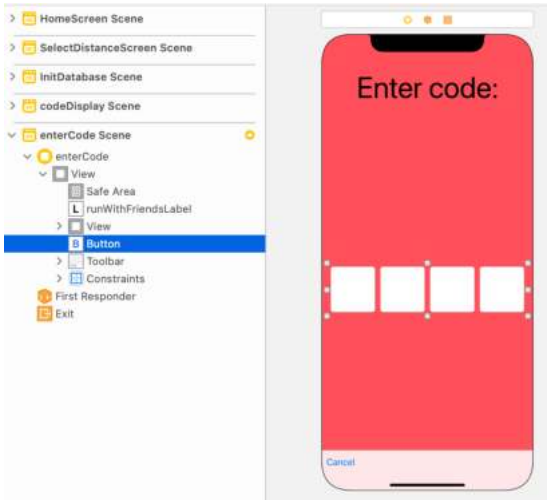
```
@objc func hideKeyboard()
{
    view.endEditing(true)    //hides the keyboard
}
```

However, when testing this, a problem arose:

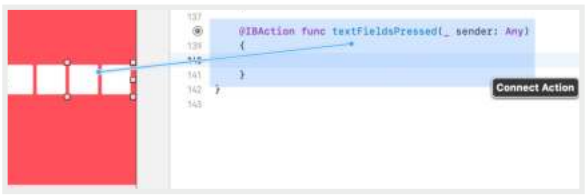


In this screenshot I entered two digits and then pressed on the background and the keyboard was successfully dismissed. However, the problem is now that I am unable to bring up the keyboard and enter more digits.

Therefore, I went back to the main storyboard file and added an invisible button which is positioned on top of the four text fields.



I then connected up an action function from the button to the enterCodeViewController swift file:

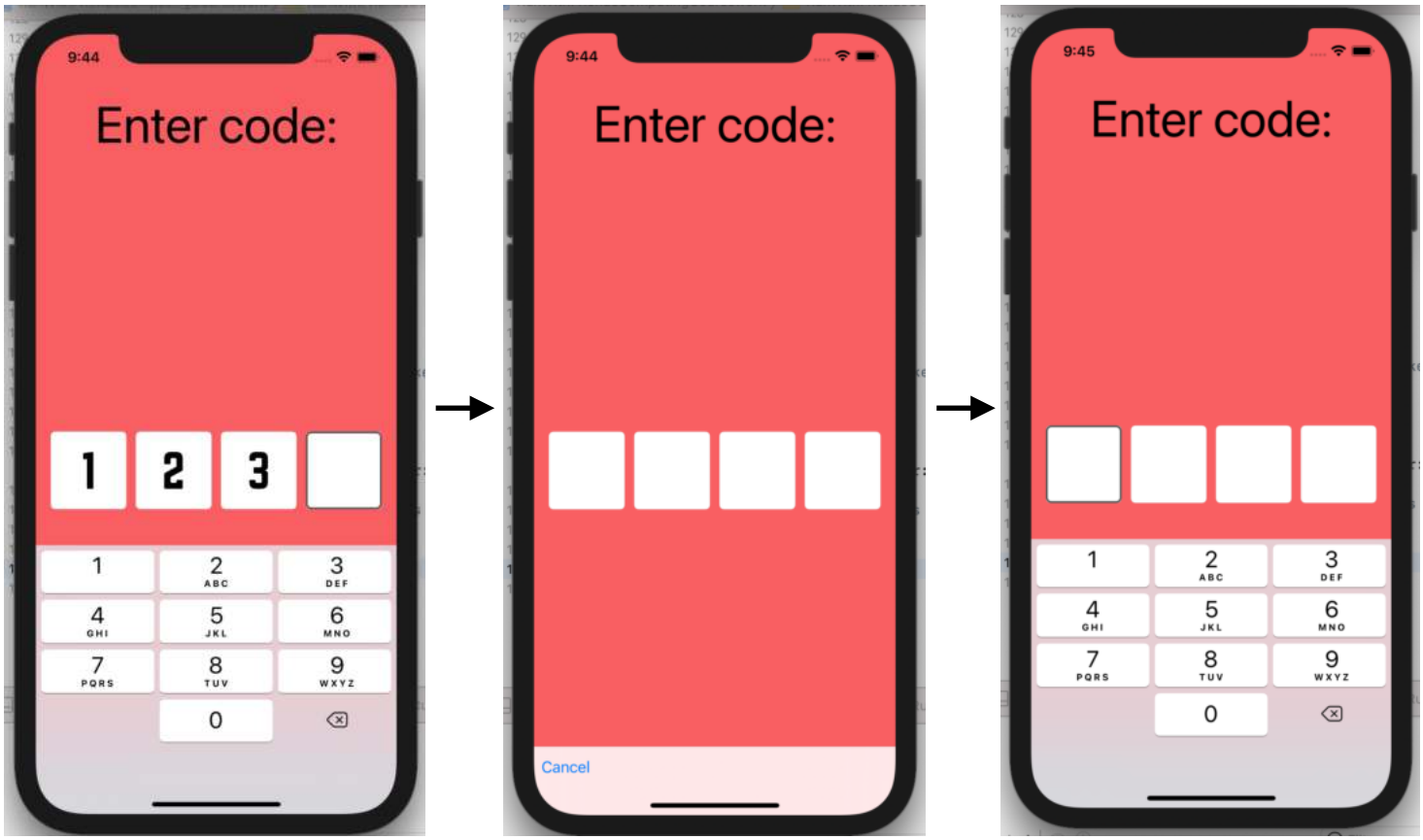


I edited the hideKeyboard function so that it also clears all the text fields when called. And so therefore, when the button is pressed (i.e. the text fields are ‘pressed'), I can assign the first text field active:

```
@objc func hideKeyboard()
{
    view.endEditing(true)   //hides the keyboard

    tf1.text = ""
    tf2.text = ""
    tf3.text = ""
    tf4.text = ""

    //reset all the border widths to 0 as it is not know which text field was the last active.
    tf1.layer.borderWidth = 0
    tf2.layer.borderWidth = 0
    tf3.layer.borderWidth = 0
    tf4.layer.borderWidth = 0
}

@IBAction func textFieldsPressed(_ sender: Any)
{
    tf1.becomeFirstResponder()  //prompts the user to enter in the first digit again
    tf1.layer.borderWidth = 2
}
```

Testing my code was all successful:

For example, after entering 3 digits I pressed the background. The keyboard was dismissed and the text fields were cleared. Then when I pressed in the text field area again the keyboard was bought up allowing me to type into the first text field:



All the programming for the text fields is now complete!

I also added the following code so that the user can press the cancel button to return to the home screen and connected the cancelButtonPressed function to the cancel button:

```
@IBAction func cancelButtonPressed(_ sender: Any)
{
    returnToHomeScreen()
}

func returnToHomeScreen()
{
    let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
    if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier:
        "HomeScreenViewConroller") as? HomeScreenViewConroller {
        self.present(destinationViewController, animated: true, completion: nil)
    }
}
```

Version 5.3 - Checking the user's code with the database

This is the checkCode function which I wrote back in the design section. I will now write this algorithm in swift by splitting the code into coloured blocks and rewriting each section:

I will not write the transition function yet because I have not created the screen to which I will transition to (the race screen).

```
procedure checkCode()
        Find all the documents in the "Races" collection of the database where the "code" field == code
        if the number of documents found == 0 then
            //Incorrect code
            show alert to user saying that the code is incorrect
            userInputsCode()    //Allow the user to re-enter the code
        else
            //Correct code
            show alert to user saying that the code is code
            To this document update the following field {
                "player2Joined" = true          //sets the field called 'player2Joined' to true
            }
            In this document add to the 'Players' collection a document called "Player2" with the following fields {
                "playerDistance": 0,            //field called 'playerDistance' which is set to 0
                "isReady":false,                //field called 'isReady' which is set to false
                "finalTime":"",                 //field called 'finalTime' which is set to an empty string
                "averageSpeed":0.0,             //field called 'averageSpeed' which is set to 0.0
                "topSpeed":0.0,                 //field called 'topSpeed' which is set to 0.0
                "speedDataArray": [0.0],        //field called 'speedDataArray' which is set to a double array containing 0.0
                "distanceDataArray": [0.0],     //field called 'distanceDataArray' which is set to a double array containing 0.0
                "timeDataArray": [0.0],         //field called 'timeDataArray' which is set to a double array containing 0.0
                "hasFinished":false             //field called 'hasFinished' which is set to false
            }
            transition to raceScreenController {
                //this next bit of code is again needed because it labels the user as player2 and the opponent as player1
                destinationViewController.mainPlayerString = "Player2"
                destinationViewController.mainPlayerInt = 2
                destinationViewController.opponentPlayerString = "Player1"
                destinationViewController.opponentPlayerInt = 1
            }
        endif
endprocedure
```

Here is the code for the blue section:

```
    //finds all documents in the Races collection of the database where the code is the same as the user's inputted code
    db.collection("Races").whereField("code", isEqualTo: code).whereField("player2Joined", isEqualTo: false)
        .getDocuments() { (querySnapshot, err) in
            if let err = err {
                print("Error getting documents: \(err)")
            }
            else
            {
                if (querySnapshot?.documents.count == 0)    //if no documents are found
                {
                    codeIsCorrect = false
                }
                else
                {
                    codeIsCorrect = true
                }
            }
    }
```

I have made one slight change from my pseudocode above. Instead of only searching for every document with the same code as the user's inputted code, I have added an additional search criteria to my query: I will also check that the player2Joined field is set to false. So to summarise, the query will return all the

documents in the database which have not been joined and also have the same code as the user's code.

I do not want there to be complications with two races having the same code. Therefore, at this point I will only continue if only one document is returned. So before I can code the red section of my pseudocode, I will need to code this:

```
if (querySnapshot?.documents.count == 0)    //if no documents are found
{
    codeIsCorrect = false
}
else
{
    codeIsCorrect = true
    //Duplicate codes in database
    if ((querySnapshot?.documents.count)! > 1)
    {
        print("ERROR - duplicate codes")
        codeIsCorrect = false
    }
}
```

The chances of a duplicate code occurring is very very low (1 in 10,000). However, I think it would be worth preventing this from occurring in the first place. Therefore, I went back to my InitDatabaseViewController swift class and added this code (which is very similar to the one used in this screen), hence **version 3.10** of my development:

InitDatabaseViewController

```
func randomiseCode ()
{
    let int1 = Int.random(in: 0 ... 9)
    let int2 = Int.random(in: 0 ... 9)
    let int3 = Int.random(in: 0 ... 9)
    let int4 = Int.random(in: 0 ... 9)
    code = String(int1) + String(int2) + String(int3) + String(int4)

    setUpDatabase()
}

func setUpDatabase()
{
    print("setting up db")

    //Queries the database to find a document with the same randomly generated code.
    db.collection("Races").whereField("code", isEqualTo: code)
        .getDocuments() { [self] (querySnapshot, err) in
            if let err = err {
                print("Error getting documents: \(err)")
            }
            else
            {
                if ((querySnapshot?.documents.count)! >= 1)    //if there is a duplicate
                {
                    randomiseCode()
                }
            }
        }
```

Now, if the random code happens to already exist in the database, then the code is re-randomised.

Therefore, I can delete the code I just added before because it is impossible for there to be a duplicate code:

```
if (querySnapshot?.documents.count == 0)    //if no documents are found
{
    codeIsCorrect = false
}
else
{
    codeIsCorrect = true
    //D   ate codes in database
    i   shot?.docume      1)
    {
        print("E      codes")
        code
    }
}
```

Now I can implement the green section of my pseudocode.

I first created a new global variable called raceID which will store the ID of the document:

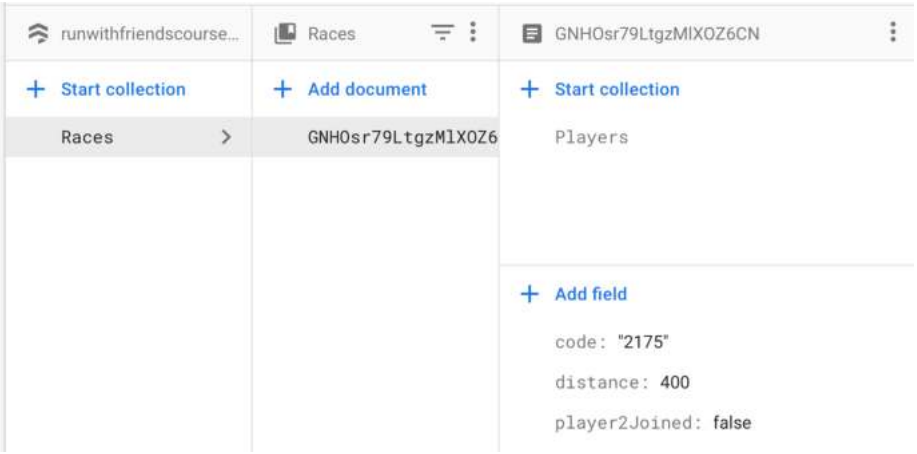
```
var raceID = ""
```

I then added the following code to my checkCode function:

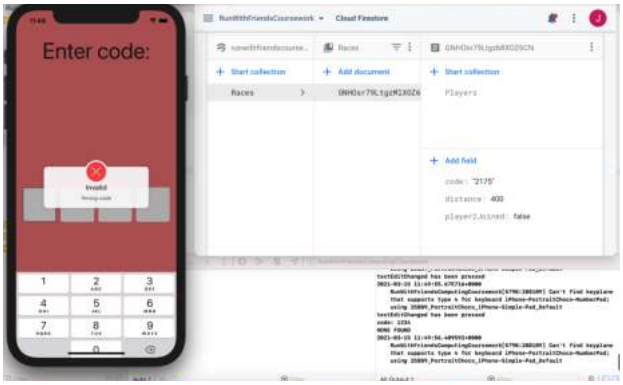
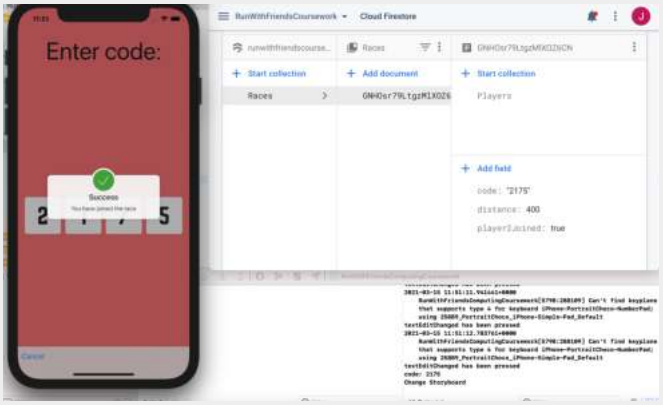
```
if (querySnapshot?.documents.count == 0)    //if no documents are found
{
    codeIsCorrect = false
}
else
{
    codeIsCorrect = true

    //the following code must be executed here because it is inside the database query closure:
    self.raceID = (querySnapshot?.documents[0].documentID)!  //stores the ID of the document where this code was found

    //Creates a document reference to this document
    let documentRef = self.db.collection("Races").document(self.raceID)

    //updates the player2Joined field to true
    documentRef.updateData([ "player2Joined": true ]) { err in
        if let err = err {
            print("Error updating document: \(err)")
        } else {
            //print("Document successfully updated")
        }
    }

    //adds a new document to the 'Players' collection called 'Player2' with the necessary fields
    documentRef.collection("Players").document("Player2").setData([
        "playerDistance": 0,
        "isReady":false,
        "finalTime":"",
        "averageSpeed":0.0,
        "topSpeed":0.0,
        "speedDataArray": [0.0],
        "distanceDataArray": [0.0],
        "timeDataArray": [0.0],
        "hasFinished":false
    ])
}
```

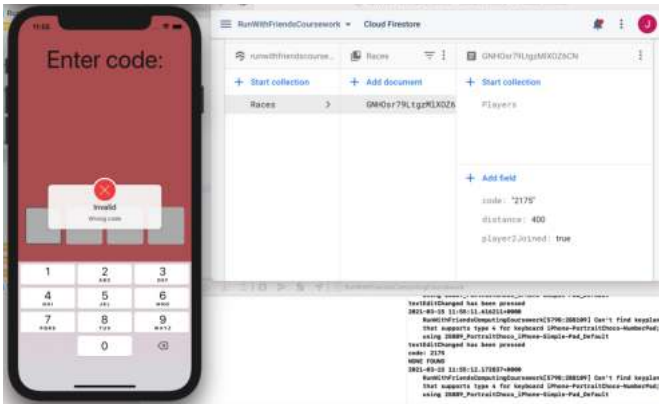
Version 5.4 - Testing

I firstly cleared the database. I then ran the app and created a new race. I then quit the app so that the document could be retained in the database. This is what my database looked like:



I then ran my app once more and went to the enterCodeScreen and then conducted the following tests:

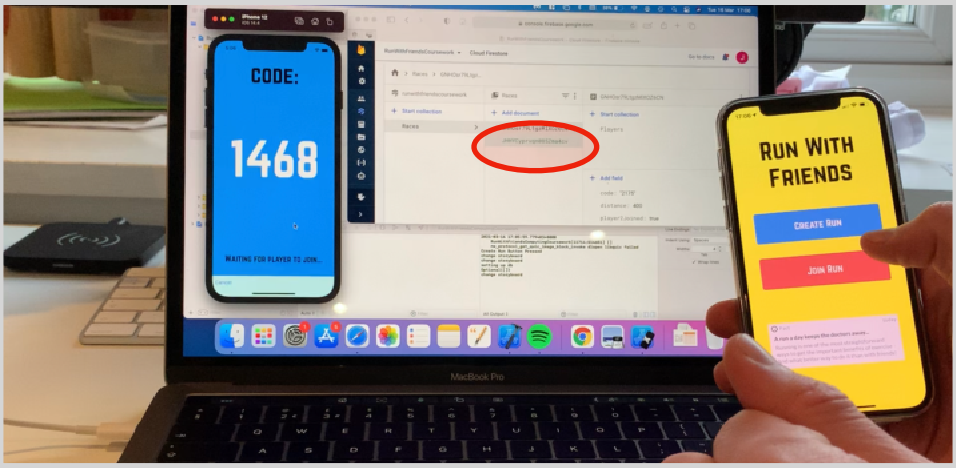
Test	Outcome	Evidence
Entering an incorrect code - a code which doesn't exist in the database. I entered the code 1234.	App says the code is incorrect.	
Entering a correct code - a code which does exist in the database. I entered the code 2175.	App says the code is correct and in the database the player2Joined field is updated, aswell as the player2 document being created	

Test	Outcome	Evidence
Entering a correct code - a code which does exist in the database, however it has already been joined. I again entered the code 2175 (since the document at this point had been joined).	App says the code is incorrect (because it thinks this race has already been joined).	

Again to test that my app could work smoothly in real life I ran my app on the simulator and also at the same time ran it on my phone. I would then proceed to create a race on the iPhone simulator and then joined the race on my phone.

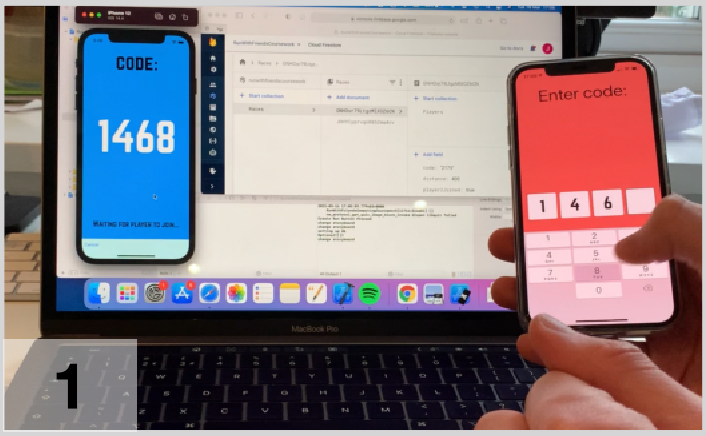
I recorded a video so that I could measure the time lag/difference between interactions between the two players and the database by looking at the time stamps. The following images are screenshots from the video:

Here is the app being run on my phone and on the simulator, with the Firebase database in the background.

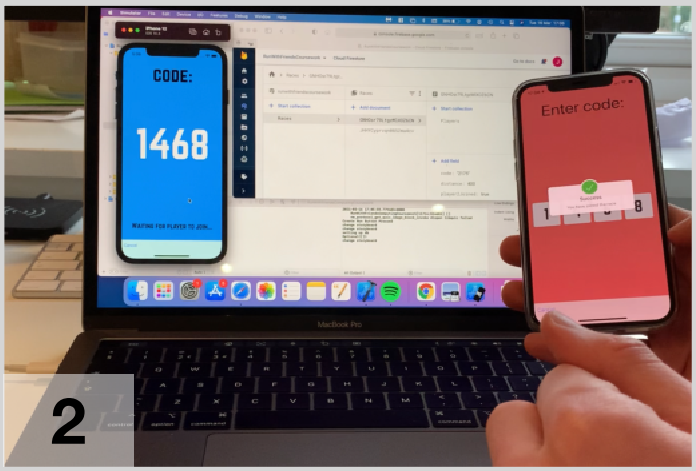


I then created a race on the simulator. As you can see the new document has just been added to the database.

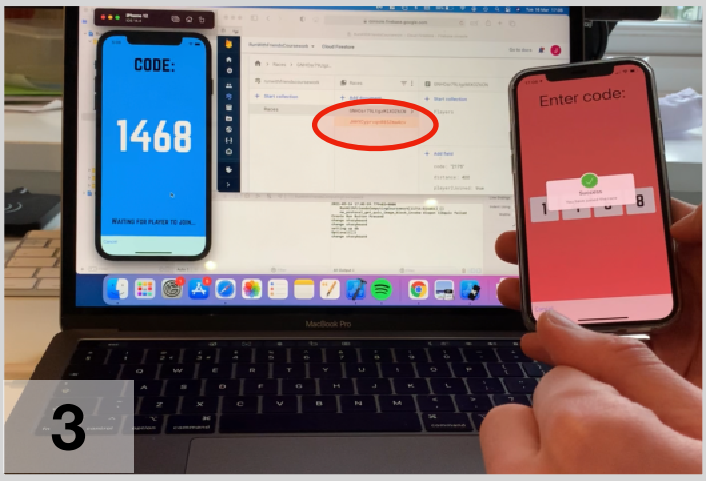
I calculated the delay from the moment the last digit was entered (screenshot 1):



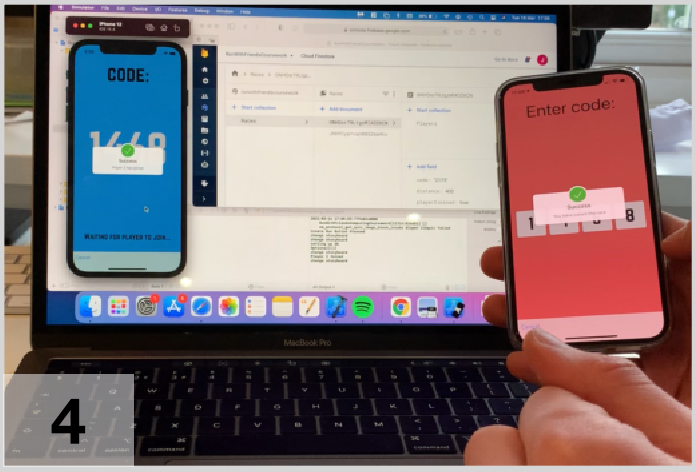
Time stamp: 15.53 seconds
The last of the four digits is entered into the phone.



Time stamp: 15.97 seconds (delay = 0.44s)
The success alert shows on player 2's phone (my phone).



Time stamp: 16.2 seconds (delay = 0.67s)
The database fields are updated.



Time stamp: 16.97 seconds (delay = 1.43s)
The success alert shows on player 1's phone (the simulator).

As you can see, there is a delay of 1.43 seconds from the last digit being entered on player 2's phone to the success alert being shown on player 1's phone. For a different application (like an online game), this would be a very substantial delay. However, for the purpose of this part of my app (the two players joining the same race) a super low latency is not necessary because it is not a time critical feature. Therefore, I am quite satisfied with this connection speed.

I then sent my stakeholders the video by email and asked them for any feedback and I also asked them their opinion on the 1.4 second.

Aron's response:

First of all, of course I like the design - it followed the prototype very closely. From this video it seems to me that it is very simple to create a race. With regards to the lag, in my view it's only a very small time delay so I am delighted with what you have built so far.

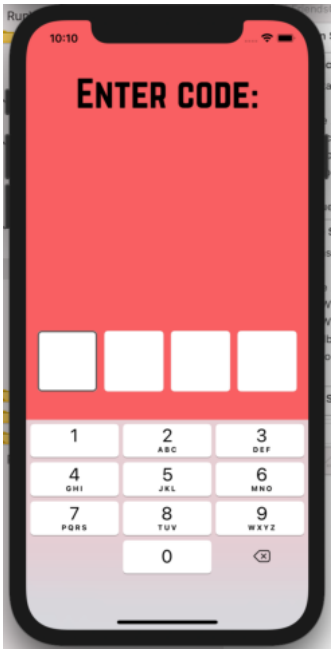
I just noticed a small mistake on the screen where you enter in the code. The text at the top of the screen reading 'Enter code:' is not in the nice font you had. Just being picky :)

David's response:

I love how easy it is to create a race. I particularly like how the cells highlight when you type in the 4 digit race code. It is not super critical to have a very short delay. 1.4 seconds is still very fast. However, I think it would be important to reduce the delay between the players during the actual race.

— — — — — — — — — — — — — — — — — — — —

Overall, the feedback from my stakeholders was very positive. I made the one change to the font of the label in the enterCodeScreen:



Version 5 - Review

What has been done

The enterCodeScreen has been created. It allows the user to enter a 4 digit code. The code is then checked by searching the database for any documents who's code field matches it. If this is so, a success alert is shown. But if the code is not found (i.e. the code is incorrect), the user is shown a red failure alert. The user can then re-enter the code.

I programmed all the code to search the database as well as to update the database if the user (player 2) enters the correct code. I also designed the text fields according to a set of predefined rules so that the process of typing in the code is as simple and quick as possible.

What has changed from the original design

The design is almost identical. One addition is that when the user is typing into a text field, it is highlighted (by giving it a border).

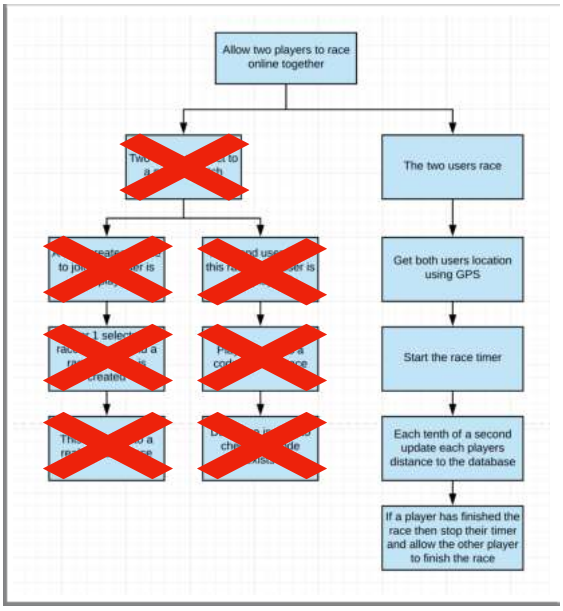
What has been tested

I have tested the text field to their full extent so that can bugs can occur when inputting the code. The user can only enter numbers into the text fields (because of the digit keyboard) so in that sense the code is already validated. It is also impossible to enter more than 4 digits or less than 4 digits.

I have also tested that the algorithm for searching for a matching code in the database is error-free. It tested every possibility using a test plan table.

I then ran my app on my phone and on the simulator simultaneously to see what it is like to use in real life, and also to measure the latency.

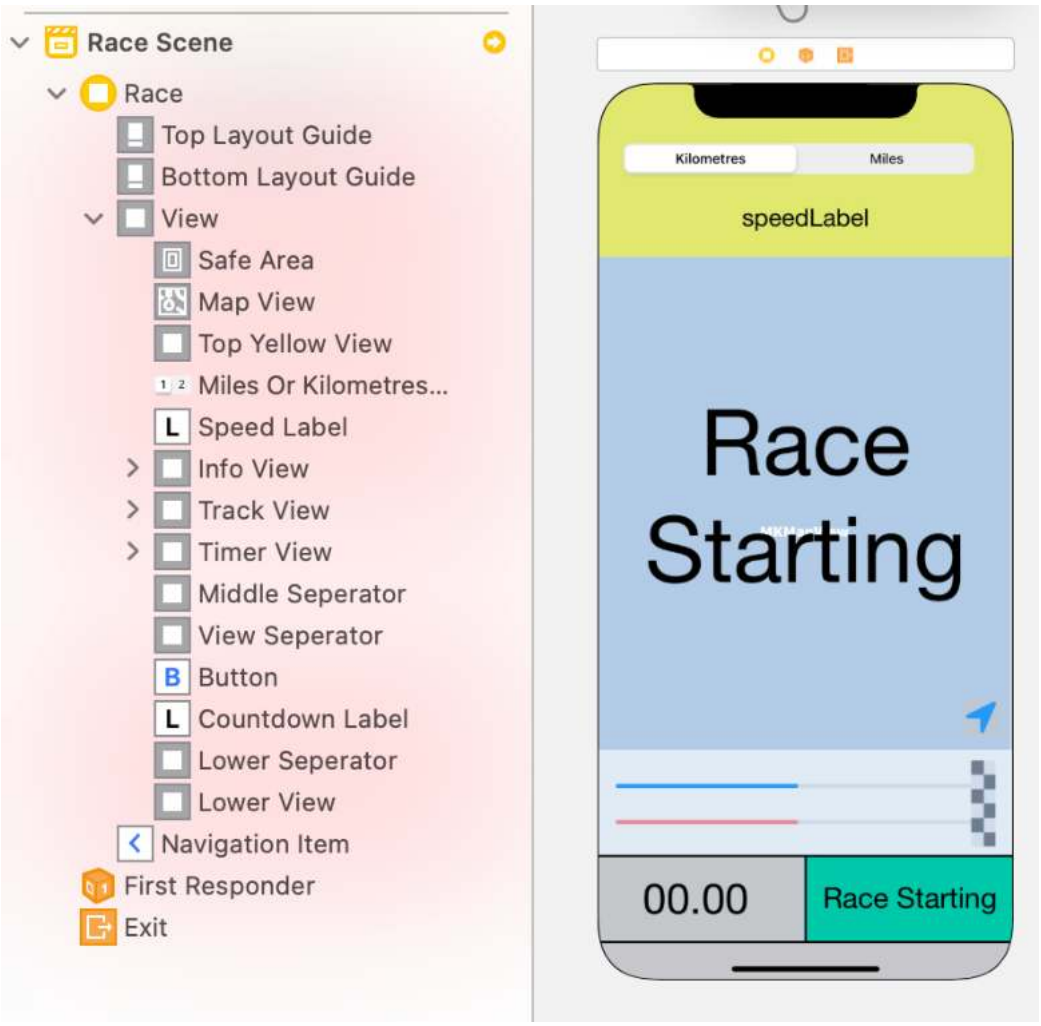
At this point in my development, looking back at the diagram I created of my solution broken down, I have developed half of my solution. Now all I have left to do is develop the actual online realtime race:



Version 6 - Building the race screen

Version 6.1 - Creating the UI on main storyboard

Here's the screen I created on the main storyboard file - it is almost identical to the screen I created in my design section.



I then connected the components which will need to be interacted with to the RaceViewController swift file:

```
14      // - MARK: Outlets
        @IBOutlet weak var mapView: MKMapView!
        @IBOutlet weak var milesOrKilometresSegment: UISegmentedControl!
        @IBOutlet weak var speedLabel: UILabel!
        @IBOutlet weak var timerLabel: UILabel!
        @IBOutlet weak var infoLabel: UILabel!
        @IBOutlet weak var player1ProgressBar: UIProgressView!
        @IBOutlet weak var player2ProgressBar: UIProgressView!
        @IBOutlet weak var countdownLabel: UILabel!
        @IBOutlet weak var locateButton: UIButton!
```

Version 6.2 - Configuring the map

In the background of this screen a map centred around the user's location will be shown. Currently, the map doesn't show anything. I need to program it in the following order:

1) Due to privacy, the user has to first agree for their location to be accessed.
2) Initialise the users location.
3) Centre the map on the user's location
4) Continually update the user's location.

These steps will become more clear when I go through my code:

Firstly, as soon as the screen loads the checkLocationServices function is called:

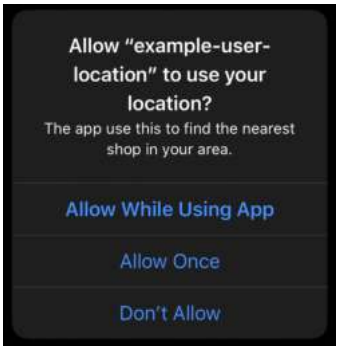
```
override func viewDidLoad()
{
    super.viewDidLoad()

    checkLocationServices()
}
```

```
func checkLocationServices()
{
    //checks if user has enabled their location.
    if CLLocationManager.locationServicesEnabled()
    {
        setupLocationManager()
        checkLocationAuthorization()
    }
    else
    {
        print ("location services not turned on")
    }
}
```

If the users location services is enabled the setUpLocationManager function is called, which essentially sets out that this swift class will need to manage the user's location:

```
51  func setupLocationManager()
52  {
53      locationManager.delegate = self
54      locationManager.desiredAccuracy = kCLLocationAccuracyBest   //best accuracy needed.
55      locationManager.requestWhenInUseAuthorization()    //request to access location when in use.
56      locationManager.activityType = .fitness     //most precise activty type.
57      //at this point ask user's both users to press ready
58  }
```

When line 55 is executed, if the user hasn't agreed to their location being accessed, an alert will be shown (similar to the one on the right):



To add a custom message to this alert, I added the following item to the info.plist file (a metadata/app property file list):

Next the checkLocationAuthorization function is called:

```
60      func checkLocationAuthorization ()
61      {
62          switch CLLocationManager.authorizationStatus()
63          {
64          case .authorizedWhenInUse:
65              locationManager.requestLocation()   //gets the user's location
66              mapView.showsUserLocation = true
67              centreViewOnUserLocation()
68              locationManager.startUpdatingLocation()
69          case .denied:
70              break
71          case .notDetermined:
72              locationManager.requestWhenInUseAuthorization()
73          case .restricted:
74              break
75          case .authorizedAlways:
76              break
77          }
78      }
```

It checks the user's authorisation status and if it is authorised the user's location is shown on the map. Then the centreViewOnUserLocation function is called, which centres the view around the user's location showing a region of 300 by 300 metres (due to the value of the regionInMeters global variable).

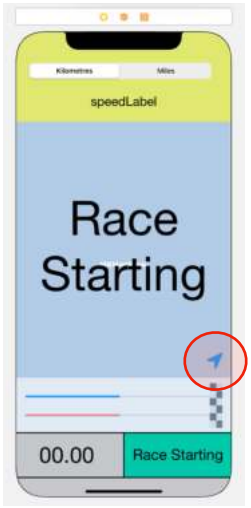
```
var regionInMeters: Double = 300

func centreViewOnUserLocation ()
{
    if let location = locationManager.location?.coordinate
    {
        print("--- REGIONING ---")
        let region = MKCoordinateRegion.init(center: location, latitudinalMeters: regionInMeters, longitudinalMeters: regionInMeters)
        mapView.setRegion(region, animated: true)
    }
}
```

Then the locationManager is told to start updating the user's location (line 68).

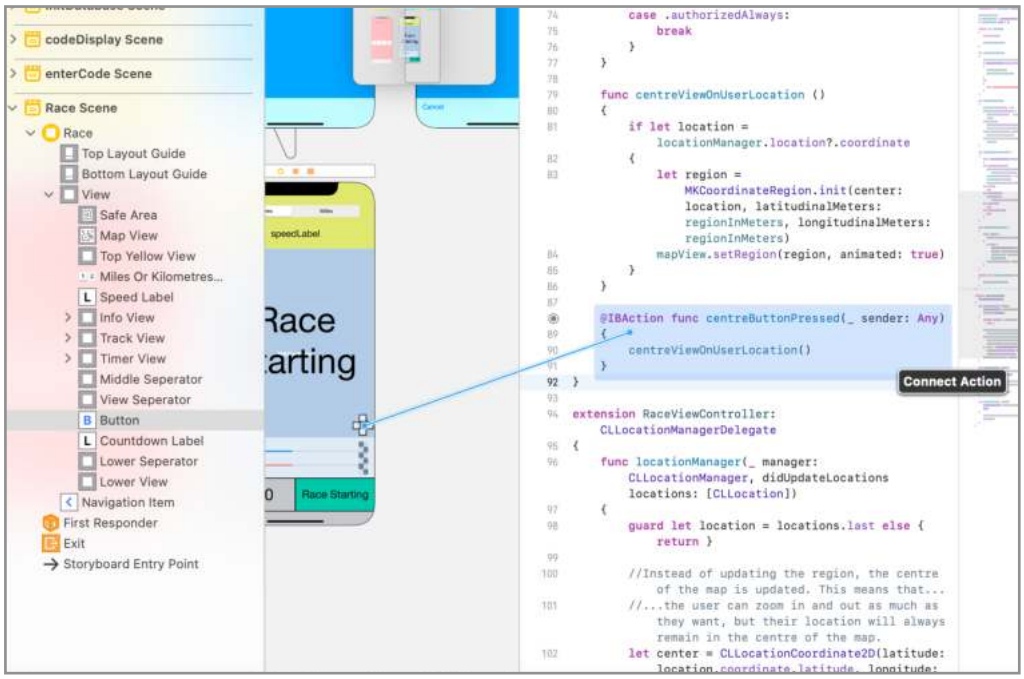
```
90  extension RaceViewController: CLLocationManagerDelegate
91  {
92      func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation])
93      {
94          guard let location = locations.last else { return }
95
96          //Instead of updating the region, the centre of the map is updated. This means that...
97          //...the user can zoom in and out as much as they want, but their location will always remain in the centre of the map.
98          let center = CLLocationCoordinate2D(latitude: location.coordinate.latitude, longitude: location.coordinate.longitude)
99          mapView.setCenter(center, animated: true)
100     }
101
102     func locationManager(_ manager: CLLocationManager, didChangeAuthorization status: CLAuthorizationStatus)
103     {
104         //If the user changes the location privacy in iOS settings, request permission to access location again.
105         checkLocationAuthorization()
106     }
107
108     func locationManager(_ manager: CLLocationManager, didFailWithError error: Error)
109     {
110         //If error has occured, print error.
111         print(error)
112     }
113 }
```

The above code is responsible for managing the user's location. The highlighted function is the most important of the three for my program. It is triggered when the user updates their location (i.e. when they move enough for the GPS to detect). I added two lines of code to this function (lines 98 and 99) which keeps the user's location centred on the map.

If the user wants to return to the original zoom span (300m), they should be able to press the centre button (shown on the right):

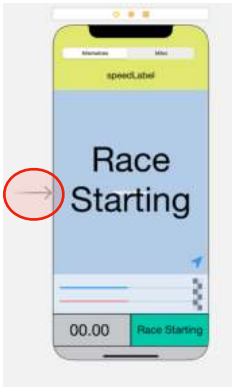


I created a new action function and connected it to the centre button:



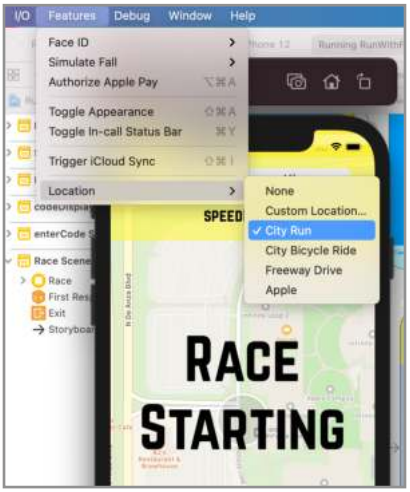
Now, it is time to test that the map is working as planned.

I firstly set the Race screen to the initial screen because I haven't added the transition to the race screen yet.



As soon as the app loaded, the iOS privacy alert was shown successfully with my custom message.

What's also great about the iOS simulator, is that you can simulate different locations and scenarios (see image below). I set the location to simulate a 'city run'.



This feature of the simulator, will be perfect to simulate and test the running feature of my app, later in the development of this screen.

When I gave the simulator this location, the map instantly panned to the user's (new) location and the users location was shown in the middle (hiding behind the letter R), constantly in the centre of the screen.

When I zoomed out, the user's location maintained in the centre of the screen and when I pressed the re-centre button, the normal 300m by 300m span was restored.

Version 6.3 - Updating the user's speed

The following code simply gets the user's speed (at their current location, I have stored the speed in a global variable called speedInMPS).

Depending on the user's current selected on the segmented control (the kilometres or miles setting at the top of the screen), the speed units may have to be displayed in either units. I have handled this by appropriately converting the speed from metres per second to the unit of choice.

```
189         // Displaying Speed
190         speedInMPS = location.speed
191         print("Speed (mps): \(speedInMPS)")
192
193         //If user has selected kilometres units
194         if milesOrKilometresSegment.selectedSegmentIndex == 0
195         {
196             if speedInMPS >= 0
197             {
198                 speedLabel.font = speedLabel.font.withSize(50)
199                 //rounds the calculated value, then trancates the decimal place, then converts it to String, then
                        adds " kmh"
200                 speedLabel.text = String (Int((speedInMPS * 3.6).rounded())) + " kmh"
201                 calibrating = false
202             }
203             else
204             {
205                 speedLabel.font = speedLabel.font.withSize(30)
206                 speedLabel.text = "Calibrating..."
207                 calibrating = true
208                 calibratingFunc()
209             }
210         }
211
212         //If user has selected miles units
213         if milesOrKilometresSegment.selectedSegmentIndex == 1
214         {
215             if speedInMPS >= 0
216             {
217                 speedLabel.font = speedLabel.font.withSize(50)
218                 //rounds the calculated value, then trancates the decimal place, then converts it to String, then
                        adds " mph"
219                 speedLabel.text = String (Int((speedInMPS * 2.23694).rounded())) + " mph"
220                 calibrating = false
221             }
222             else
223             {
224                 speedLabel.font = speedLabel.font.withSize(30)
225                 speedLabel.text = "Calibrating..."
226                 calibrating = true
227                 calibratingFunc()
228             }
229         }
```

Sometimes the speed received from GPS can be less than 0 (often -1 indicates an error). Therefore, if this is so, the calibrating global variable is set to true, and the calibratingFunc is called:

```
140     func calibratingFunc ()
141     {
142         //Only show alert if there is still an error with the user's GPS connection
143         if (calibrating == true)
144         {
145             presentCalibratingAlert()
146         }
147     }
148
149     func presentCalibratingAlert()
150     {
151         let alert = UIAlertController(title: "Calibrating...", message: "Currently we are calibrating and locating
                you so this app can run as smoothly as possible.", preferredStyle: .alert)
152
153         let action1 = UIAlertAction(title: "OK", style: .default, handler: nil)
154
155         let imgTitle = UIImage(named:"imgTitle.png")
156         let imgViewTitle = UIImageView(frame: CGRect(x: 10, y: 10, width: 30, height: 30))
157         imgViewTitle.image = imgTitle
158
159         alert.view.addSubview(imgViewTitle)
160         alert.addAction(action1)
161
162         self.present(alert, animated: true, completion: nil)
163     }
```

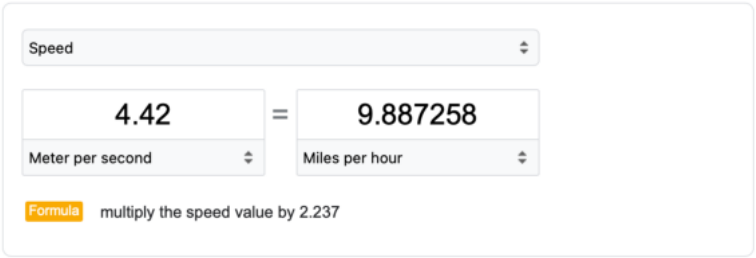
It ensures that if there is a problem with the GPS connection, the user is alerted so that they know the results of the race may be impacted.

When running my app, the speed was successfully shown and it could be displayed in the varying units:

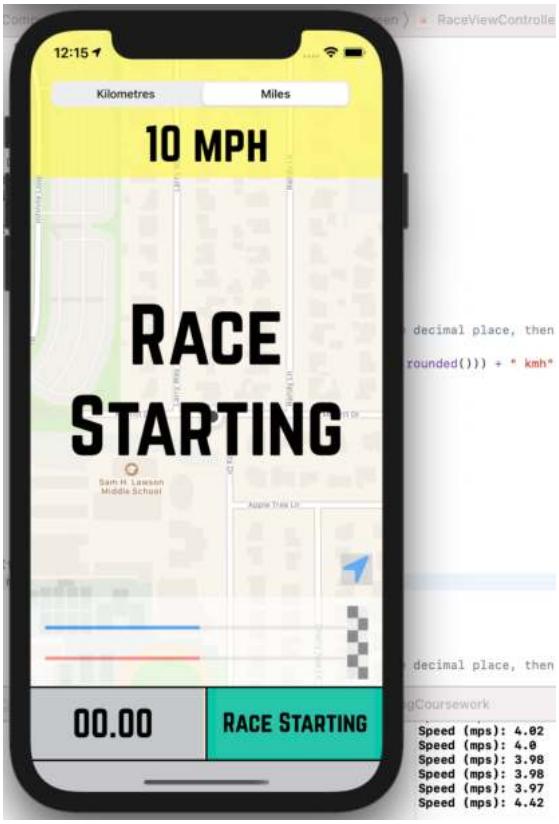
I also tested the accuracy of my unit conversions. I compared the speeds in the different location scenarios which can be simulated on the simulator, with the speeds found on this table I found online. I compared the speed in mps (which is printed to the console) in addition to the speed the user can see displayed in km and miles).

Method of travel	Typical speed (m/s)	Typical speed (mph)
Walking	1.5	3.4
Running	5	11
Cycling	7	15
Car	13 - 30	29 - 67

I also used an online convertor to test the accuracy of my program:



For example, in the screenshot on the right, the location in the simulator was set ‘city run’. The speed in mps (printed in the console) at the moment this screenshot was taken is 4.42, and the speed in mph was 10:



As you can see, the conversion is clearly accurate (to the nearest whole number) and the 10mph is a typical running speed according to the table.

Version 6.5 - Passing variables from previous screens

As I have talked about previously, the same code runs for both players. Therefore, I need to accommodate for this. On this screen, the user (both player 1 and player 2) will need to access following the variables:

- raceDocumentID - to access the specific race in the database. This will be passed from the previous screens.
- the race distance - this will be read from the database.

I also need to indicate whether the user is player 1 or player 2, using the following variables:

```
13 mainPlayerString = ""                //will store "Player1" if the user is player1 or "Player2" if the user is player2
14 mainPlayerInt = 0                    //will store 1 if the user is player1 or 2 if the user is player2
15 opponentPlayerString = ""            //will store the opposite of the above to variables ("Player1" or "Player2")
16 opponentPlayerInt = 0                //will store the opposite of the above to variables (1 or 2)
```

Here is the code for the transitions I added to the displayCodeScreen and enterCodeScreen:

```
func changeStoryboard()
{
    print("change storyboard")
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    if let destinationViewController = storyboard.instantiateViewController(withIdentifier: "RaceViewController") as? RaceViewController
    {
        destinationViewController.raceDocumentID = documentObject!
        //User is player 1:
        destinationViewController.mainPlayerString = "Player1"
        destinationViewController.mainPlayerInt = 1
        destinationViewController.opponentPlayerString = "Player2"
        destinationViewController.opponentPlayerInt = 2
        self.present(destinationViewController, animated: true, completion: nil)
    }
}
```

DisplayCodeScreen

```
func changeStoryboard ()
{
    print("Change Storyboard")

    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    if let destinationViewController = storyboard.instantiateViewController(withIdentifier: "RaceViewController") as? RaceViewController
    {
        destinationViewController.raceDocumentID = self.raceID
        //User is player 2:
        destinationViewController.mainPlayerString = "Player2"
        destinationViewController.mainPlayerInt = 2
        destinationViewController.opponentPlayerString = "Player1"
        destinationViewController.opponentPlayerInt = 1
        self.present(destinationViewController, animated: true, completion: nil)
    }
}
```

EnterCodeScreen

And these are the corresponding global variables on the RaceScreen:

```
// Passed from previous view controllers
var raceDocumentID = ""
var mainPlayerString = ""
var mainPlayerInt = 0
var opponentPlayerString = ""
var opponentPlayerInt = 0
```

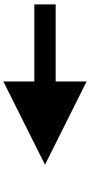
The following function is called when the race screen loads to get the distance so it can be used later during the race:

```
//Sets global variable 'raceDistance' to the race distance from the document in database
func getRaceDistance ()
{
    let documentRef = self.db.collection("Races").document(self.raceDocumentID)
    var d = ""
    documentRef.getDocument { (document, error) in
        if let document = document, document.exists {
            let dataDescription = document.data()
            d = dataDescription!["distance"]! as! String
            d = d.replacingOccurrences(of: "m", with: "")
            self.raceDistance = Double(Int(d)!)
        } else {
            print("Document does not exist")
        }
    }
```

Version 6.6 - Readying up

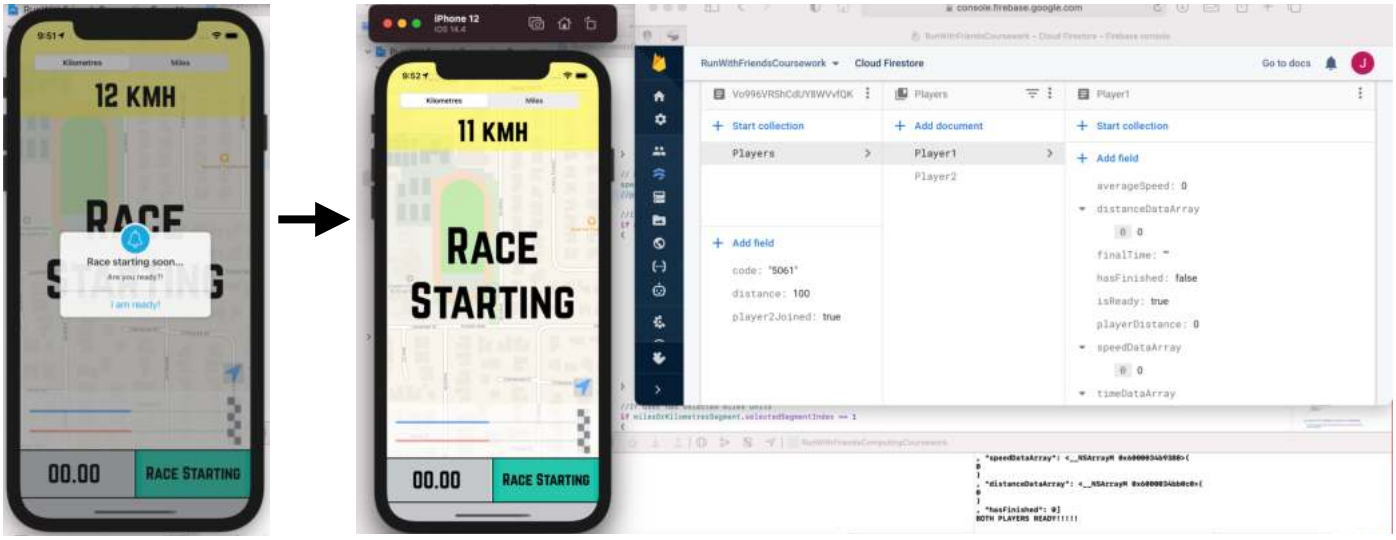
This is the function from my pseudocode I need to code in Swift. The ready button shown when the screen loads (in the viewDidLoad function):

```
procedure readyButtonPressed()
    To the mainPlayerString document update the following field {
        "isReady":true          //sets the field called 'isReady' to true
    }
    Read the "isReady" in the opponentPlayerString document {
        if the "isReady" field is set to true then
            startRaceCountdown()
        else
            wait 1 second
            readyButtonPressed()
        endif
    }
endprocedure
```



```swift
    override func viewDidAppear(_ animated: Bool)
    {
        showReadyButton()
    }

    // - MARK: Readying Up
    func showReadyButton()
    {
        //Shows alert with a button which reads 'ready'
        let alert = CDAlertView(title: "Race starting soon...", message: "Are you ready?!", type: .notification)
        let readyAction = CDAlertViewAction(title: "I am ready!")
        alert.add(action: readyAction)
        alert.canHideWhenTapBack = false
        alert.show() { (alert) in
            self.userIsReady()
        }
    }

    func userIsReady()
    {
        //updates the user's 'isReady' field to true
        var documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(self.mainPlayerString)
        documentRef.updateData([ "isReady": true ]) { err in
            if let err = err {
                print("Error updating document: \(err)")
            } else {
                print("Document successfully updated")
            }
        }

        //Adds a listener to the opponents document
        documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(self.opponentPlayerString)
        documentRef.addSnapshotListener { documentSnapshot, error in
                guard let document = documentSnapshot else {
                    print("Error fetching document: \(error!)")
                    return
                }
                let source = document.metadata.hasPendingWrites ? "Local" : "Server"
                print("\(source) data: \(document.data() ?? [:])")
            //if the field 'isReady' is set to true, both players are ready
            if (document.data()?["isReady"] as! Int == 1)
            {
                print("BOTH PLAYERS READY!!!!!")
            }
        }
    }
```

When the screen loads, the ready alert is displayed. When the user presses the ‘I am ready!’ button, the database is updated. If the other player is also ‘ready’, then for now “BOTH PLAYERS READY!!!!!” is printed to the console.



At this point, I need to commence the race countdown:

Version 6.7 - Race countdown

Firstly, when the view loads I will clear the countdownLabel text.

```
//clear countdownLabel text
countdownLabel.text = ""
```

At this point in the code the startCountdown function is called:=

```
if (document.data()?["isReady"] as! Int == 1)
{
    print("BOTH PLAYERS READY!!!!!")
    self.startCountdown()
```

```
// - MARK: Countdown
func startCountdown()
{
    countdownLabel.font = countdownLabel.font.withSize(140)
    infoLabel.text = "Race Starting"
    countdownTimerCount = 0
    //creates a timer which calls the function updateCountdownTime() every 1 second:
    countdownTimer = Timer.scheduledTimer(timeInterval: 1, target: self, selector: (#selector(RaceViewController.updateCountdownTime)), userInfo: nil, repeats: true)
}

@objc func updateCountdownTime()
{
    countdownTimerCount += 1
    if (countdownTimerCount == 3)
    {
        countdownLabel.font = countdownLabel.font.withSize(230)
        countdownLabel.text = "3"
        playVibrate()
    }
    else if (countdownTimerCount == 4)
    {
        countdownLabel.text = "2"
        playVibrate()
    }
    else if (countdownTimerCount == 5)
    {
        countdownLabel.text = "1"
        playVibrate()
    }
    else if (countdownTimerCount == 6)
    {
        infoLabel.text = "Race!"
        countdownLabel.font = countdownLabel.font.withSize(200)
        countdownLabel.text = "GO!"
        for _ in 1...3
        {
            DispatchQueue.main.asyncAfter(deadline: .now() + 0.2) {
                //self.playVibrate()
                print("vibrate 1")
            }
            self.playVibrate()
            print("vibrate 2")
        }
        //*Start the race timer here*
    }
    else if (countdownTimerCount == 7)
    {
        countdownLabel.text = ""
        countdownTimer.invalidate() //stops the timer
    }
}
```

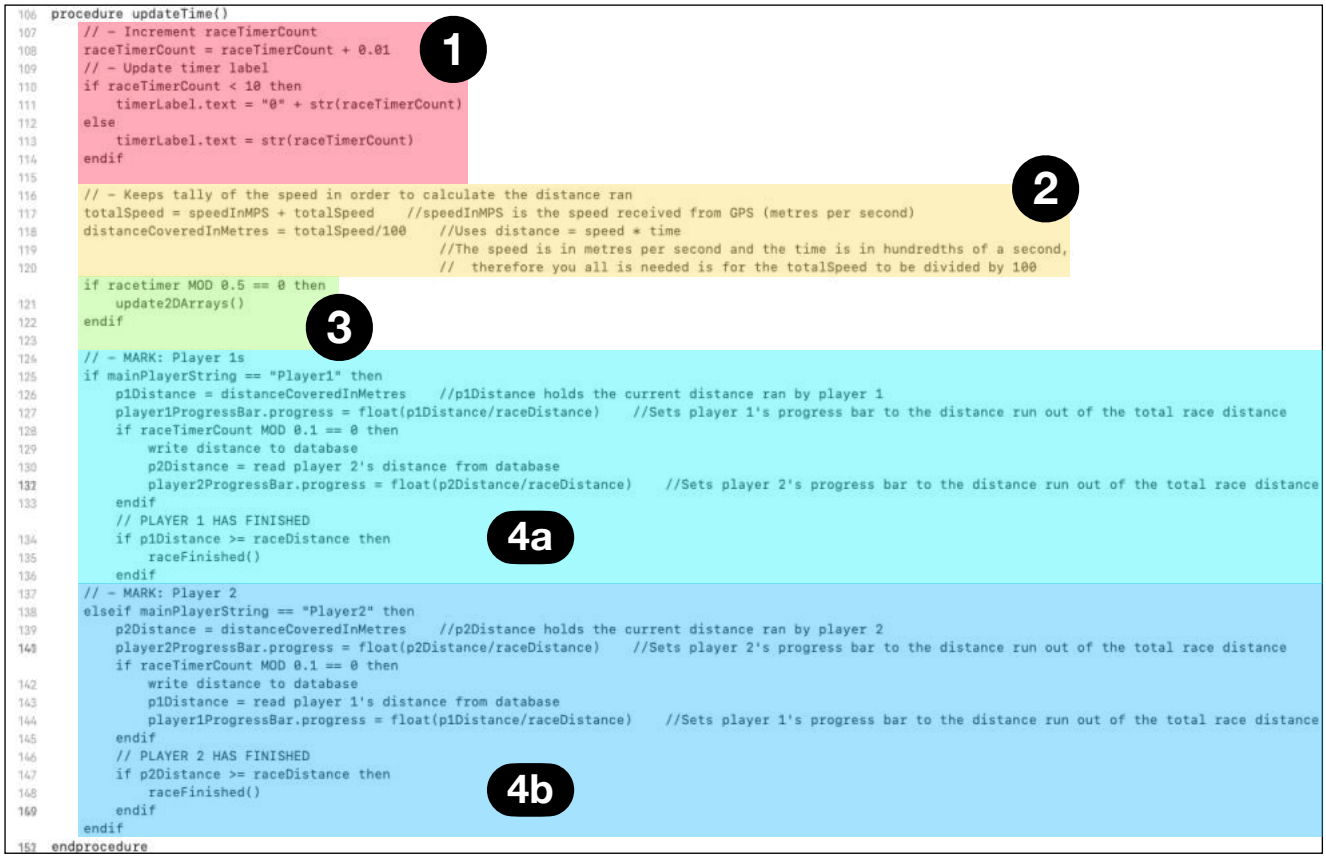
Version 6.8 - Creating the global variables

I created the following global variables because I will need to them lots in the next few sections:

```
//Race Data Class variables
// ****************************************
var speedDataArray: [[Double]] = [[], []]
var distanceDataArray: [[Double]] = [[], []]
var timeDataArray: [[Double]] = [[], []]

var finalTimeArray: [String] = ["", ""]
var averageSpeedArray: [Double] = [0.0, 0.0]
var topSpeedArray: [Double] = [0.0, 0.0]
// ****************************************

var speedInMPS = 0.0
var totalSpeed = 0.0
var distanceCoveredInMetres = 0.0

var raceDistance = 0.0
var distance = 0.0
var otherPlayerHasFinished = false

var p1Distance = 0.0
var p2Distance = 0.0
```

Version 6.9 - Race Timer

This is the main function of this screen. Looking back my pseudocode, I will code each separately, using the same technique as I used to code the keyboard (splitting my development into further sub-sections).




```
procedure updateTime()
    // - Increment raceTimerCount
    raceTimerCount = raceTimerCount + 0.01
    // - Update timer label
    if raceTimerCount < 10 then
        timerLabel.text = "0" + str(raceTimerCount)
    else
        timerLabel.text = str(raceTimerCount)
    endif

    // - Keeps tally of the speed in order to calculate the distance ran
    totalSpeed = speedInMPS + totalSpeed    //speedInMPS is the speed received from GPS (metres per second)
    distanceCoveredInMetres = totalSpeed/100    //Uses distance = speed * time
                                                //The speed is in metres per second and the time is in hundredths of a second,
                                                //  therefore you all is needed is for the totalSpeed to be divided by 100
    if racetimer MOD 0.5 == 0 then
        update2DArrays()
    endif

    // - MARK: Player 1s
    if mainPlayerString == "Player1" then
        p1Distance = distanceCoveredInMetres    //p1Distance holds the current distance ran by player 1
        player1ProgressBar.progress = float(p1Distance/raceDistance)    //Sets player 1's progress bar to the distance run out of the total race distance
        if raceTimerCount MOD 0.1 == 0 then
            write distance to database
            p2Distance = read player 2's distance from database
            player2ProgressBar.progress = float(p2Distance/raceDistance)    //Sets player 2's progress bar to the distance run out of the total race distance
        endif
        // PLAYER 1 HAS FINISHED
        if p1Distance >= raceDistance then
            raceFinished()
        endif
    // - MARK: Player 2
    elseif mainPlayerString == "Player2" then
        p2Distance = distanceCoveredInMetres    //p2Distance holds the current distance ran by player 2
        player2ProgressBar.progress = float(p2Distance/raceDistance)    //Sets player 2's progress bar to the distance run out of the total race distance
        if raceTimerCount MOD 0.1 == 0 then
            write distance to database
            p1Distance = read player 1's distance from database
            player1ProgressBar.progress = float(p1Distance/raceDistance)    //Sets player 1's progress bar to the distance run out of the total race distance
        endif
        // PLAYER 2 HAS FINISHED
        if p2Distance >= raceDistance then
            raceFinished()
        endif
    endif
endprocedure
```

Where I left the comment “*Start the race timer here*” in the updateCountdownTime function, I have called the new ‘startRaceTimer’ function. This function starts the creates the timer which calls the updateCountdownTime function every 0.01 seconds:

```
// - MARK: Race Timer
func startRaceTimer()
{
    //add a 0 to each stat array (because at time = 0 the user's speed and distance are both 0)
    speedDataArray[mainPlayerInt-1].append(0)
    distanceDataArray[mainPlayerInt-1].append(0)
    timeDataArray[mainPlayerInt-1].append(0)
    raceTimerCount = 0
    //creates a timer which calls the function updateCountdownTime() every 0.01 second:
    raceTimer = Timer.scheduledTimer(timeInterval: 0.01, target: self, selector: (#selector(RaceViewController.updateRaceTime)), userInfo: nil, repeats: true)
}

@objc func updateRaceTime()
{

}
```

For each of the following versions the code is being added to the updateRaceTime function:

Version 6.9.1

The following code is very straightforward - it increments the raceTimerCount global variable and then updates the timer label:

```
raceTimerCount = raceTimerCount + 0.01   //increments the raceTimerCount by 0.01
raceTimerCount = round(raceTimerCount * 100) / 100   //rounds it to 2.dp (because floats can sometimes become recurring)

//updates the timer label
var timerText = String(format: "%.2f", raceTimerCount)
if (raceTimerCount < 10)
{
    timerText.insert("0", at: timerText.startIndex)  //inserts a '0' at the beginning of the string
}
timerLabel.text = timerText
```

Version 6.9.2

This addition is also very straightforward:

```
totalSpeed += speedInMPS
distanceCoveredInMetres = (Double (totalSpeed) * 1000).rounded() / 100000   //rounds the distance to 2dp
```

Version 6.9.3

```
//Every 0.5 seconds update the stats arrays
if (remainderOfDoubles(x: raceTimerCount, y: 0.5) == 0)
{
    update2DArrays()
}
```

The update2DArrays function essentially adds the speed, distance and time at the current time to the end of each list:

```
func update2DArrays()
{
    speedDataArray[mainPlayerInt-1].append(speedInMPS)
    distanceDataArray[mainPlayerInt-1].append(distanceCoveredInMetres)
    timeDataArray[mainPlayerInt-1].append(raceTimerCount)
}
```

Version 6.9.4

Here is the code for the blue highlighted pseudocode (part 4a and 4b):

```
//Player 1's
if (mainPlayerString == "Player1")
{
    p1Distance = distanceCoveredInMetres
    player1ProgressBar.progress = Float(p1Distance/raceDistance)    //progress bar set to proportion of race distance covered
    //Every 0.2 seconds:
    if (remainderOfDoubles(x: raceTimerCount, y: 0.2) == 0)
    {
        //read player2's distance and update the progress bar
        p2Distance = readDistance()
        player2ProgressBar.progress = Float(p2Distance/raceDistance)

        writeDistance()    //write player 1's distance to the database
    }
    // PLAYER 1 HAS FINISHED
    if (p1Distance >= raceDistance)
    {
        //player 1 has finished the race
    }
}
//Player 2's
else if (mainPlayerString == "Player2")
{
    p2Distance = distanceCoveredInMetres
    player2ProgressBar.progress = Float(p2Distance/raceDistance)    //progress bar set to proportion of race distance covered
    //Every 0.2 seconds:
    if (remainderOfDoubles(x: raceTimerCount, y: 0.2) == 0)
    {
        //read player1's distance and update the progress bar
        p1Distance = readDistance()
        player1ProgressBar.progress = Float(p1Distance/raceDistance)

        writeDistance()    //write player 2's distance to the database
    }
    // PLAYER 2 HAS FINISHED
    if (p2Distance >= raceDistance)
    {
        //player 2 has finished the race
    }
}
```

Here is the readDistance function. As you can it returns the distance (as a Double) the opponent has run at that point in time, by using the opponentPlayerString variable to access that players document in the database:

```
//returns the distance the opponent has run at that point in time:
func readDistance () -> Double
{
    let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(opponentPlayerString)
    documentRef.getDocument { (document, error) in
        if let document = document, document.exists {
            let dataDescription = document.data()
            self.distance = dataDescription!["playerDistance"]! as! Double
        } else {
            print("Document does not exist")
        }
    }
    return distance
}
```

Here is the writeDistance function. It updates the distance which the user has run at that point in time, by using the mainPlayerString variable to access the user's document in the database:

```
//writes the distance the user has run at that point in time:
func writeDistance()
{
    let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(mainPlayerString)
    documentRef.updateData([
        "playerDistance": distanceCoveredInMetres
    ]) { err in
        if let err = err {
            print("Error updating document: \(err)")
        } else {
            //print("Document successfully updated")
        }
    }
}
```

Version 6.10 - Race Finished

Here's my pseudocode for the raceFinished function. I now need to implement this in Swift 4:

```
procedure raceFinished()
    finalTimeArray[mainPlayerInt-1] = str(raceTimerCount)
    stop raceTimer
    calulateAverageSpeed()
    calculateTopSpeed()
    write final stats to the database
    if checkPlayerHasFinished() == false then
        add a snapshot listener which will do the following when the 'hasFinished' field in the Player2 document in the database is true{
            read other players final statistics from the database
            stop waitForOtherPlayerTimer
            transition screens to display the final times and statistics
        }
        waitForOtherPlayerTimer = Timer which calls the function updateTimeOtherPlayer() every 0.2 seconds
    else
        read other players final statistics from the database
        transition screens to display the final times and statistics
    endif
endprocedure
```

```
// - MARK: Race Finished
func raceFinished()
{
    finalTimeArray[mainPlayerInt-1] = String(raceTimerCount)    //adds the user's time to the user's index of the finalTimeArray
    print ("Time: " + finalTimeArray[mainPlayerInt-1])
    playVibrate()
    raceTimer.invalidate()     //stops the raceTimer
    calulateAverageSpeed ()
    calculateTopSpeed ()
    writeFinalStats()     //write the user's final stats to the database
    if (checkPlayerHasFinished() == false)
    {
        waitTillPlayerHasFinished()
        startWaitForOtherPlayerTimer()
    }
    else
    {
        readFinalStatistics()
    }
}
```

I am going to start going through this function line by line, as many functions are called from this function which each need to be explained.

Firstly, on lines 295 and 296, the calculateAverageSpeed and calculateTopSpeed functions are called. The results of the calculations are stored in the 2d arrays:

```
// - MARK: Calculations
func calulateAverageSpeed ()
{
    let count = speedDataArray[mainPlayerInt-1].count * 50  //times by 50 because the array is updated every 0.2 seconds

    averageSpeedArray[mainPlayerInt-1] = (totalSpeed / Double(count)).roundTo(places: 2)
    print ("Average Speed: " + String (averageSpeedArray[mainPlayerInt-1]))
}

func calculateTopSpeed ()
{
    topSpeedArray[mainPlayerInt-1] = Double(speedDataArray[mainPlayerInt-1].max()!).roundTo(places: 2)
    print ("Top Speed: " + String (topSpeedArray[mainPlayerInt-1]))
}
```

On the next line, the writeFinalStats function is called. It utilises the mainPlayerString and mainPlayerInt variables to update the fields in that user's document in the database:

```
func writeFinalStats()
{
    //references the mainPlayerString's document
    let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(mainPlayerString)

    //updates the fields to the user's data in the arrays using the mainPlayerInt variable
    documentRef.updateData(["finalTime": finalTimeArray[mainPlayerInt-1],
                            "averageSpeed":averageSpeedArray[mainPlayerInt-1],
                            "topSpeed":topSpeedArray[mainPlayerInt-1],
                            "speedDataArray": speedDataArray[mainPlayerInt-1],
                            "distanceDataArray": distanceDataArray[mainPlayerInt-1],
                            "timeDataArray": timeDataArray[mainPlayerInt-1],
                            "hasFinished":true
        ])
    { err in
        if let err = err {
            print("Error updating document: \(err)")
        } else {
            print("Document successfully updated")
        }
    }
}
```

The next line, line 298, calls the function checkPlayerHasFinished to check if the opponent has finished the race. The function reads the 'hasFinished' field of the opponents document in the database and returns either true or false:

```
    func checkPlayerHasFinished () -> Bool
    {
        let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(opponentPlayerString)

        documentRef.getDocument { (document, error) in
            if let document = document, document.exists {
                let dataDescription = document.data()
                self.otherPlayerHasFinished = dataDescription!["hasFinished"]! as! Bool
            } else {
                print("Document does not exist")
            }
        }
        return otherPlayerHasFinished
    }
```

If the checkPlayerHasFinished function returns true (i.e. the other player has finished the race), then the other player's final statistics are read:

```
func readFinalStatistics ()
{
    //references the opponents's document
    let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(opponentPlayerString)

    //reads the opponents's document's fields and stores the results in the corresponding arrays
    documentRef.getDocument { (document, error) in
        if let document = document, document.exists {
            let dataDescription = document.data()
            self.finalTimeArray[self.opponentPlayerInt-1] = dataDescription!["finalTime"]! as! String
            self.averageSpeedArray[self.opponentPlayerInt-1] = dataDescription!["averageSpeed"]! as! Double
            self.topSpeedArray[self.opponentPlayerInt-1] = dataDescription!["topSpeed"]! as! Double
            self.speedDataArray[self.opponentPlayerInt-1] = dataDescription!["speedDataArray"] as! [Double]
            self.distanceDataArray[self.opponentPlayerInt-1] = dataDescription!["distanceDataArray"] as! [Double]
            self.timeDataArray[self.opponentPlayerInt-1] = dataDescription!["timeDataArray"] as! [Double]
            print("SUCESS READING FINAL STATISTICS")
            //*change screen here*
        } else {
            print("Document does not exist")
        }
    }
}
```

However, if this returns false, as discussed in my design section I need to add a snapshot listener which will notify the user when the other player has finished the race, i.e. when the 'hasFinished' field in the opponent document is set to true. However, at the same time the opponents progress bar should also be continue to be constantly updated. That is why the following two functions are called:

```
waitTillPlayerHasFinished()
startWaitForOtherPlayerTimer()
```

```
    func waitTillPlayerHasFinished ()
    {
        //references the opponents's document
        let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(opponentPlayerString)

        //Adds a listener to this document
        documentRef.addSnapshotListener { documentSnapshot, error in
          guard let document = documentSnapshot else {
            print("Error fetching document: \(error!)") //error
            return
          }
          guard let data = document.data() else {
            print("Document data was empty.")  //document is empty
            return
          }
            if (data["hasFinished"] as! Bool == true)   //if the field 'hasFinished' is set to true, both players have finished
            {
                self.readFinalStatistics()  //read the opponents final statistics
                self.waitForOtherPlayerTimer.invalidate()  //stops the waitForOtherPlayerTimer
            }
        }
    }

func startWaitForOtherPlayerTimer()
{
    //creates a timer which calls the function updateTimeOtherPlayer() every 0.2 seconds:
    waitForOtherPlayerTimer = Timer.scheduledTimer(timeInterval: 0.2, target: self, selector: (#selector(RaceViewController.updateTimeOtherPlayer)),
        userInfo: nil, repeats: true)
}

@objc func updateTimeOtherPlayer ()
{
    //if user is player 1, update player 2's progress bar
    if (mainPlayerString == "Player1")
    {
        p2Distance = readDistance()
        player2ProgressBar.progress = Float(p2Distance/raceDistance)
    }
    //if user is player 2, update player 1's progress bar
    else
    {
        p1Distance = readDistance()
        player1ProgressBar.progress = Float(p1Distance/raceDistance)
    }
}
```

So essentially this is happening:

- Firstly, the snapshot listener is added to the opponents document.
- Until the ‘hasFinished’ field has turned to true, the opponent’s progress bar is being updated every 0.2 seconds through the updateTimeOtherPlayer function.
- When the opponent’s ‘hasFinished’ field becomes true, the opponents final statistics are read and stored in the corresponding arrays and then the waitForOtherPlayerTimer is stopped.

This concludes the raceFinished function and in fact the RaceScreen (except for the transition to the next screen). I will test it in the evaluation section because it is impossible for me to test on the simulator and on my phone at the same time. I will need to download the app onto my brothers phone and race against him in order to extensively test the app.

Version 6 - Review

What has been done

The race screen has been created. It has a map in the background which shows the user's location in the centre. As a user of any app, security and privacy is very important. This is why when the user reaches this screen for the first time, they are informed that their location needs to be accessed using GPS and they have a choice of whether they want to allow this.

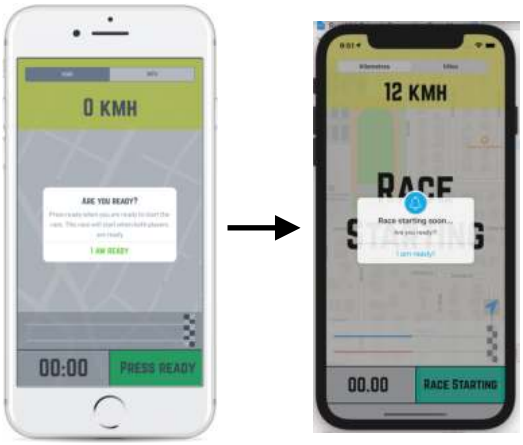
I have written code so that when the user loads on to this screen, a pop-up will be shown, and the user can press the 'ready' button if they are ready to start the race.

I have also written code which handles what happens during the race, such as updating the race timer, updating the statistics and updating the progress bars.

Finally, certain algorithms are in place which handle what happens when the user finishes the race. For example, if the user finishes before the other player, then they must wait for the other player to finish the race. Meanwhile whilst they are waiting, all their final stats are written to the database and then the other player's progress bar must still continue to be updated.

What has changed from the original design

Not much at all has changed from my original design. One small change is instead of using the inbuilt iOS default alert to display the 'ready' alert, I used the CDAlertView to create the alert. This is because it is more colourful and more customisable, so hence more appropriate for my solution.



What has been tested

I have tested that the map works and that the user's location can be updated. I also tested that the user's speed was being calculated accurately from the GPS by comparing it to average running speeds I found online.

I also tested that the ready alert feature was working sound.

Apart from this, I have not tested the anything else. This is because as I have mentioned, it is impossible for me to test my app on the simulator and on my phone at the same time. I will test my app in the evaluation phase, because I will need to download the app onto my brothers phone and race against him in order to extensively test the app.

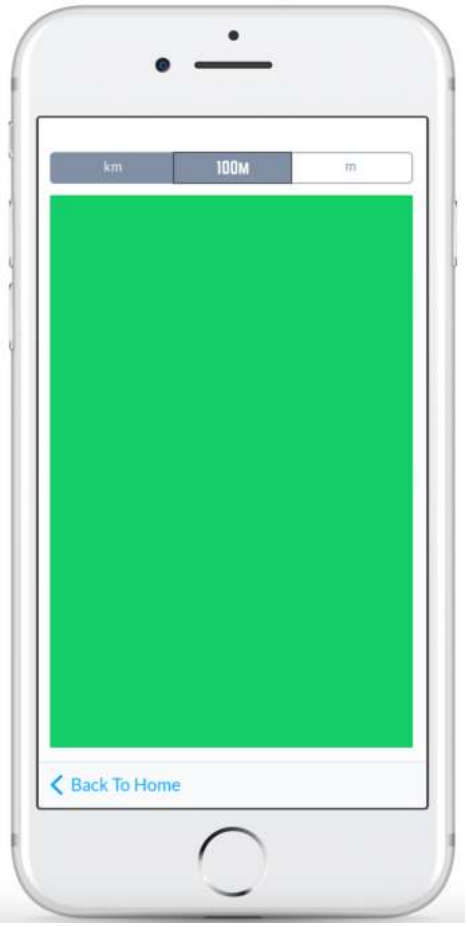
Version 7 - Building the stats screen

Version 7.1 - Creating the RaceData class

I created a new swift file and called it ‘RaceData’. This class mirrors my pseudocode on page 47:

```
class raceData
{
    static let sharedInstance = raceData()

    var speedDataArray: [[Double]] = [[], []]       //2D arrays to hold each players speed over the course of the race
    var distanceDataArray: [[Double]] = [[], []]      //2D arrays to hold each players distance over the course of the race
    var timeDataArray: [[Double]] = [[], []]      //2D arrays to hold each players time over the course of the race

    var finalTimeArray: [String] = ["", ""]       //2D arrays to hold each players final time
    var averageSpeedArray: [Double] = [0.0, 0.0]       //2D arrays to hold each players average speed
    var topSpeedArray: [Double] = [0.0, 0.0]       //2D arrays to hold each players top speed

    var raceDistance: Int = 0      //integer to hold the race distance
    var playerString: String = ""      //string to hold the current player: "player1" or "player2"
    var playerInt: Int = 0      //integer to hold the current player: 1 or 2

    var isKilometres: Bool = true      //boolean which determines whether the stats should be shown in km or m.
}
```

Version 7.2 - Creating the HomeStats Screen



As I have already discussed for the stats screen screen, I will be making use of a pageViewController, so that the user can swipe and see different stats and graphs.

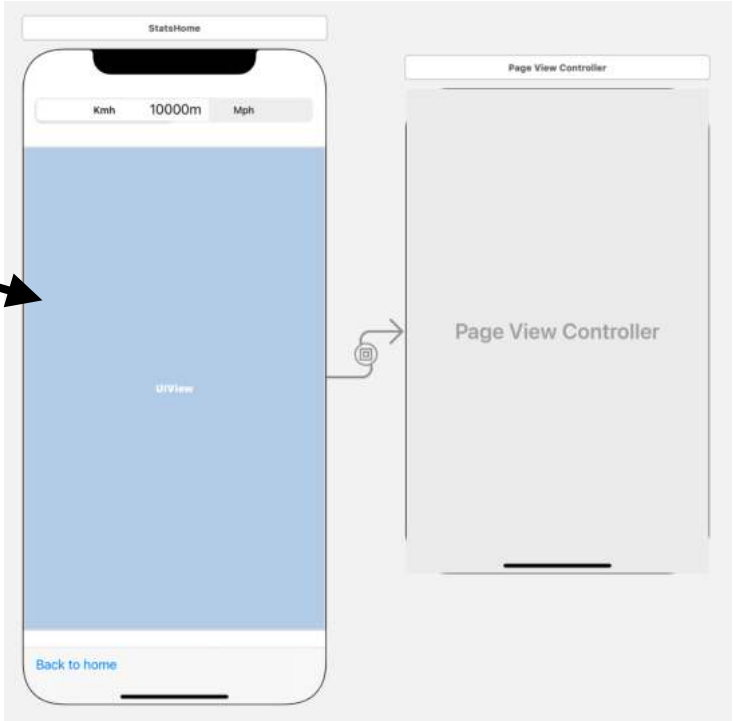
My plan is to embed the pageViewController into the main statsScreen which I will refer to as the HomeStats screen. The design on the right shows the HomeStats screen:

The large green rectangle represents the placeholder for the pageViewController. The user will be able to swipe in this area, whilst the rest of the screen stays the same.

At the top of this screen, there is a segmented control so that the user can change the stats from km/h to mph. In the centre of the segmented control, the race distance will be displayed.

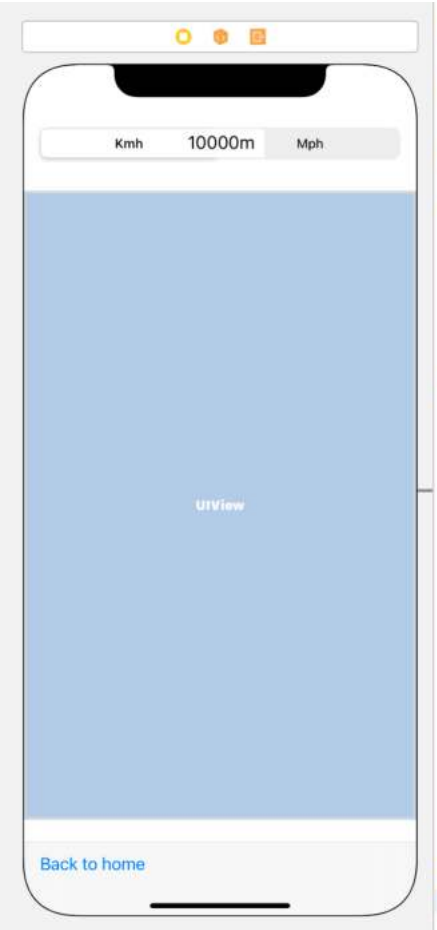
The user should also be allowed to return to the home screen by pressing a button in the toolbar.

Here is the screen I designed on the mainStoryboard file. As you can see I have added a container view and embedded in this view is the PageViewController manager.



I then connected this screen up to the new swift file named ‘statsHomeViewController’.

I then connected the screen up to the swift file and added the following code which sets up the screen:




```swift
//
//  StatsHomeViewController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 05/04/2021.
//

import Foundation
import UIKit

class statsHomeScreenController: UIViewController
{
    @IBOutlet weak var speedSegment: UISegmentedControl!
    @IBOutlet weak var distanceDisplayView: UIView!
    @IBOutlet weak var distanceDisplayLabel: UILabel!

    override func viewDidLoad()
    {
        super.viewDidLoad()

        distanceDisplayView.layer.cornerRadius = 5
        //gets the distance from the RaceData class to display the race
            distance at the top of the screen
        distanceDisplayLabel.text =
            String(RaceData.sharedInstance.raceDistance) + "m"
    }

    @IBAction func speedSegmentChanged(_ sender: Any)
    {
        //set to kilometres
        if (speedSegment.selectedSegmentIndex == 0)
        {
            RaceData.sharedInstance.isKilometres = true
        }
        //set to miles
        else if (speedSegment.selectedSegmentIndex == 1)
        {
            RaceData.sharedInstance.isKilometres = false
        }
    }
}
```

Finally, to finish off the HomeStats screen, I added the following code to transition the user back to the Home screen when the ‘back to home screen’ button is pressed:

```swift
@IBAction func backToHomeButtonPressed(_ sender: Any)
{
    let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
    if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier: "HomeScreenViewConroller") as? HomeScreenViewConroller {
        self.present(destinationViewController, animated: true, completion: nil)
    }
}
```

Version 7.3 - Setting up the PageViewController

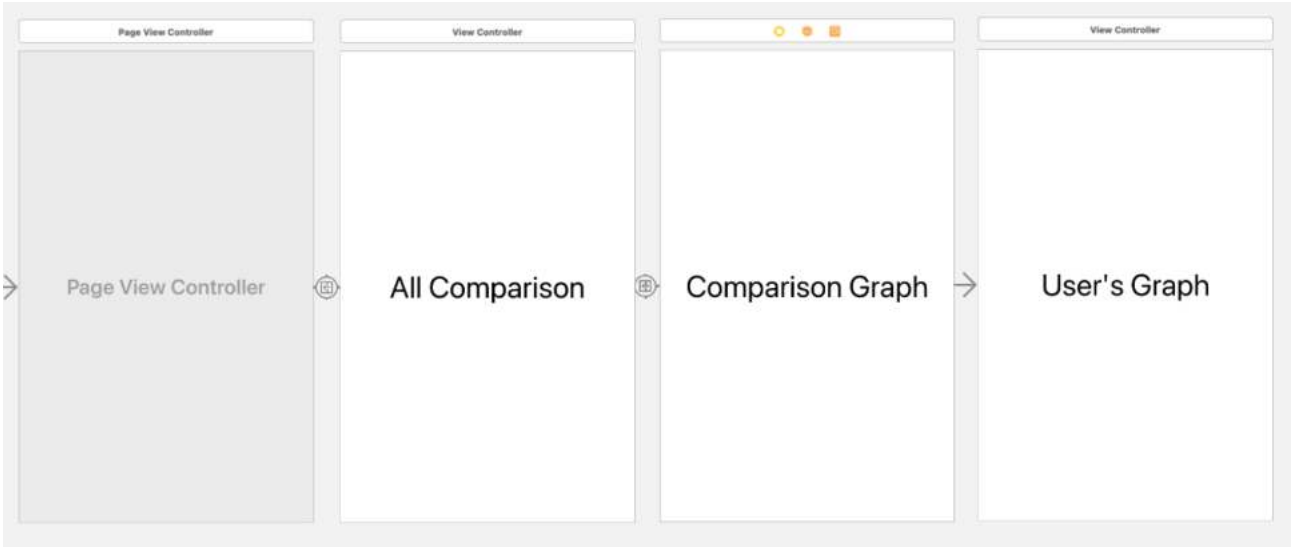
I will need a Swift class to which I can link the PageViewController so that it can be configured.

Before I do this, I need to know which screens will be embedded in the PageViewController:

There are three screens which I plan for the user to be able to see:

1) A screen which compares all the users stats
2) A speed-distance graph, which is plotted for both users so that they can draw comparisons
3) A speed-distance graph for only the user.

I created these screens in my MainStoryboard file and I also created their corresponding files. I have also added a label to each view so that I can identify which screen is which when it comes to testing, because my main focus at the moment is making sure the PageViewController is working so that I can start working on the other screens.



Now that this is all set up, I can now start programming the PageViewController.

The next page shows the code for the PageViewController. It is not that important in terms of my overall aim or remotely interesting, so I will not go through it in detail, but essentially it is responsible for handling things like the user swipes, handling the pages which the user can transition to, and handling the page control (the dots at the bottom of the page view which indicate which page the user is on).

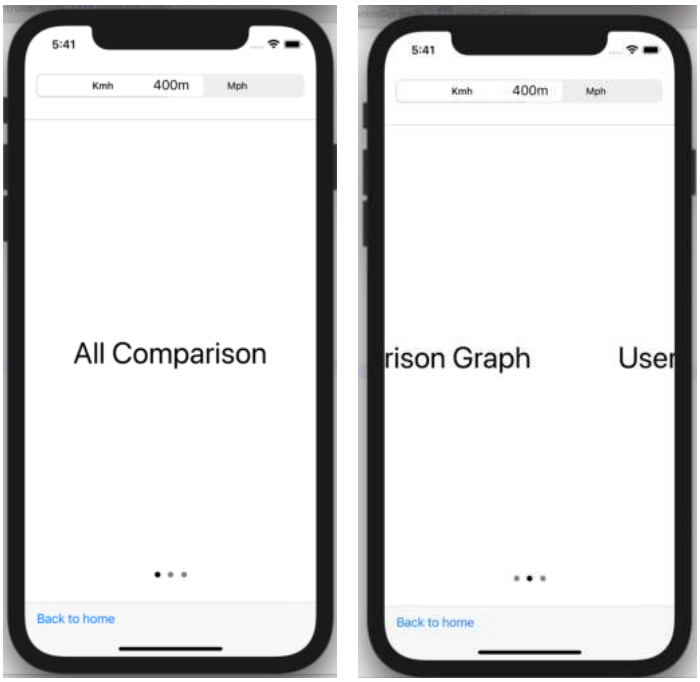
I have summarised with comments what each function is responsible for:

```swift
class PageViewController: UIPageViewController, UIPageViewControllerDelegate, UIPageViewControllerDataSource {

    //initiliases the 3 screens which will be part of the page view controller
    lazy var orderedViewControllers: [UIViewController] = {

        let sb = UIStoryboard(name: "Main", bundle: nil)

        let vc1 = sb.instantiateViewController(withIdentifier: "AllStatsComparisonViewController")
        let vc2 = sb.instantiateViewController(withIdentifier: "SpeedComparisonViewController")
        let vc3 = sb.instantiateViewController(withIdentifier: "UsersSpeedViewController")

        return [vc1, vc2, vc3]
    }()

    var pageControl = UIPageControl()

    //when the view appears, set up the page view controller
    override func viewDidAppear(_ animated: Bool)
    {
        self.dataSource = self
        self.delegate = self
        configurePageControl()
        if let firstViewController = orderedViewControllers.first
        {
            setViewControllers([firstViewController], direction: .forward, animated: true, completion: nil)
        }
    }

    //configures the 3 dots which indicate which page the user is on
    func configurePageControl()
    {
        pageControl = UIPageControl(frame: CGRect(x: 0, y: self.view.bounds.maxY - 40, width: UIScreen.main.bounds.width, height: 50))
        pageControl.numberOfPages = orderedViewControllers.count
        pageControl.currentPage = 0
        pageControl.tintColor = UIColor.black
        pageControl.pageIndicatorTintColor = UIColor.gray
        pageControl.currentPageIndicatorTintColor = UIColor.black
        self.view.addSubview(pageControl)
    }
    //code for what happens if the user swipes right (backwards)
    func pageViewController(_ pageViewController: UIPageViewController, viewControllerBefore viewController: UIViewController) -> UIViewController?
    {
        guard let viewControllerIndex = orderedViewControllers.index(of: viewController) else
        {
            return nil
        }

        let previousIndex = viewControllerIndex - 1

        //if there is no previous page, then do nothing
        guard previousIndex >= 0 else
        {
            return nil
        }
        guard orderedViewControllers.count > previousIndex else
        {
            return nil
        }

        //if there is a previous page, then show this page
        return orderedViewControllers[previousIndex]
    }

    //code for what happens if the user swipes left (forwards)
    func pageViewController(_ pageViewController: UIPageViewController, viewControllerAfter viewController: UIViewController) -> UIViewController?
    {
        guard let viewControllerIndex = orderedViewControllers.index(of: viewController) else
        {
            return nil
        }

        let nextIndex = viewControllerIndex + 1

        //if there is no next page, then do nothing
        guard orderedViewControllers.count != nextIndex else
        {
            return nil
        }
        guard orderedViewControllers.count > nextIndex else
        {
            return nil
        }

        //if there is a next page, then show this page
        return orderedViewControllers[nextIndex]
    }

    //displays the current page with a transition
    func pageViewController(_ pageViewController: UIPageViewController, didFinishAnimating finished: Bool, previousViewControllers: [UIViewController], transitionCompleted
        completed: Bool)
    {
        let pageContentViewController = pageViewController.viewControllers![0]
        self.pageControl.currentPage = orderedViewControllers.index(of: pageContentViewController)!
    }
}
```

Before I could test out the PageViewController, I had to add the following code so that I could set the StatsHome screen to the initial screen so that I wouldn't have to run a whole race every-time I wanted to test out the stats screen. The code essentially fills out the RaceData class so that at this point in the app, it is as if a race has just concluded.

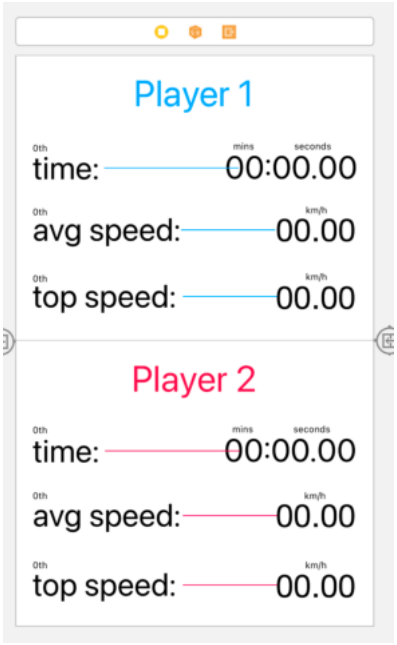
```
override func viewDidLoad()
{
    super.viewDidLoad()

    RaceData.sharedInstance.speedDataArray = [[0.1, 0.4, 3, 5, 7], [0.04, 2.3, 5.7]]
    RaceData.sharedInstance.timeDataArray = [[1, 2, 3], [1, 2, 3]]
    RaceData.sharedInstance.distanceDataArray = [[1, 2, 3, 4, 20], [1, 2, 3]]
    RaceData.sharedInstance.finalTimeArray = ["10.0", "11.5"]
    RaceData.sharedInstance.averageSpeedArray = [3, 4]
    RaceData.sharedInstance.topSpeedArray = [8, 5]
    RaceData.sharedInstance.raceDistance = 400
    RaceData.sharedInstance.playerInt = 1
    RaceData.sharedInstance.playerString = "Player1"

    print("speedDataArray: \(RaceData.sharedInstance.speedDataArray)")
    print("distanceDataArray: \(RaceData.sharedInstance.distanceDataArray)")
    print("timeDataArray: \(RaceData.sharedInstance.timeDataArray)")
    print("finalTimeArray: \(RaceData.sharedInstance.finalTimeArray)")
    print("averageSpeedArray: \(RaceData.sharedInstance.averageSpeedArray)")
    print("topSpeedArray: \(RaceData.sharedInstance.topSpeedArray)")
    print("raceDistance: \(RaceData.sharedInstance.raceDistance)")
    print("player: \(RaceData.sharedInstance.playerString)")

    distanceDisplayView.layer.cornerRadius = 5
    //gets the distance from the RaceData class to display the race distance at the top of the screen
    distanceDisplayLabel.text = String(RaceData.sharedInstance.raceDistance) + "m"
}
```

When I ran my app everything was successful. The first screen shown is the AllComparison screen and I could successfully transition between two screens in the PageView by swiping. When I reached the end, I was not able to swipe anymore.



Now I can start working on the three screens which are in the PageView controller.

Version 7.4 - Creating the AllStatsComparison screens

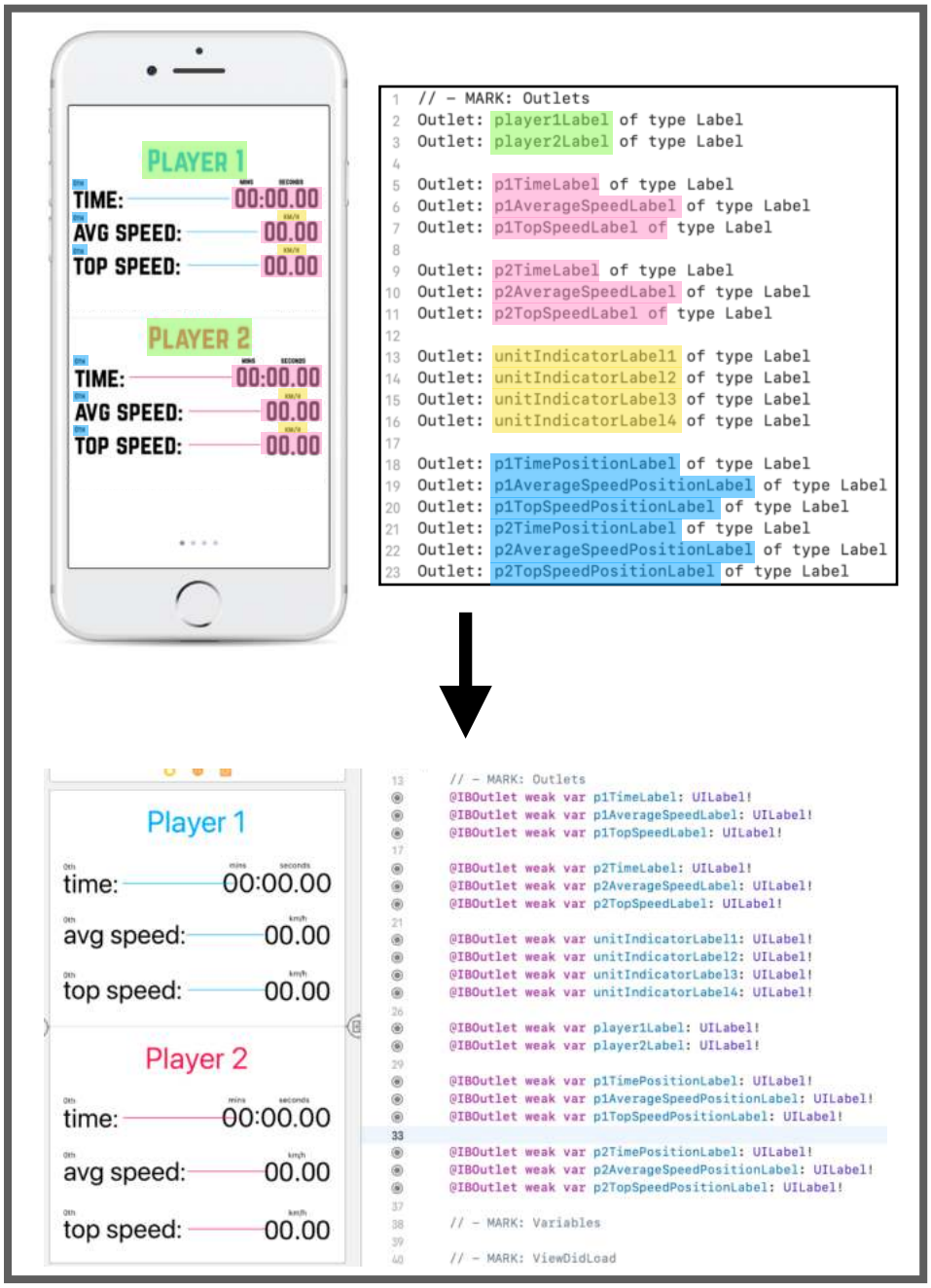


Version 7.4.1 - Creating the MainStoryboard screen

I created this design. I have followed the colour scheme of blue (player 1) and red (player 2):

Version 7.4.2 - Connecting up the components to the swift file

Everything now needs to be connected up to the swift file. The diagram below shows how my pseudocode helped me to connect the components on the screen to the swift file.



Version 7.4.3 - Coding the algorithms

Here is my code.

```
override func viewDidLoad()
{
    super.viewDidLoad()

    //shows each player's time
    p1TimeLabel.text = formatTime(timeLet: RaceData.sharedInstance.finalTimeArray[0])
    p2TimeLabel.text = formatTime(timeLet: RaceData.sharedInstance.finalTimeArray[1])

    //shows each player 1's average and top speed (in kilometres)
    p1AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[0] * 3.6).roundTo(places: 2)))
    p1TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[0] * 3.6).roundTo(places: 2)))

    //shows each player 2's average and top speed (in miles)
    p2AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[1] * 3.6).roundTo(places: 2)))
    p2TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[1] * 3.6).roundTo(places: 2)))

    //If the user is player 1, then the player 1 label should read 'You' and the player 2 label should read 'opponent'
    if (RaceData.sharedInstance.playerInt == 1)
    {
        player1Label.text = "You"
        player2Label.text = "Opponent"
    }
    //If the user is player 2, then the player 2 label should read 'You' and the player 1 label should read 'opponent'
    else
    {
        player2Label.text = "You"
        player1Label.text = "Opponent"
    }

    highlightWinners()
}
```

First of all, when the screen loads, each player's final time and speeds are shown. The times are passed through a function called formatTime and the speeds are passed through a function called formatSpeeds:

```
//function which takes in a speed and returns the newly formatted speed
func formatSpeeds (speedLet: String) -> String
{
    var speed = speedLet
    let speedString = speed
    //If the speed is less than 10, add a 0 to the beggining of the speed:
    //e.g. 9.25kmh -> 09.25kmh
    if (Double(speed)! < 10)
    {
        speed = "0" + speed
    }
    //If the speed is divisible by 0.1, add a 0 to the end of the speed:
    //e.g. 3.2kmh -> 3.20kmh
    if(remainderOfDoubles(x: Double(speedString)!, y: 0.1) == 0)
    {
        speed = speed + "0"
    }
    return speed
}
```

```
//function which takes in a time and returns the newly formatted time
func formatTime (timeLet: String) -> String
{
    var time = timeLet
    var mins = 0
    var seconds = 0.0
    var minsString = ""
    //If the time in seconds is less than 10, add a 0 to the beggining of the time:
    //e.g. 8.35s -> 08.35s
    if (Double(time)! < 10)
    {
        time = "0" + time
    }
    mins = Int((Double(time)!/60).rounded(.down))   //works out the minutes from
        the time
    seconds = (Double(time)! - (Double(mins)*60)).roundTo(places: 2)    //works out
        the seconds remaining from the time and minutes
    //If the minutes is less than 10, add a 0 to the beggining of the minutes:
    //e.g. 8 minutes -> 08 minutes
    if (mins < 10)
    {
        minsString = "0" + String(mins)
    }
    else
    {
        minsString = String(mins)
    }
    //If the seconds is less than 10, add a 0 to the beggining of the seconds part
        of the string:
    //e.g. 12:4.83 -> 12:04.83     ~ 12 minutes, 4 seconds and 83 miliseconds
    if (seconds < 10)
    {
        time = String(minsString) + ":0" + String(seconds)
    }
    else
    {
        time = String(minsString) + ":" + String(seconds)
    }
    //If the seconds is divisible by than 0.1, add a 0 to the end of the time part
        of the string:
    //e.g. 03:45.6 -> 03:45.60    ~ 3 minutes, 45 seconds and 60 miliseconds
    if(remainderOfDoubles(x: seconds, y: 0.1) == 0)
    {
        time = time + "0"
    }
    return time
}
```

Next, the player1Label and player2Label's texts are labelled correctly - displaying either 'You' or 'Opponent', depending on whether the user is player 1 or player 2.

After this the highlightWinners function is called. It is the same function as the one I designed in my pseudocode:

```
//Organisational function to highlight the winners of each stat
func highlightWinners()
{
    //Compares the times of each player - it is handled differently because you are checking for a lower (faster) time.
    if (Double(RaceData.sharedInstance.finalTimeArray[0])! < Double(RaceData.sharedInstance.finalTimeArray[1])!)
    {
        playerWins(winnerLabel: p1TimePositionLabel, loserLabel: p2TimePositionLabel)
    }
    else
    {
        playerWins(winnerLabel: p2TimePositionLabel, loserLabel: p1TimePositionLabel)
    }

    //Calls the function to compare the two average speeds
    compareTwoSpeeds(arr: RaceData.sharedInstance.averageSpeedArray, p1Label: p1AverageSpeedPositionLabel, p2Label: p2AverageSpeedPositionLabel)

    //Calls the function to compare the two top speeds
    compareTwoSpeeds(arr: RaceData.sharedInstance.topSpeedArray, p1Label: p1TopSpeedPositionLabel, p2Label: p2TopSpeedPositionLabel)
}
```

This functions calls the compareTwoSpeeds:

```
//Compares two speeds and calls the playerWins function depending on which speed is greater
func compareTwoSpeeds(arr: [Double], p1Label: UILabel, p2Label: UILabel)
{
    //If player 1's speed is greater than player 2's speed: player1 is the winnner, player2 is the loser.
    if (arr[0] > arr[1])
    {
        playerWins(winnerLabel: p1Label, loserLabel: p2Label)
    }
    //If player 2's speed is greater than player 1's speed: player2 is the winnner, player1 is the loser.
    else if (arr[1] > arr[0])
    {
        playerWins(winnerLabel: p2Label, loserLabel: p1Label)
    }
}
```

The compareTwoSpeeds function above compares the two indexes (which will be speeds) in the inputted array. The playerWins function is then called, with the arguments depending on who's speed is larger, player 1 or player 2's:

```
//Handles the formatting for the position labels
func playerWins(winnerLabel: UILabel, loserLabel: UILabel)
{
    //Loser's string: says '2nd' and has a silver background
    let myString = "2nd"
    let myAttribute = [ NSAttributedString.Key.backgroundColor: UIColor.init(red: 0/255, green: 0/255, blue: 0/255, alpha: 0.25)]
    let losingString = NSAttributedString(string: myString, attributes: myAttribute)

    //Winners's string: says '1st' and has a gold background
    let myString2 = "1st"
    let myAttribute2 = [ NSAttributedString.Key.backgroundColor: UIColor.init(red: 255/255, green: 211/255, blue: 0/255, alpha: 0.9)]
    let winningString = NSAttributedString(string: myString2, attributes: myAttribute2)

    winnerLabel.attributedText = winningString  //the winner label's text is set to the 'winningString'
    loserLabel.attributedText = losingString  //the losers label's text is set to the 'winningString'
}
```

The playerWins function essentially sets the winners label to read '1st' with a gold background colour and the losers label to read '2nd' with a silver background colour.

Version 7.4.3 - Handling the km vs miles segment

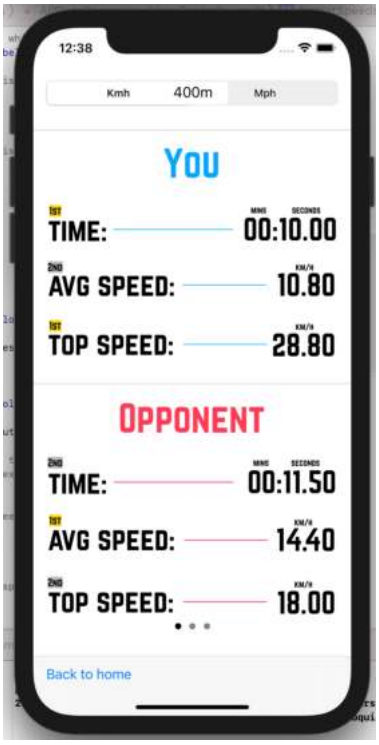
The following function is triggered when the user switches the segment. If ‘kmh’ is selected then the speeds are converted into kmh. If ‘mph’ is selected the speeds are converted to mph.

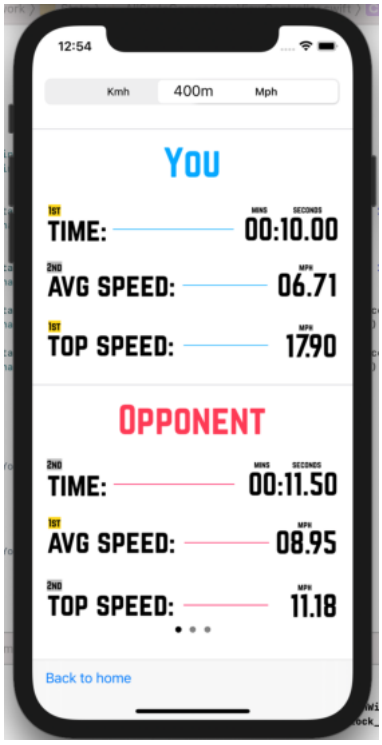
```
@IBAction func segmentValueChanged(_ sender: Any)
{
    //km selected - multiply speeds by 3.6 to get from metres per second to kilometres per hour
    if (segment.selectedSegmentIndex == 0)
    {
        p1AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[0] * 3.6).roundTo(places: 2)))
        p1TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[0] * 3.6).roundTo(places: 2)))

        p2AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[1] * 3.6).roundTo(places: 2)))
        p2TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[1] * 3.6).roundTo(places: 2)))

        unitIndicatorLabel1.text = "km/h"
        unitIndicatorLabel2.text = "km/h"
        unitIndicatorLabel3.text = "km/h"
        unitIndicatorLabel4.text = "km/h"
    }
    //miles selected - multiply speeds by 2.23694 to get from metres per second to miles per hour
    else if (segment.selectedSegmentIndex == 1)
    {
        p1AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[0] * 2.23694).roundTo(places: 2)))
        p1TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[0] * 2.23694).roundTo(places: 2)))

        p2AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[1] * 2.23694).roundTo(places: 2)))
        p2TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[1] * 2.23694).roundTo(places: 2)))

        unitIndicatorLabel1.text = "mph"
        unitIndicatorLabel2.text = "mph"
        unitIndicatorLabel3.text = "mph"
        unitIndicatorLabel4.text = "mph"
    }
}
```

Version 7.4.4 - Testing

This is what my app looked like when run. As you can see, the correct winners for each stat have been calculated and the position labels have been formatted correctly. The correct labels have been shown for the player1Label and player2Label - as with this test the playerInt in the RaceData class has been set to 1, therefore the user is player 1 and so the blue text reads ‘you’.

Furthermore, the speeds have been formatted correctly and in a consistent manner. The units can also be changed from kmh to mph:



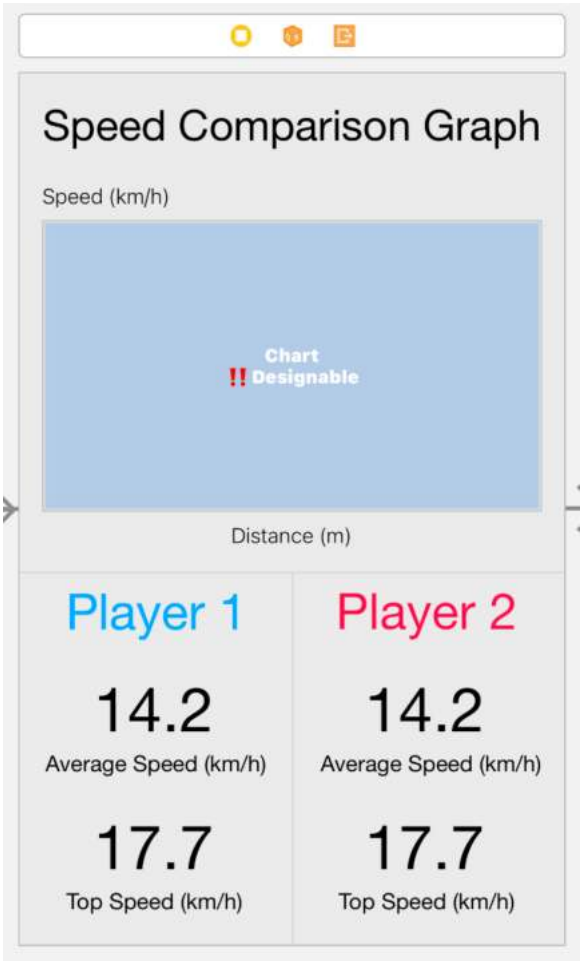


Version 7.5 - Creating the SpeedComparison Screen

Version 7.5.1 - Designing the MainStoryboard screen

Before I could start designing the screen, I had to install the graphing library. The library is called SwiftChart.

I then designed the screen:



As you can see the graph is at the top of the screen.

To fill in the space at the bottom, I have placed labels for player 1 and player 2's average speed and top speed, I will apply the same techniques from the previous screen to format these speeds.

I then connected up the components to my swift file:

```
@IBOutlet weak var chart: Chart!

@IBOutlet weak var player1AverageSpeedLabel: UILabel!
@IBOutlet weak var player1TopSpeedLabel: UILabel!
@IBOutlet weak var player2AverageSpeedLabel: UILabel!
@IBOutlet weak var player2TopSpeedLabel: UILabel!

@IBOutlet weak var player1DisplayLabel: UILabel!
@IBOutlet weak var player2DisplayLabel: UILabel!
```

Version 7.5.2 - Displaying the average speeds and top speeds

I used the same code as the code I used in the previous screen to display the average speeds and top speeds in a formatted manner and to set the text of the blue and red player labels depending on which player the user is:

```
player1AverageSpeedLabel.text =  String((RaceData.sharedInstance.averageSpeedArray [0]*3.6).roundTo(places: 2))
player1TopSpeedLabel.text =  String((RaceData.sharedInstance.topSpeedArray [0]*3.6).roundTo(places: 2))
player2AverageSpeedLabel.text =  String((RaceData.sharedInstance.averageSpeedArray [1]*3.6).roundTo(places: 2))
player2TopSpeedLabel.text =  String((RaceData.sharedInstance.topSpeedArray [1]*3.6).roundTo(places: 2))

if(RaceData.sharedInstance.playerInt == 1)
{
    player1DisplayLabel.text = "You"
    player2DisplayLabel.text = "Opponent"
}
else
{
    player2DisplayLabel.text = "You"
    player1DisplayLabel.text = "Opponent"
}
```

Version 7.5.3 - Creating the graph

I essentially have to follow my pseudocode to program the graph.

Firstly, I created the following two global arrays which will be used later to label the axis.

```
var xLabelsArray: [Double] = []
var yLabelsArray: [Double] = [0, 10, 20, 30]
```

The way the chart works is that you can add multiple series to a graph (i.e. multiple lines). The data for a series is inputted as an array of (x: Double, y: Double) tuples.

Therefore, in my RaceData class I created the following function which takes two 2d arrays and arranges them to the required array of tuple format.

```
//Takes the two arrays of a specific player and formats each element into an array...
//... of tuples so it can be used to add the series to a graph.
func arrangeToData (player: Int, xArray: [[Double]], yArray: [[Double]]) -> [(x: Double, y: Double)]
{
    var data: [(x: Double, y: Double)] = []     //dummy array to be returned
    for i in 0...xArray[player-1].count-1   //loops to the end of the specified xArray
    {
        //appends to the data array the element at the current index in both the xArray and yArray
        data.append((x: xArray[player-1][i], y: yArray[player-1][i]*3.6))
    }
    return data
}
```

This can then be utilised in the SpeedComparison screen to make the graph, because to get the data I can do the following:

```
let data1 = RaceData.sharedInstance.arrangeToData(player:  1, xArray:  RaceData.sharedInstance.distanceDataArray, yArray:
    RaceData.sharedInstance.speedDataArray)
```

The y-axis of the graph/data will be 'speed'

Looking at player 1's part of the 2d array

The x-axis of the graph/data will be 'distance'

I can then do the same thing two create a data holder for player2's series:

```
let data2 = RaceData.sharedInstance.arrangeToData(player:  2, xArray:  RaceData.sharedInstance.distanceDataArray, yArray:
    RaceData.sharedInstance.speedDataArray)
```

Then here is the code which sets up the chart. It is very straightforward and nothing has differed from my pseudocode design:

```
//*Formatting the data*
let data1 = RaceData.sharedInstance.arrangeToData(player:  1, xArray:  RaceData.sharedInstance.distanceDataArray, yArray:
    RaceData.sharedInstance.speedDataArray)
let data2 = RaceData.sharedInstance.arrangeToData(player:  2, xArray:  RaceData.sharedInstance.distanceDataArray, yArray:
    RaceData.sharedInstance.speedDataArray)

fillAxis()  //fills the xLabelsArray.

chart.xLabels = xLabelsArray    //the labels in the xLabelsArray are added to the chart
//if the user went faster than 30kmh (i.e. a fast sprint) then add 40kmh to the yLabels array
if (RaceData.sharedInstance.topSpeedArray[0]*3.6 > 30.0 || RaceData.sharedInstance.topSpeedArray[1]*3.6 > 30.0)
{
    yLabelsArray.append(40)
}
chart.yLabels = yLabelsArray    //the labels in the yLabelsArray are added to the chart

chart.xLabelsFormatter = { String(Int(round($1))) + "m" }    //adds a 'm' character to the last label on the x-axis
chart.yLabelsFormatter = { String(Int(round($1))) + "km/h" }    //adds a 'km/h' character to the last label on the y-axis

//Adds a series to the chart containing player 1's data
let series1 = ChartSeries(data: data1)
series1.color = ChartColors.blueColor()
series1.area = true
chart.add(series1)

//Adds a series to the chart containing player 2's data
let series2 = ChartSeries(data: data2)
series2.color = ChartColors.redColor()
series2.area = true
chart.add(series2)
```

The fillAxis function divides the raceDistance by 5, to get 5 equally spaced distances which will be used to label the x-axis:

```
//fills the xLabelsArray by adding 5 equally spaced distances
func fillAxis ()
{
    let interval = RaceData.sharedInstance.raceDistance/5
    var x = 0
    while (x <= RaceData.sharedInstance.raceDistance)
    {
        xLabelsArray.append(Double(x))
        x+=interval
    }
 }
```

Before I can test out this screen, I first need to add the transition from the RaceScreen to the StatsHome screen. This is because at the moment I am testing the stats using these ‘mock’ stats. The 2d arrays are very small so the graph wouldn’t look anywhere near to a normal graph. Therefore, if I can use real data from a real race, it will be a much better and more appropriate test.

```
override func viewDidLoad()
{
    super.viewDidLoad()

    RaceData.sharedInstance.speedDataArray = [[0.1, 0.4, 3, 5, 7], [0.04, 2.3, 5.7]]
    RaceData.sharedInstance.timeDataArray = [[1, 2, 3], [1, 2, 3]]
    RaceData.sharedInstance.distanceDataArray = [[1, 2, 3, 4, 20], [1, 2, 3]]
    RaceData.sharedInstance.finalTimeArray = ["10.0", "11.5"]
    RaceData.sharedInstance.averageSpeedArray = [3, 4]
    RaceData.sharedInstance.topSpeedArray = [8, 5]
    RaceData.sharedInstance.raceDistance = 400
    RaceData.sharedInstance.playerInt = 1
    RaceData.sharedInstance.playerString = "Player1"

    print("speedDataArray: \(RaceData.sharedInstance.speedDataArray)")
    print("distanceDataArray: \(RaceData.sharedInstance.distanceDataArray)")
    print("timeDataArray: \(RaceData.sharedInstance.timeDataArray)")
    print("finalTimeArray: \(RaceData.sharedInstance.finalTimeArray)")
    print("averageSpeedArray: \(RaceData.sharedInstance.averageSpeedArray)")
    print("topSpeedArray: \(RaceData.sharedInstance.topSpeedArray)")
    print("raceDistance: \(RaceData.sharedInstance.raceDistance)")
    print("player: \(RaceData.sharedInstance.playerString)")

    distanceDisplayView.layer.cornerRadius = 5
    //gets the distance from the RaceData class to display the race distance at the top of the screen
    distanceDisplayLabel.text = String(RaceData.sharedInstance.raceDistance) + "m"
}
```

Version 7.6 - Transitioning from the RaceScreen to the StatsHome screen

Looking back at the RaceScreen code, once the user has read the other user's final statistics, it is definitely time to switch screens.

```
func readFinalStatistics ()
{
    //references the opponents's document
    let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(opponentPlayerString)

    //reads the opponents's document's fields and stores the results in the corresponding arrays
    documentRef.getDocument { (document, error) in
        if let document = document, document.exists {
            let dataDescription = document.data()
            self.finalTimeArray[self.opponentPlayerInt-1] = dataDescription!["finalTime"]! as! String
            self.averageSpeedArray[self.opponentPlayerInt-1] = dataDescription!["averageSpeed"]! as! Double
            self.topSpeedArray[self.opponentPlayerInt-1] = dataDescription!["topSpeed"]! as! Double
            self.speedDataArray[self.opponentPlayerInt-1] = dataDescription!["speedDataArray"] as! [Double]
            self.distanceDataArray[self.opponentPlayerInt-1] = dataDescription!["distanceDataArray"] as! [Double]
            self.timeDataArray[self.opponentPlayerInt-1] = dataDescription!["timeDataArray"] as! [Double]
            print("SUCESS READING FINAL STATISTICS")
            //*change screen here*
        } else {
            print("Document does not exist")
        }
    }
}
```

Therefore, in the above functions, I replaced my comment with a call to the following changeStoryboard function:

```
// - MARK: Change Storyboard
func changeStoryboard()
{
    let storyboard = UIStoryboard(name: "Main", bundle: nil)

    if let destinationViewController = storyboard.instantiateViewController(withIdentifier: "statsHomeScreenController") as? statsHomeScreenController
    {
        RaceData.sharedInstance.speedDataArray      = speedDataArray
        RaceData.sharedInstance.distanceDataArray   = distanceDataArray
        RaceData.sharedInstance.timeDataArray       = timeDataArray

        RaceData.sharedInstance.finalTimeArray      = finalTimeArray
        RaceData.sharedInstance.averageSpeedArray   = averageSpeedArray
        RaceData.sharedInstance.topSpeedArray       = topSpeedArray

        RaceData.sharedInstance.raceDistance        = Int(raceDistance)
        RaceData.sharedInstance.playerString        = mainPlayerString
        RaceData.sharedInstance.playerInt           = mainPlayerInt

        playVibrate()
        self.present(destinationViewController, animated: true, completion: nil)
    }
}
```

It essentially fills out the RaceData class, setting all the fields to the corresponding global variables, which have collected all the data from both players over the race.

```
//Race Data Class variables
// ******************************************
var speedDataArray: [[Double]] = [[], []]
var distanceDataArray: [[Double]] = [[], []]
var timeDataArray: [[Double]] = [[], []]

var finalTimeArray: [String] = ["", ""]
var averageSpeedArray: [Double] = [0.0, 0.0]
var topSpeedArray: [Double] = [0.0, 0.0]
// ******************************************
```

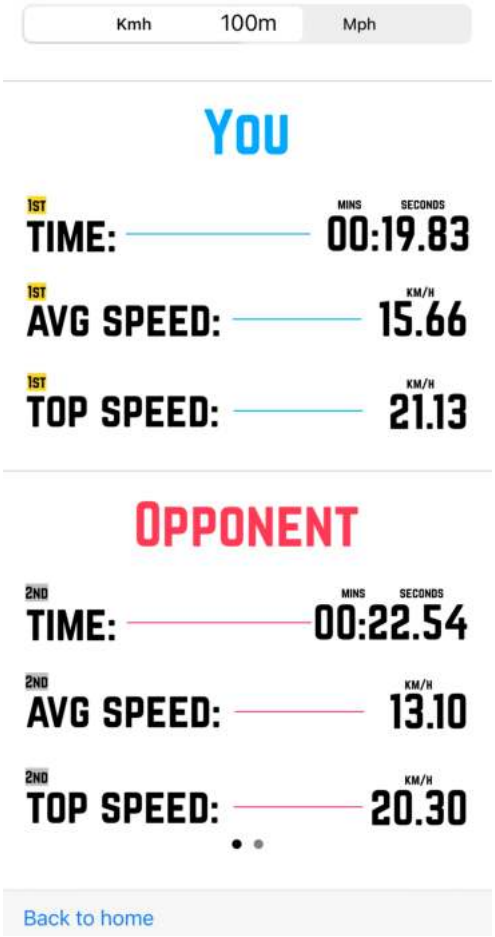
Examples of the global variables created in the RaceScreen.

Note on final screen - user's graph screen

Due to time restraints, I have chosen to not include this screen in my final solution. It is not essential to the solution and it would just look very similar to the speed comparison graph but with only one series.

Version 7.7 - Testing the stats screens

To test the stats screens, I installed the app onto my brothers phone aswell as my own. We then raced each other outside. Here is the stats screen I was shown (I was player 1):



As you can see, in a real life situation, the stats screen works great!

Stakeholder feedback

I sent these two screenshots to my stakeholders to see what they thought:

Aron's response:

"I think the two screens you have designed are great. They are clear, colourful and simple for any user to understand. I particularly love how you display which stat won for each player either highlighted gold or silver - it is so easy to see from a glance

who won each stat. Also, the way the times and speeds are consistently formatted also makes for an easier to read."

David's response:

"First of all, it is amazing how easy it is to draw comparisons between two players from these screens - it further adds to the competitive nature of the app. I also like how you have again kept to the simple colour theme of 'blue vs red'.

Reflecting on the graph, in my opinion, it is easy to interpret and the labels on the x and y axis and the axis labels make it easy for anyone to understand. My only point of concern is that the graph isn't especially smooth - it is quite jagged with sharp jumps."

Response to stakeholder feedback

Overall, the stakeholder's feedback is very positive. However, in response to David's comment about the graph not being smooth, the reason it is very sharp is due to the GPS and not due to my code. At the moment, I am reading the user's speed, distance (and the current time) every 0.5 seconds and appending them to a list. The speed only changes however when the GPS detects that the user has changed location. This happens less frequently due to the nature of GPS not being accurate to the nearest metre. Therefore, the GPS function which reads the user's speed may only be triggered every 1 second. So in reflection, there is not much I can do to improve the number of data points being collected. However, over the course of a longer race, as opposed to the shown 100m race, the graph will look much smoother because proportionally there will be more data points being collected.

Version 7 - Review

What has been done

First of all, I created the RaceData class, which acts like a data structure to store all the running stats collected so that they can be stored in one variable and accessed easily.

I then created the StatsHome screen. It contains a segmented control which allows the user to convert the units from kmh to mph. It contains a placeholder view for the PageViewController.

I configured the PageViewController so that it is set up for the two screens I will be showing. I then created each screen:

The first screen shows a comparison of all the stats collected. The time statistic is formatted in a consistent manner of ‘minutes:seconds.miliseconds’ and the two speed stats (average and top speed) are formatted in a constant manner of having 2 decimal places. A small text is also shown above each stat illustrating which player won the stat. Through efficient algorithms, which compare that statistic of both players, the winner’s text is formatted to read ‘1st’ and is highlighted gold, whilst the loser’s text is formatted to read ‘2nd’ and is highlighted silver.

The next screen shows a graph of each players speed plotted against the distance run. Each player’s line (series) is plotted in their respective colour theme - blue for player 1, red for player 2. Below the graph, each player’s average speeds and top speeds are shown in the same consistent formatting.

What has changed from the original design

I came to a decision to not included the final screen due to time restrictions.

What has been tested

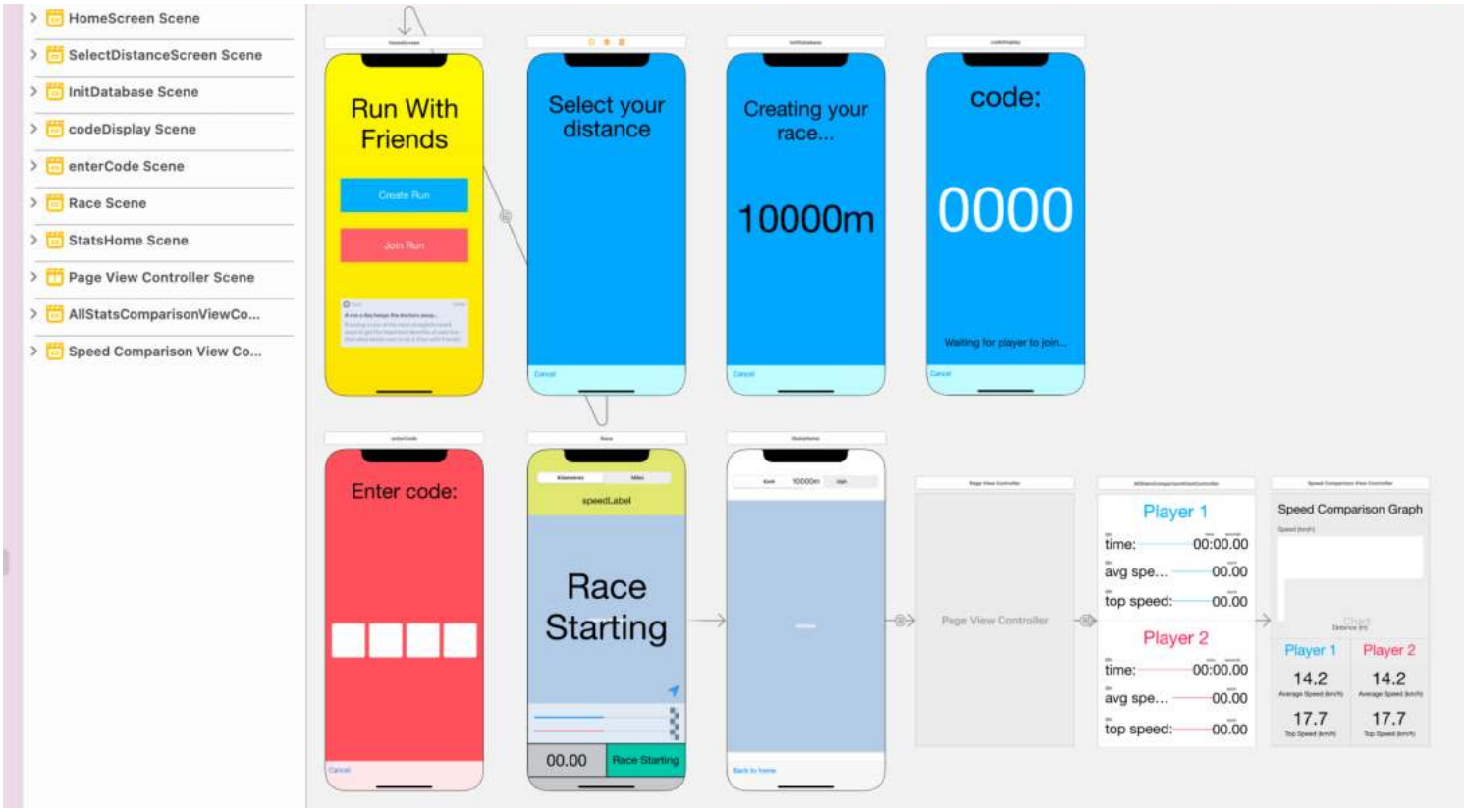
I have tested that the page view controller is fully functional, and when the end of the pages are reached, the user cannot swipe any further.

On the AllStatsComparison screen, I checked that the speeds and times where being formatted correctly and also that the correct position indicators above each stats where being displayed correctly (i.e. if player 1’s final time is faster, then a 1st should be written above that stat in gold).

On the SpeedComparison screen, I put the graph to the test by running a race against my brother. I installed the app onto my brothers phone aswell as my own. We then raced each other outside. I will test this screen even more when I carry out further tests as part of my evaluation section.

Final Code

Main Storyboard



HomeScreenViewController.Swift

```swift
//
//  HomeScreen.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 03/02/2021.
//

import Foundation
import UIKit

class HomeScreenViewController: UIViewController
{
    @IBOutlet var createRunButton: UIButton!
    @IBOutlet var joinRunButton: UIButton!

    override func viewDidLoad()
    {
        super.viewDidLoad()

        createRunButton.layer.cornerRadius = 6
        joinRunButton.layer.cornerRadius = 6
    }

    @IBAction func createRunButtonPressed(_ sender: Any)
    {
        print("Create Run Button Pressed")
        createRunScreen()
    }

    @IBAction func joinRunButtonPressed(_ sender: Any)
    {
        print("Join Run Button Pressed")
        joinRunScreen()
    }

    func createRunScreen()
    {
        let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
        print("change storyboard")
        if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier: "SelectDistanceViewController") as? SelectDistanceViewController
        {
            self.present(destinationViewController, animated: false, completion: nil)
        }
    }

    func joinRunScreen()
    {
        let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
        print("change storyboard")
        if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier: "EnterCodeViewController") as? EnterCodeViewController
        {
            self.present(destinationViewController, animated: false, completion: nil)
        }
    }
}
```

SelectDistanceViewController.Swift

```swift
//
//  selectselectedDistanceViewController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 05/02/2021.
//

import Foundation
import UIKit
import SDCAlertView

class SelectDistanceViewController: UIViewController
{
    var selectedDistance = 0

    override func viewDidLoad()
    {
        super.viewDidLoad()
    }

    override func viewDidAppear(_ animated: Bool)
    {
        let alert = AlertController(title: "Please select your distance", message: "", preferredStyle: .actionSheet)

        alert.addAction(AlertAction(title: "Cancel", style: .destructive) { (action) in
            self.dismiss(animated: true, completion: nil)
            })
        alert.addAction(AlertAction(title: "100m", style: .normal) { (action) in
            self.selectedDistance = 100
            self.changeStoryboard()
        })
        alert.addAction(AlertAction(title: "200m", style: .normal) { (action) in
            self.selectedDistance = 200
            self.changeStoryboard()
        })
        alert.addAction(AlertAction(title: "400m", style: .normal) { (action) in
            self.selectedDistance = 400
            self.changeStoryboard()
        })
        alert.addAction(AlertAction(title: "1000m", style: .normal) { (action) in
            self.selectedDistance = 1000
            self.changeStoryboard()
        })
        alert.addAction(AlertAction(title: "2000m", style: .normal) { (action) in
            self.selectedDistance = 2000
            self.changeStoryboard()
        })
        alert.addAction(AlertAction(title: "5000m", style: .normal) { (action) in
            self.selectedDistance = 5000
            self.changeStoryboard()
        })
        alert.addAction(AlertAction(title: "10000m", style: .normal) { (action) in
            self.selectedDistance = 10000
            self.changeStoryboard()
        })

        alert.present()
    }

    //transitions from the SelectDistance screen to the InitDatabase screen
    func changeStoryboard()
    {
        let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
        print("change storyboard")
        if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier: "InitDatabaseViewController") as? InitDatabaseViewController
        {
            destinationViewController.distanceObject = selectedDistance //passes the selectedDistance value to the next screen
            self.present(destinationViewController, animated: false, completion: nil)
        }
    }
}

```

InitDatabaseViewController.Swift

```swift
//
//  InitDatabaseScreenController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 06/03/2021.
//

import Foundation
import UIKit
import Firebase

class InitDatabaseViewController: UIViewController
{
    // MARK: Outlets
    @IBOutlet weak var distanceLabel: UILabel!

    // MARK: Variables
    var distanceObject: Int?
    var code = ""

    var ref: DocumentReference? = nil
    var db: Firestore!

    override func viewDidLoad()
    {
        super.viewDidLoad()
        //randomise the code before setting up the database

        db = Firestore.firestore() //required for setting connecting to database
        randomiseCode() //randomises the code then set's up the database

        distanceLabel.text = String(distanceObject!) + "m"
    }

    func randomiseCode ()
    {
        let int1 = Int.random(in: 0 ... 9)
        let int2 = Int.random(in: 0 ... 9)
        let int3 = Int.random(in: 0 ... 9)
        let int4 = Int.random(in: 0 ... 9)
        code = String(int1) + String(int2) + String(int3) + String(int4)

        setUpDatabase()
    }

    func setUpDatabase()
    {
        print("setting up db")

        //Queries the database to find a document with the same randomly generated code.
        db.collection("Races").whereField("code", isEqualTo: code)
            .getDocuments() { (querySnapshot, err) in
                if let err = err {
                    print("Error getting documents: \(err)")
                }
                else
                {
                    print(querySnapshot?.documents)                                                    ⚠ Expression implicitly coerc
                    if ((querySnapshot?.documents.count)! >= 1)    //if there is a duplicate
                    {
                        self.randomiseCode()
                    }
                    else
                    {
                        //creates a new document (of auto-id) in the Races collection with the distance field set to the distance selected by the user
                        self.ref = self.db.collection("Races").addDocument(data: ["code": self.code, "distance": self.distanceObject!, "player2Joined": false])

                        //creates a new collection in that document called "Players"...
                        //...and inside that collection a new document is created with the following fields.
                        self.db.collection("Races").document(self.ref!.documentID).collection("Players").document("Player1").setData([
                            "playerDistance": 0,
                            "isReady":false,
                            "finalTime":"",
                            "averageSpeed":0.0,
                            "topSpeed":0.0,
                            "speedDataArray": [0.0],
                            "distanceDataArray": [0.0],
                            "timeDataArray": [0.0],
                            "hasFinished":false
                        ])

                        self.changeStoryboard()
                    }
                }
            }
    }

    func changeStoryboard()
    {
        print("change storyboard")
        let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
        if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier: "CodeDisplayViewController") as? CodeDisplayViewController
        {
            destinationViewController.codeObject = code
            destinationViewController.documentObject = ref!.documentID
            //ref!.documentID holds the ID of the document of the specific race being established.
            self.present(destinationViewController, animated: true, completion: nil)
        }
    }
}

```

CodeDisplayViewController.Swift

```swift
//
//  codeDisplayViewController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 09/03/2021.
//

import Foundation
import UIKit
import Firebase
import CDAlertView

class CodeDisplayViewController: UIViewController
{
    // MARK: Outlets
    @IBOutlet weak var codeLabel: UILabel!

    // MARK: Variables
    var codeObject: String?
    var documentObject: String?

    var ref: DocumentReference? = nil
    var db: Firestore!

    override func viewDidLoad()
    {
        super.viewDidLoad()

        codeLabel.text = codeObject!

        db = Firestore.firestore() //line required for database
        checkForJoin() //calls the checkForJoin function
    }

    func checkForJoin ()
    {
        //Adds a listener to the document where the code field is the same as the code/codeObject
        db.collection("Races").whereField("code", isEqualTo: codeObject!)
            .addSnapshotListener { querySnapshot, error in
                guard let snapshot = querySnapshot else {
                    print("Error fetching snapshots: \(error!)") //in case of error
                    return
                }
                snapshot.documentChanges.forEach { diff in
                    if (diff.type == .modified) //when a field in the document is 'modified' this is triggered
                                                //i.e. when the player2HasJoined field is changed to true
                    {
                        //sets a timer for 0.8 seconds which when finished calls the player2Joined function
                        Timer.scheduledTimer(timeInterval: 0.8, target: self, selector: #selector(self.player2Joined), userInfo: nil, repeats: false)
                    }
                }
        }
    }

    @objc func player2Joined ()
    {
        print("Player 2 Joined")

        //Creates the alert, of type 'success' which means it will have a green tick
        //.show() means the alert is shown
        CDAlertView(title: "Success", message: "Player 2 has joined", type: .success).show()

        //Vibrates the phone to notify the user
        let generator = UINotificationFeedbackGenerator()
        generator.notificationOccurred(.success)

        changeStoryboard()
    }

    func changeStoryboard()
    {
        print("change storyboard")
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        if let destinationViewController = storyboard.instantiateViewController(withIdentifier: "RaceViewController") as? RaceViewController
        {
            destinationViewController.raceDocumentID = documentObject!
            //User is player 1:
            destinationViewController.mainPlayerString = "Player1"
            destinationViewController.mainPlayerInt = 1
            destinationViewController.opponentPlayerString = "Player2"
            destinationViewController.opponentPlayerInt = 2
            self.present(destinationViewController, animated: true, completion: nil)
        }
    }

    @IBAction func cancelButtonPressed(_ sender: Any)
    {
        //Deletes the document so that the other player can no longer join an empty race.
        db.collection("Races").document(documentObject!).delete() { err in
            if let err = err {
                print("Error removing document: \(err)")
            } else {
                print("Document successfully removed!")
            }
        }

        returnToHomeScreen()
    }

    func returnToHomeScreen()
    {
        let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
        if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier: "HomeScreenViewController") as? HomeScreenViewController {
            self.present(destinationViewController, animated: true, completion: nil)
        }
    }
}
```

EnterCodeViewController.Swift

```swift
//
//  EnterCodeViewController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 11/03/2021.
//

import Foundation
import UIKit
import CDAlertView
import Firebase

class EnterCodeViewController: UIViewController
{
    //MARK: IBOutlets
    @IBOutlet weak var tf1: UITextField!
    @IBOutlet weak var tf2: UITextField!
    @IBOutlet weak var tf3: UITextField!
    @IBOutlet weak var tf4: UITextField!

    //MARK: Variables
    var ref: DocumentReference? = nil
    var db: Firestore!
    var raceID = ""
    var codeIsCorrect = false   //Variable which will be set to whether or not the code has been found in the database


    override func viewDidLoad()
    {
        super.viewDidLoad()

        let tapGesture = UITapGestureRecognizer(target: self, action: #selector(hideKeyboard))
        view.addGestureRecognizer(tapGesture)

        //activates text field 1:
        tf1.becomeFirstResponder()
        tf1.layer.borderWidth = 2

        tf1.layer.borderColor = UIColor.darkGray.cgColor
        tf2.layer.borderColor = UIColor.darkGray.cgColor
        tf3.layer.borderColor = UIColor.darkGray.cgColor
        tf4.layer.borderColor = UIColor.darkGray.cgColor

        tf1.layer.cornerRadius = 6
        tf2.layer.cornerRadius = 6
        tf3.layer.cornerRadius = 6
        tf4.layer.cornerRadius = 6

        db = Firestore.firestore()
    }

    //function is triggered when the user types in any of the four text fields
    @IBAction func textFieldEdited(_ sender: UITextField)
    {
        print("textEditChanged has been pressed")

        //vibrates the phone very lightly
        let generator = UIImpactFeedbackGenerator(style: .light)
        generator.impactOccurred()

        //counts the number of characters/digits in that text field
        let count = sender.text?.count

        //if the number of digits is 1, then move to the next text field:
        if count == 1
        {
            switch sender {
            case tf1:   //if text field 1 now contains a digit, then text field 2 is activated
                tf2.becomeFirstResponder()
                tf1.layer.borderWidth = 0
                tf2.layer.borderWidth = 2
            case tf2:   //if text field 2 now contains a digit, then text field 3 is activated
                tf3.becomeFirstResponder()
                tf2.layer.borderWidth = 0
                tf3.layer.borderWidth = 2
            case tf3:   //if text field 3 now contains a digit, then text field 4 is activated
                tf4.becomeFirstResponder()
                tf3.layer.borderWidth = 0
                tf4.layer.borderWidth = 2
            case tf4:    //if text field 4 now contains a digit, then it is disabled and dismissed...
                         //... so that the user cannot enter another digit.
                tf4.isEnabled = false
                tf4.layer.borderWidth = 0
                checkCode()
            default:
                print("default")
            }
        }

    }

    func checkCode()
    {
        //the users inputted code is stored in a variable
        let code = tf1.text! + tf2.text! + tf3.text! + tf4.text!
        print ("code: \(code)")

        //*************************
        //finds all documents in the Races collection of the database where the code is the same as the user's inputted code
        db.collection("Races").whereField("code", isEqualTo: code).whereField("player2Joined", isEqualTo: false)
            .getDocuments() { (querySnapshot, err) in
                if let err = err {
                    print("Error getting documents: \(err)")
                }
                else
                {
                    if (querySnapshot?.documents.count == 0)    //if no documents are found
                    {
                        print("NONE FOUND")
                        self.codeIsCorrect = false
                    }
                    else
                    {
                        self.codeIsCorrect = true

                        //the following code must be executed here because it is inside the database query closure:
                        self.raceID = (querySnapshot?.documents[0].documentID)!  //stores the ID of the document where this code was found

                        //Creates a document reference to this document
                        let documentRef = self.db.collection("Races").document(self.raceID)

                        //updates the player2Joined field to true
                        documentRef.updateData([ "player2Joined": true ]) { err in
                            if let err = err {
                                print("Error updating document: \(err)")
                            } else {
                                //print("Document successfully updated")
```

```
                        } else {
                            //print("Document successfully updated")
                        }
                    }

                    //adds a new document to the 'Players' collection called 'Player2' with the necessary fields
                    documentRef.collection("Players").document("Player2").setData([
                        "playerDistance": 0,
                        "isReady":false,
                        "finalTime":"",
                        "averageSpeed":0.0,
                        "topSpeed":0.0,
                        "speedDataArray": [0.0],
                        "distanceDataArray": [0.0],
                        "timeDataArray": [0.0],
                        "hasFinished":false
                    ])
                }

                if (self.codeIsCorrect == false)
                {
                    //clears all the text fields
                    self.clearTextFields()
                    self.tf1.becomeFirstResponder()  //prompts the user to enter in the first digit
                    self.tf1.layer.borderWidth = 2
                    self.tf4.isEnabled = true    //re-enables the last text field

                    //Creates the alert, of type 'error' which means it will have a red x
                    let alert = CDAlertView(title: "Invalid", message: "Wrong code", type: .error)
                    alert.autoHideTime = 3 // This will hide alert box after 3 second
                    alert.show()

                    //Vibrates the phone in a pattern which represents an error
                    let generator = UINotificationFeedbackGenerator()
                    generator.notificationOccurred(.error)
                }
                else
                {
                    //Creates the alert, of type 'success' which means it will have a green tick
                    CDAlertView(title: "Success", message: "You have joined the race", type: .success).show()

                    //Vibrates the phone to notify the user of the success
                    let generator = UINotificationFeedbackGenerator()
                    generator.notificationOccurred(.success)

                    self.changeStoryboard()
                }
            }
        }
    }

    func changeStoryboard ()
    {
        print("Change Storyboard")

        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        if let destinationViewController = storyboard.instantiateViewController(withIdentifier: "RaceViewController") as? RaceViewController
        {
            destinationViewController.raceDocumentID = self.raceID
            //User is player 2:
            destinationViewController.mainPlayerString = "Player2"
            destinationViewController.mainPlayerInt = 2
            destinationViewController.opponentPlayerString = "Player1"
            destinationViewController.opponentPlayerInt = 1
            self.present(destinationViewController, animated: true, completion: nil)
        }
    }

    func clearTextFields()
    {
        tf1.text = ""
        tf2.text = ""
        tf3.text = ""
        tf4.text = ""

        //reset all the border widths to 0 as it is not know which text field was the last active.
        tf1.layer.borderWidth = 0
        tf2.layer.borderWidth = 0
        tf3.layer.borderWidth = 0
        tf4.layer.borderWidth = 0
    }

    @objc func hideKeyboard()
    {
        view.endEditing(true)   //hides the keyboard

        clearTextFields()
    }

    @IBAction func textFieldsPressed(_ sender: Any)
    {
        tf1.becomeFirstResponder()  //prompts the user to enter in the first digit again
        tf1.layer.borderWidth = 2
    }

    @IBAction func cancelButtonPressed(_ sender: Any)
    {
        returnToHomeScreen()
    }

    func returnToHomeScreen()
    {
        let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
        if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier: "HomeScreenViewController") as? HomeScreenViewController {
            self.present(destinationViewController, animated: true, completion: nil)
        }
    }
}
```

RaceViewController.Swift

```swift
//
//  RaceScreen.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 22/03/2021.
//

import UIKit
import Foundation
import MapKit
import Firebase
import CDAlertView
import AudioToolbox

class RaceViewController: UIViewController
{
    // - MARK: Outlets
    @IBOutlet weak var mapView: MKMapView!
    @IBOutlet weak var milesOrKilometresSegment: UISegmentedControl!
    @IBOutlet weak var speedLabel: UILabel!
    @IBOutlet weak var timerLabel: UILabel!
    @IBOutlet weak var infoLabel: UILabel!
    @IBOutlet weak var player1ProgressBar: UIProgressView!
    @IBOutlet weak var player2ProgressBar: UIProgressView!
    @IBOutlet weak var countdownLabel: UILabel!
    @IBOutlet weak var locateButton: UIButton!

    // - MARK: Variables
    let locationManager = CLLocationManager()
    var regionInMeters: Double = 300
    var oldLocation = CLLocation()

    // Passed from previous view controllers
    var raceDocumentID = ""
    var mainPlayerString = ""
    var mainPlayerInt = 0
    var opponentPlayerString = ""
    var opponentPlayerInt = 0

    //Race Data Class variables
    // ********************************************
    var speedDataArray: [[Double]] = [[], []]
    var distanceDataArray: [[Double]] = [[], []]
    var timeDataArray: [[Double]] = [[], []]

    var finalTimeArray: [String] = ["", ""]
    var averageSpeedArray: [Double] = [0.0, 0.0]
    var topSpeedArray: [Double] = [0.0, 0.0]
    // ********************************************

    var speedInMPS = 0.0
    var totalSpeed = 0.0
    var distanceCoveredInMetres = 0.0

    var raceDistance = 0.0
    var distance = 0.0
    var otherPlayerHasFinished = false

    var p1Distance = 0.0
    var p2Distance = 0.0

    var calibrating = false

    var countdownTimer = Timer()
    var countdownTimerCount = 0.0
    var raceTimer = Timer()
    var raceTimerCount = 0.0
    var waitForOtherPlayerTimer = Timer()

    var ref: DocumentReference? = nil
    var db: Firestore!

    // - MARK: View Life Cycles
    override func viewDidLoad()
    {
        super.viewDidLoad()

        db = Firestore.firestore()
        getRaceDistance()

        checkLocationServices()

        locateButton.layer.cornerRadius = 5
        player1ProgressBar.progress = 0.0025
        player2ProgressBar.progress = 0.0025

        //clear countdownLabel text
        countdownLabel.text = ""
    }

    override func viewDidAppear(_ animated: Bool)
    {
        showReadyButton()
    }

    // - MARK: Readying Up
    func showReadyButton()
    {
        //Shows alert with a button which reads 'ready'
        let alert = CDAlertView(title: "Race starting soon...", message: "Are you ready?!", type: .notification)
        let readyAction = CDAlertViewAction(title: "I am ready!")
        alert.add(action: readyAction)
        alert.canHideWhenTapBack = false
        alert.show() { (alert) in
            self.userIsReady()
        }
    }

    func userIsReady()
    {
        //updates the user's 'isReady' field to true
        var documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(self.mainPlayerString)
        documentRef.updateData([ "isReady": true ]) { err in
            if let err = err {
                print("Error updating document: \(err)")
            } else {
                print("Document successfully updated")
            }
        }

        //Adds a listener to the opponents document
        documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(self.opponentPlayerString)
        documentRef.addSnapshotListener { documentSnapshot, error in
                guard let document = documentSnapshot else {
                    print("Error fetching document: \(error!)")
                    return
                }
                let source = document.metadata.hasPendingWrites ? "Local" : "Server"
```

```
                print("\(source) data: \(document.data() ?? [:])")
            //if the field 'isReady' is set to true, both players are ready
            if (document.data()?["isReady"] as! Int == 1)
            {
                Timer.scheduledTimer(timeInterval: 0.5, target: self, selector: #selector(self.startCountdown), userInfo: nil, repeats: false)
            }
        }
    }

    // - MARK: Countdown
    @objc func startCountdown()
    {
        countdownLabel.font = countdownLabel.font.withSize(140)
        infoLabel.text = "Race Starting"
        countdownTimerCount = 0
        //creates a timer which calls the function updateCountdownTime() every 1 second:
        countdownTimer = Timer.scheduledTimer(timeInterval: 1, target: self, selector: (#selector(RaceViewController.updateCountdownTime)), userInfo: nil, repeats: true)
    }

    @objc func updateCountdownTime()
    {
        countdownTimerCount += 1
        if (countdownTimerCount == 3)
        {
            countdownLabel.font = countdownLabel.font.withSize(230)
            countdownLabel.text = "3"
            playVibrate()
        }
        else if (countdownTimerCount == 4)
        {
            countdownLabel.text = "2"
            playVibrate()
        }
        else if (countdownTimerCount == 5)
        {
            countdownLabel.text = "1"
            playVibrate()
        }
        else if (countdownTimerCount == 6)
        {
            infoLabel.text = "Race!"
            countdownLabel.font = countdownLabel.font.withSize(200)
            countdownLabel.text = "GO!"
            for _ in 1...3
            {
                self.playVibrate()
                print("vibrate 2")
            }
            startRaceTimer()
        }
        else if (countdownTimerCount == 7)
        {
            countdownLabel.text = ""
            countdownTimer.invalidate() //stops the timer
        }
    }

    // - MARK: Race Timer
    func startRaceTimer()
    {
        //add a 0 to each stat array (because at time = 0 the user's speed and distance are both 0)
        speedDataArray[mainPlayerInt-1].append(0)
        distanceDataArray[mainPlayerInt-1].append(0)
        timeDataArray[mainPlayerInt-1].append(0)
        raceTimerCount = 0
        //creates a timer which calls the function updateCountdownTime() every 0.01 second:
        raceTimer = Timer.scheduledTimer(timeInterval: 0.01, target: self, selector: (#selector(RaceViewController.updateRaceTime)), userInfo: nil, repeats: true)
    }

    @objc func updateRaceTime()
    {
        raceTimerCount = raceTimerCount + 0.01   //increments the raceTimerCount by 0.01
        raceTimerCount = round(raceTimerCount * 100) / 100   //rounds it to 2.dp (because floats can sometimes become recurring)

        //updates the timer label
        var timerText = String(format: "%.2f", raceTimerCount)
        if (raceTimerCount < 10)
        {
            timerText.insert("0", at: timerText.startIndex)  //inserts a '0' at the beginning of the string
        }
        timerLabel.text = timerText

        totalSpeed += speedInMPS
        distanceCoveredInMetres = (Double (totalSpeed) * 1000).rounded() / 100000   //rounds the distance to 2dp

        //Every 0.5 seconds update the stats arrays
        if (remainderOfDoubles(x: raceTimerCount, y: 0.5) == 0)
        {
            update2DArrays()
        }

        //Player 1's
        if (mainPlayerString == "Player1")
        {
            p1Distance = distanceCoveredInMetres
            player1ProgressBar.progress = Float(p1Distance/raceDistance)    //progress bar set to proportion of race distance covered
            //Every 0.2 seconds:
            if (remainderOfDoubles(x: raceTimerCount, y: 0.2) == 0)
            {
                //read player2's distance and update the progress bar
                p2Distance = readDistance()
                player2ProgressBar.progress = Float(p2Distance/raceDistance)

                writeDistance()    //write player 1's distance to the database
            }
            // PLAYER 1 HAS FINISHED
            if (p1Distance >= raceDistance)
            {
                raceFinished()
            }
        }
        //Player 2's
        else if (mainPlayerString == "Player2")
        {
            p2Distance = distanceCoveredInMetres
            player2ProgressBar.progress = Float(p2Distance/raceDistance)    //progress bar set to proportion of race distance covered
            //Every 0.2 seconds:
            if (remainderOfDoubles(x: raceTimerCount, y: 0.2) == 0)
            {
                //read player1's distance and update the progress bar
                p1Distance = readDistance()
                player1ProgressBar.progress = Float(p1Distance/raceDistance)

                writeDistance()    //write player 2's distance to the database
            }
            // PLAYER 2 HAS FINISHED
```

```
            if (p2Distance >= raceDistance)
            {
                raceFinished()
            }
        }
    }

    func update2DArrays()
    {
        speedDataArray[mainPlayerInt-1].append(speedInMPS)
        distanceDataArray[mainPlayerInt-1].append(distanceCoveredInMetres)
        timeDataArray[mainPlayerInt-1].append(raceTimerCount)
    }

    // - MARK: Race Finished
    func raceFinished()
    {
        finalTimeArray[mainPlayerInt-1] = String(raceTimerCount)    //adds the user's time to the user's index of the finalTimeArray
        print ("Time: " + finalTimeArray[mainPlayerInt-1])
        playVibrate()
        raceTimer.invalidate()     //stops the raceTimer
        calulateAverageSpeed ()
        calculateTopSpeed ()
        writeFinalStats()     //write the user's final stats to the database
        if (checkPlayerHasFinished() == false)
        {
            waitTillPlayerHasFinished()
            startWaitForOtherPlayerTimer()
        }
        else
        {
            readFinalStatistics()
        }
    }

    func startWaitForOtherPlayerTimer()
    {
        //creates a timer which calls the function updateTimeOtherPlayer() every 0.2 seconds:
        waitForOtherPlayerTimer = Timer.scheduledTimer(timeInterval: 0.2, target: self, selector: (#selector(RaceViewController.updateTimeOtherPlayer)), userInfo: nil, repeats: true)
    }

    @objc func updateTimeOtherPlayer ()
    {
        //if user is player 1, update player 2's progress bar
        if (mainPlayerString == "Player1")
        {
            p2Distance = readDistance()
            player2ProgressBar.progress = Float(p2Distance/raceDistance)
        }
        //if user is player 2, update player 1's progress bar
        else
        {
            p1Distance = readDistance()
            player1ProgressBar.progress = Float(p1Distance/raceDistance)
        }
    }

    // - MARK: Database

    //Sets global variable 'raceDistance' to the race distance from the document in database
    func getRaceDistance ()
    {
        let documentRef = self.db.collection("Races").document(self.raceDocumentID)
        var d = 0
        documentRef.getDocument { (document, error) in
            if let document = document, document.exists {
                let dataDescription = document.data()
                d = dataDescription!["distance"]! as! Int
                self.raceDistance = Double(d)
            } else {
                print("Document does not exist")
            }
        }
    }

    //writes the distance the user has run at that point in time:
    func writeDistance()
    {
        let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(mainPlayerString)
        documentRef.updateData([
            "playerDistance": distanceCoveredInMetres
        ]) { err in
            if let err = err {
                print("Error updating document: \(err)")
            } else {
                //print("Document successfully updated")
            }
        }
    }

    //returns the distance the opponent has run at that point in time:
    func readDistance () -> Double
    {
        let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(opponentPlayerString)
        documentRef.getDocument { (document, error) in
            if let document = document, document.exists {
                let dataDescription = document.data()
                self.distance = dataDescription!["playerDistance"]! as! Double
            } else {
                print("Document does not exist")
            }
        }
        return distance
    }

    func writeFinalStats()
    {
        //references the mainPlayerString's document
        let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(mainPlayerString)

        //updates the fields to the user's data in the arrays using the mainPlayerInt variable
        documentRef.updateData(["finalTime": finalTimeArray[mainPlayerInt-1],
                                "averageSpeed":averageSpeedArray[mainPlayerInt-1],
                                "topSpeed":topSpeedArray[mainPlayerInt-1],
                                "speedDataArray": speedDataArray[mainPlayerInt-1],
                                "distanceDataArray": distanceDataArray[mainPlayerInt-1],
                                "timeDataArray": timeDataArray[mainPlayerInt-1],
                                "hasFinished":true
            ])
        { err in
            if let err = err {
                print("Error updating document: \(err)")
            } else {
                print("Document successfully updated")
            }
        }
    }
```

```
func checkPlayerHasFinished () -> Bool
{
    let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(opponentPlayerString)

    documentRef.getDocument { (document, error) in
        if let document = document, document.exists {
            let dataDescription = document.data()
            self.otherPlayerHasFinished = dataDescription!["hasFinished"]! as! Bool
        } else {
            print("Document does not exist")
        }
    }
    return otherPlayerHasFinished
}

func waitTillPlayerHasFinished ()
{
    //references the opponents's document
    let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(opponentPlayerString)

    //Adds a listener to this document
    documentRef.addSnapshotListener { documentSnapshot, error in
      guard let document = documentSnapshot else {
        print("Error fetching document: \(error!)") //error
        return
      }
      guard let data = document.data() else {
        print("Document data was empty.")  //document is empty
        return
      }
        if (data["hasFinished"] as! Bool == true)   //if the field 'hasFinished' is set to true, both players have finished
        {
            self.readFinalStatistics()  //read the opponents final statistics
            self.waitForOtherPlayerTimer.invalidate()  //stops the waitForOtherPlayerTimer
        }
    }
}

func readFinalStatistics ()
{
    //references the opponents's document
    let documentRef = self.db.collection("Races").document(self.raceDocumentID).collection("Players").document(opponentPlayerString)

    //reads the opponents's document's fields and stores the results in the corresponding arrays
    documentRef.getDocument { (document, error) in
        if let document = document, document.exists {
            let dataDescription = document.data()
            self.finalTimeArray[self.opponentPlayerInt-1] = dataDescription!["finalTime"]! as! String
            self.averageSpeedArray[self.opponentPlayerInt-1] = dataDescription!["averageSpeed"]! as! Double
            self.topSpeedArray[self.opponentPlayerInt-1] = dataDescription!["topSpeed"]! as! Double
            self.speedDataArray[self.opponentPlayerInt-1] = dataDescription!["speedDataArray"] as! [Double]
            self.distanceDataArray[self.opponentPlayerInt-1] = dataDescription!["distanceDataArray"] as! [Double]
            self.timeDataArray[self.opponentPlayerInt-1] = dataDescription!["timeDataArray"] as! [Double]
            print("SUCESS READING FINAL STATISTICS")
            self.changeStoryboard()
        } else {
            print("Document does not exist")
        }
    }
}

// - MARK: Change Storyboard
func changeStoryboard()
{
    let storyboard = UIStoryboard(name: "Main", bundle: nil)

    if let destinationViewController = storyboard.instantiateViewController(withIdentifier: "statsHomeScreenController") as? statsHomeScreenController
    {
        RaceData.sharedInstance.speedDataArray      = speedDataArray
        RaceData.sharedInstance.distanceDataArray   = distanceDataArray
        RaceData.sharedInstance.timeDataArray       = timeDataArray

        RaceData.sharedInstance.finalTimeArray      = finalTimeArray
        RaceData.sharedInstance.averageSpeedArray   = averageSpeedArray
        RaceData.sharedInstance.topSpeedArray       = topSpeedArray

        RaceData.sharedInstance.raceDistance        = Int(raceDistance)
        RaceData.sharedInstance.playerString        = mainPlayerString
        RaceData.sharedInstance.playerInt           = mainPlayerInt

        playVibrate()
        self.present(destinationViewController, animated: true, completion: nil)
    }
}

// - MARK: Calculations
func calulateAverageSpeed ()
{
    let count = speedDataArray[mainPlayerInt-1].count + 50  //times by 50 because the array is updated every 0.2 seconds

    averageSpeedArray[mainPlayerInt-1] = (totalSpeed / Double(count)).roundTo(places: 2)
    print ("Average Speed: " + String (averageSpeedArray[mainPlayerInt-1]))
}

func calculateTopSpeed ()
{
    topSpeedArray[mainPlayerInt-1] = Double(speedDataArray[mainPlayerInt-1].max()!).roundTo(places: 2)
    print ("Top Speed: " + String (topSpeedArray[mainPlayerInt-1]))
}

// - MARK: Other
func playVibrate()
{
    AudioServicesPlayAlertSound(SystemSoundID(kSystemSoundID_Vibrate))
}

func remainderOfDoubles (x: Double, y: Double) -> Double
{
    let a = Double(x/y).roundTo(places: 2)
    let b = Double(x/y).roundTo(places: 2).rounded(.down)
    return (a - b)
}

// - MARK: Map View
func checkLocationServices()
{
    //checks if user has enabled their location.
    if CLLocationManager.locationServicesEnabled()
    {
        setupLocationManager()
        checkLocationAuthorization()
    }
    else
    {
        print ("location services not turned on")
    }
}

func setupLocationManager()
```

```
510
511     func setupLocationManager()
512     {
513         locationManager.delegate = self
514         locationManager.desiredAccuracy = kCLLocationAccuracyBest   //best accuracy needed.
515         locationManager.requestWhenInUseAuthorization()    //request to access location when in use.
516         locationManager.activityType = .fitness     //most precise activty type.
517         //at this point ask user's both users to press ready
518     }
519
520     func checkLocationAuthorization ()
521     {
522         switch CLLocationManager.authorizationStatus()
523         {
524         case .authorizedWhenInUse:
525             locationManager.requestLocation()   //gets the user's location
526             mapView.showsUserLocation = true
527             centreViewOnUserLocation()
528             locationManager.startUpdatingLocation()
529         case .denied:
530             break
531         case .notDetermined:
532             locationManager.requestWhenInUseAuthorization()
533         case .restricted:
534             break
535         case .authorizedAlways:
536             break
537         }
538     }
539
540     func centreViewOnUserLocation ()
541     {
542         if let location = locationManager.location?.coordinate
543         {
544             let region = MKCoordinateRegion.init(center: location, latitudinalMeters: regionInMeters, longitudinalMeters: regionInMeters)
545             mapView.setRegion(region, animated: true)
546         }
547     }
548
549     @IBAction func centreButtonPressed(_ sender: Any)
550     {
551         centreViewOnUserLocation()
552     }
553
554     func calibratingFunc ()
555     {
556         //Only show alert if there is still an error with the user's GPS connection
557         if (calibrating == true)
558         {
559             presentCalibratingAlert()
560         }
561     }
562
563     func presentCalibratingAlert()
564     {
565         let alert = UIAlertController(title: "Calibrating...", message: "Currently we are calibrating and locating you so this app can run as smoothly as possible.", preferredStyle: .alert)
566
567         let action1 = UIAlertAction(title: "OK", style: .default, handler: nil)
568
569         let imgTitle = UIImage(named:"imgTitle.png")
570         let imgViewTitle = UIImageView(frame: CGRect(x: 10, y: 10, width: 30, height: 30))
571         imgViewTitle.image = imgTitle
572
573         alert.view.addSubview(imgViewTitle)
574         alert.addAction(action1)
575
576         self.present(alert, animated: true, completion: nil)
577     }
578 }
579
580 extension RaceViewController: CLLocationManagerDelegate
581 {
582     func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation])
583     {
584         guard let location = locations.last else { return }
585
586         //Instead of updating the region, the centre of the map is updated. This means that...
587         //...the user can zoom in and out as much as they want, but their location will always remain in the centre of the map.
588         if (location.distance(from: oldLocation) > 7)
589         {
590             if (location.distance(from: oldLocation) < 500)
591             {
592                 //print("--- CENTERING ---")
593                 let center = CLLocationCoordinate2D(latitude: location.coordinate.latitude, longitude: location.coordinate.longitude)
594                 mapView.setCenter(center, animated: true)
595             }
596             else
597             {
598                 centreViewOnUserLocation()
599             }
600             oldLocation = location
601         }
602
603         // Displaying Speed
604         speedInMPS = location.speed
605         //print("Speed (mps): \(speedInMPS)")
606
607         //If user has selected kilometres units
608         if milesOrKilometresSegment.selectedSegmentIndex == 0
609         {
610             if speedInMPS >= 0
611             {
612                 speedLabel.font = speedLabel.font.withSize(50)
613                 //rounds the calculated value, then trancates the decimal place, then converts it to String, then adds " kmh"
614                 speedLabel.text = String (Int((speedInMPS * 3.6).rounded())) + " kmh"
615                 calibrating = false
616             }
617             else
618             {
619                 speedLabel.font = speedLabel.font.withSize(30)
620                 speedLabel.text = "Calibrating..."
621                 calibrating = true
622                 calibratingFunc()
623             }
624         }
625
626         //If user has selected miles units
627         if milesOrKilometresSegment.selectedSegmentIndex == 1
628         {
629             if speedInMPS >= 0
630             {
631                 speedLabel.font = speedLabel.font.withSize(50)
632                 //rounds the calculated value, then trancates the decimal place, then converts it to String, then adds " mph"
633                 speedLabel.text = String (Int((speedInMPS * 2.23694).rounded())) + " mph"
634                 calibrating = false
635             }
636             else
637             {
638                 speedLabel.font = speedLabel.font.withSize(30)
639                 speedLabel.text = "Calibrating..."
640                 calibrating = true
```

```
641                 calibratingFunc()
642             }
643         }
644     }
645
646     func locationManager(_ manager: CLLocationManager, didChangeAuthorization status: CLAuthorizationStatus)
647     {
648         //If the user changes the location privacy in iOS settings, request permission to access location again.
649         checkLocationAuthorization()
650     }
651
652     func locationManager(_ manager: CLLocationManager, didFailWithError error: Error)
653     {
654         //If error has occured, print error.
655         print(error)
656     }
657 }
658
659 extension Double
660 {
661     func roundTo(places:Int) -> Double
662     {
663         let divisor = pow(10.0, Double(places))
664         return (self * divisor).rounded() / divisor
665     }
666 }
667
```

StatsHomeScreenController.Swift

```swift
//
//  StatsHomeViewController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 05/04/2021.
//

import Foundation
import UIKit

class StatsHomeScreenController: UIViewController
{
    @IBOutlet weak var speedSegment: UISegmentedControl!
    @IBOutlet weak var distanceDisplayView: UIView!
    @IBOutlet weak var distanceDisplayLabel: UILabel!

    override func viewDidLoad()
    {
        super.viewDidLoad()

//        RaceData.sharedInstance.speedDataArray = [[0.1, 0.4, 3, 5, 7], [0.04, 2.3, 5.7]]
//        RaceData.sharedInstance.timeDataArray = [[1, 2, 3], [1, 2, 3]]
//        RaceData.sharedInstance.distanceDataArray = [[1, 2, 3, 4, 20], [1, 2, 3]]
//        RaceData.sharedInstance.finalTimeArray = ["19.83", "22.54"]
//        RaceData.sharedInstance.averageSpeedArray = [4.35, 3.64]
//        RaceData.sharedInstance.topSpeedArray = [5.87, 5.64]
//        RaceData.sharedInstance.raceDistance = 100
//        RaceData.sharedInstance.playerInt = 1
//        RaceData.sharedInstance.playerString = "Player1"

        print("speedDataArray: \(RaceData.sharedInstance.speedDataArray)")
        print("distanceDataArray: \(RaceData.sharedInstance.distanceDataArray)")
        print("timeDataArray: \(RaceData.sharedInstance.timeDataArray)")
        print("finalTimeArray: \(RaceData.sharedInstance.finalTimeArray)")
        print("averageSpeedArray: \(RaceData.sharedInstance.averageSpeedArray)")
        print("topSpeedArray: \(RaceData.sharedInstance.topSpeedArray)")
        print("raceDistance: \(RaceData.sharedInstance.raceDistance)")
        print("player: \(RaceData.sharedInstance.playerString)")

        distanceDisplayView.layer.cornerRadius = 5
        //gets the distance from the RaceData class to display the race distance at the top of the screen
        distanceDisplayLabel.text = String(RaceData.sharedInstance.raceDistance) + "m"
    }

    @IBAction func speedSegmentChanged(_ sender: Any)
    {
        //set to kilometres
        if (speedSegment.selectedSegmentIndex == 0)
        {
            RaceData.sharedInstance.isKilometres = true
        }
        //set to miles
        else if (speedSegment.selectedSegmentIndex == 1)
        {
            RaceData.sharedInstance.isKilometres = false
        }
    }

    @IBAction func backToHomeButtonPressed(_ sender: Any)
    {
        let mainStoryboard = UIStoryboard(name: "Main", bundle: Bundle.main)
        if let destinationViewController = mainStoryboard.instantiateViewController(withIdentifier: "HomeScreenViewController") as? HomeScreenViewController {
            self.present(destinationViewController, animated: true, completion: nil)
        }
    }
}
```

AllStatsComparisonViewController.Swift

```swift
//
//  allStatsComparisonViewController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 05/04/2021.
//

import Foundation
import UIKit

class AllStatsComparisonViewController: UIViewController
{
    // - MARK: Outlets
    @IBOutlet weak var p1TimeLabel: UILabel!
    @IBOutlet weak var p1AverageSpeedLabel: UILabel!
    @IBOutlet weak var p1TopSpeedLabel: UILabel!

    @IBOutlet weak var p2TimeLabel: UILabel!
    @IBOutlet weak var p2AverageSpeedLabel: UILabel!
    @IBOutlet weak var p2TopSpeedLabel: UILabel!

    @IBOutlet weak var unitIndicatorLabel1: UILabel!
    @IBOutlet weak var unitIndicatorLabel2: UILabel!
    @IBOutlet weak var unitIndicatorLabel3: UILabel!
    @IBOutlet weak var unitIndicatorLabel4: UILabel!

    @IBOutlet weak var player1Label: UILabel!
    @IBOutlet weak var player2Label: UILabel!

    @IBOutlet weak var p1TimePositionLabel: UILabel!
    @IBOutlet weak var p1AverageSpeedPositionLabel: UILabel!
    @IBOutlet weak var p1TopSpeedPositionLabel: UILabel!

    @IBOutlet weak var p2TimePositionLabel: UILabel!
    @IBOutlet weak var p2AverageSpeedPositionLabel: UILabel!
    @IBOutlet weak var p2TopSpeedPositionLabel: UILabel!
    // - MARK: Variables

    // - MARK: ViewDidLoad
    override func viewDidLoad()
    {
        super.viewDidLoad()

        //shows each player's time
        p1TimeLabel.text = formatTime(timeLet: RaceData.sharedInstance.finalTimeArray[0])
        p2TimeLabel.text = formatTime(timeLet: RaceData.sharedInstance.finalTimeArray[1])

        //shows each player 1's average and top speed (in kilometres)
        p1AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[0] * 3.6).roundTo(places: 2)))
        p1TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[0] * 3.6).roundTo(places: 2)))

        //shows each player 2's average and top speed (in miles)
        p2AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[1] * 3.6).roundTo(places: 2)))
        p2TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[1] * 3.6).roundTo(places: 2)))

        //If the user is player 1, then the player 1 label should read 'You' and the player 2 label should read 'opponent'
        if (RaceData.sharedInstance.playerInt == 1)
        {
            player1Label.text = "You"
            player2Label.text = "Opponent"
        }
        //If the user is player 2, then the player 2 label should read 'You' and the player 1 label should read 'opponent'
        else
        {
            player2Label.text = "You"
            player1Label.text = "Opponent"
        }

        highlightWinners()
    }

    //Organisational function to highlight the winners of each stat
    func highlightWinners()
    {
        //Compares the times of each player - it is handled differently because you are checking for a lower (faster) time.
        if (Double(RaceData.sharedInstance.finalTimeArray[0])! < Double(RaceData.sharedInstance.finalTimeArray[1])!)
        {
            playerWins(winnerLabel: p1TimePositionLabel, loserLabel: p2TimePositionLabel)
        }
        else
        {
            playerWins(winnerLabel: p2TimePositionLabel, loserLabel: p1TimePositionLabel)
        }

        //Calls the function to compare the two average speeds
        compareTwoSpeeds(arr: RaceData.sharedInstance.averageSpeedArray, p1Label: p1AverageSpeedPositionLabel, p2Label: p2AverageSpeedPositionLabel)

        //Calls the function to compare the two top speeds
        compareTwoSpeeds(arr: RaceData.sharedInstance.topSpeedArray, p1Label: p1TopSpeedPositionLabel, p2Label: p2TopSpeedPositionLabel)
    }

    //Compares two speeds and calls the playerWins function depending on which speed is greater
    func compareTwoSpeeds(arr: [Double], p1Label: UILabel, p2Label: UILabel)
    {
        //If player 1's speed is greater than player 2's speed: player1 is the winnner, player2 is the loser.
        if (arr[0] > arr[1])
        {
            playerWins(winnerLabel: p1Label, loserLabel: p2Label)
        }
        //If player 2's speed is greater than player 1's speed: player2 is the winnner, player1 is the loser.
        else if (arr[1] > arr[0])
        {
            playerWins(winnerLabel: p2Label, loserLabel: p1Label)
        }
    }

    //Handles the formatting for the position labels
    func playerWins(winnerLabel: UILabel, loserLabel: UILabel)
    {
        //Loser's string: says '2nd' and has a silver background
        let myString = "2nd"
        let myAttribute = [ NSAttributedString.Key.backgroundColor: UIColor.init(red: 0/255, green: 0/255, blue: 0/255, alpha: 0.25)]
        let losingString = NSAttributedString(string: myString, attributes: myAttribute)

        //Winners's string: says '1st' and has a guld background
        let myString2 = "1st"
        let myAttribute2 = [ NSAttributedString.Key.backgroundColor: UIColor.init(red: 255/255, green: 211/255, blue: 0/255, alpha: 0.9)]
        let winningString = NSAttributedString(string: myString2, attributes: myAttribute2)

        winnerLabel.attributedText = winningString  //the winner label's text is set to the 'winningString'
        loserLabel.attributedText = losingString  //the losers label's text is set to the 'winningString'
    }

    //function which takes in a speed and returns the newly formatted speed
    func formatSpeeds (speedLet: String) -> String
    {
        var speed = speedLet
        let speedString = speed
        //If the speed is less than 10, add a 0 to the boggining of the speed:
        //e.g. 9.25kmh -> 09.25kmh
```

```
        if (Double(speed)! < 10)
        {
            speed = "0" + speed
        }
        //If the speed is divisible by 0.1, add a 0 to the end of the speed:
        //e.g. 3.2kmh -> 3.20kmh
        if(remainderOfDoubles(x: Double(speedString)!, y: 0.1) == 0)
        {
            speed = speed + "0"
        }
        return speed
    }

    //function which takes in a time and returns the newly formatted time
    func formatTime (timeLet: String) -> String
    {
        var time = timeLet
        var mins = 0
        var seconds = 0.0
        var minsString = ""
        //If the time in seconds is less than 10, add a 0 to the beggining of the time:
        //e.g. 8.35s -> 08.35s
        if (Double(time)! < 10)
        {
            time = "0" + time
        }
        mins = Int((Double(time)!/60).rounded(.down))   //works out the minutes from the time
        seconds = (Double(time)! - (Double(mins)*60)).roundTo(places: 2)    //works out the seconds remaining from the time and minutes
        //If the minutes is less than 10, add a 0 to the beggining of the minutes:
        //e.g. 8 minutes -> 08 minutes
        if (mins < 10)
        {
            minsString = "0" + String(mins)
        }
        else
        {
            minsString = String(mins)
        }
        //If the seconds is less than 10, add a 0 to the beggining of the seconds part of the string:
        //e.g. 12:4.83 -> 12:04.83     - 12 minutes, 4 seconds and 83 miliseconds
        if (seconds < 10)
        {
            time = String(minsString) + ":0" + String(seconds)
        }
        else
        {
            time = String(minsString) + ":" + String(seconds)
        }
        //If the seconds is divisible by than 0.1, add a 0 to the end of the time part of the string:
        //e.g. 03:45.6 -> 03:45.60    - 3 minutes, 45 seconds and 60 miliseconds
        if(remainderOfDoubles(x: seconds, y: 0.1) == 0)
        {
            time = time + "0"
        }
        return time
    }

    func remainderOfDoubles (x: Double, y: Double) -> Double
    {
        let a = Double(x/y).roundTo(places: 2)
        let b = Double(x/y).roundTo(places: 2).rounded(.down)
        return (a - b)
    }

    @IBAction func segmentValueChanged(_ sender: Any)
    {
        //km selected - multiply speeds by 3.6 to get from metres per second to kilometres per hour
        if (segment.selectedSegmentIndex == 0)
        {
            p1AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[0] * 3.6).roundTo(places: 2)))
            p1TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[0] * 3.6).roundTo(places: 2)))

            p2AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[1] * 3.6).roundTo(places: 2)))
            p2TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[1] * 3.6).roundTo(places: 2)))

            unitIndicatorLabel1.text = "km/h"
            unitIndicatorLabel2.text = "km/h"
            unitIndicatorLabel3.text = "km/h"
            unitIndicatorLabel4.text = "km/h"
        }
        //miles selected - multiply speeds by 2.23694 to get from metres per second to miles per hour
        else if (segment.selectedSegmentIndex == 1)
        {
            p1AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[0] * 2.23694).roundTo(places: 2)))
            p1TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[0] * 2.23694).roundTo(places: 2)))

            p2AverageSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.averageSpeedArray[1] * 2.23694).roundTo(places: 2)))
            p2TopSpeedLabel.text = formatSpeeds(speedLet: String((RaceData.sharedInstance.topSpeedArray[1] * 2.23694).roundTo(places: 2)))

            unitIndicatorLabel1.text = "mph"
            unitIndicatorLabel2.text = "mph"
            unitIndicatorLabel3.text = "mph"
            unitIndicatorLabel4.text = "mph"
        }
    }
}
```

SpeedComparisonViewController.Swift

```swift
//
//  SpeedComparisonViewController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 05/04/2021.
//

import Foundation
import UIKit
import SwiftChart

class SpeedComparisonViewController: UIViewController, ChartDelegate
{
    @IBOutlet weak var chart: Chart!

    @IBOutlet weak var player1AverageSpeedLabel: UILabel!
    @IBOutlet weak var player1TopSpeedLabel: UILabel!
    @IBOutlet weak var player2AverageSpeedLabel: UILabel!
    @IBOutlet weak var player2TopSpeedLabel: UILabel!

    @IBOutlet weak var player1DisplayLabel: UILabel!
    @IBOutlet weak var player2DisplayLabel: UILabel!

    var xLabelsArray: [Double] = []
    var yLabelsArray: [Double] = [0, 10, 20, 30]

    override func viewDidLoad()
    {
        super.viewDidLoad()
        chart.delegate = self

        player1AverageSpeedLabel.text =  String((RaceData.sharedInstance.averageSpeedArray [0]*3.6).roundTo(places: 2))
        player1TopSpeedLabel.text =  String((RaceData.sharedInstance.topSpeedArray [0]*3.6).roundTo(places: 2))
        player2AverageSpeedLabel.text =  String((RaceData.sharedInstance.averageSpeedArray [1]*3.6).roundTo(places: 2))
        player2TopSpeedLabel.text =  String((RaceData.sharedInstance.topSpeedArray [1]*3.6).roundTo(places: 2))

        if(RaceData.sharedInstance.playerInt == 1)
        {
            player1DisplayLabel.text = "You"
            player2DisplayLabel.text = "Opponent"
        }
        else
        {
            player2DisplayLabel.text = "You"
            player1DisplayLabel.text = "Opponent"
        }

        //*Formatting the data*
        let data1 = RaceData.sharedInstance.arrangeToData(player:  1, xArray:  RaceData.sharedInstance.distanceDataArray, yArray:  RaceData.sharedInstance.speedDataArray)
        let data2 = RaceData.sharedInstance.arrangeToData(player:  2, xArray:  RaceData.sharedInstance.distanceDataArray, yArray:  RaceData.sharedInstance.speedDataArray)

        fillAxis()  //fills the xLabelsArray.

        chart.xLabels = xLabelsArray    //the labels in the xLabelsArray are added to the chart
        //if the user went faster than 30kmh (i.e. a fast sprint) then add 40kmh to the yLabels array
        if (RaceData.sharedInstance.topSpeedArray[0]*3.6 > 30.0 || RaceData.sharedInstance.topSpeedArray[1]*3.6 > 30.0)
        {
            yLabelsArray.append(40)
        }
        chart.yLabels = yLabelsArray    //the labels in the yLabelsArray are added to the chart

        chart.xLabelsFormatter = { String(Int(round($1))) + "m" }    //adds a 'm' character to the last label on the x-axis
        chart.yLabelsFormatter = { String(Int(round($1))) + "km/h" }    //adds a 'km/h' character to the last label on the y-axis

        //Adds a series to the chart containing player 1's data
        let series1 = ChartSeries(data: data1)
        series1.color = ChartColors.blueColor()
        series1.area = true
        chart.add(series1)

        //Adds a series to the chart containing player 2's data
        let series2 = ChartSeries(data: data2)
        series2.color = ChartColors.redColor()
        series2.area = true
        chart.add(series2)
    }

    //fills the xLabelsArray by adding 5 equally spaced distances
    func fillAxis ()
    {
        let interval = RaceData.sharedInstance.raceDistance/5
        var x = 0
        while (x <= RaceData.sharedInstance.raceDistance)
        {
            xLabelsArray.append(Double(x))
            x+=interval
        }
     }

    func didTouchChart(_ chart: Chart, indexes: [Int?], x: Double, left: CGFloat) {
        //
    }

    func didFinishTouchingChart(_ chart: Chart) {
        //
    }

    func didEndTouchingChart(_ chart: Chart) {
        //
    }
}

```

PageViewController.Swift

```swift
//
//  PageViewController.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 05/04/2021.
//

import Foundation
import UIKit

class PageViewController: UIPageViewController, UIPageViewControllerDelegate, UIPageViewControllerDataSource {

    //initiliases the 3 screens which will be part of the page view controller
    lazy var orderedViewControllers: [UIViewController] = {

        let sb = UIStoryboard(name: "Main", bundle: nil)

        let vc1 = sb.instantiateViewController(withIdentifier: "AllStatsComparisonViewController")
        let vc2 = sb.instantiateViewController(withIdentifier: "SpeedComparisonViewController")

        return [vc1, vc2]
    }()

    var pageControl = UIPageControl()

    //when the view appears, set up the page view controller
    override func viewDidAppear(_ animated: Bool)
    {
        self.dataSource = self
        self.delegate = self
        configurePageControl()
        if let firstViewController = orderedViewControllers.first
        {
            setViewControllers([firstViewController], direction: .forward, animated: true, completion: nil)
        }
    }

    //configures the 3 dots which indicate which page the user is on
    func configurePageControl()
    {
        pageControl = UIPageControl(frame: CGRect(x: 0, y: self.view.bounds.maxY - 40, width: UIScreen.main.bounds.width, height: 50))
        pageControl.numberOfPages = orderedViewControllers.count
        pageControl.currentPage = 0
        pageControl.tintColor = UIColor.black
        pageControl.pageIndicatorTintColor = UIColor.gray
        pageControl.currentPageIndicatorTintColor = UIColor.black
        self.view.addSubview(pageControl)
    }

    //code for what happens if the user swipes right (backwards)
    func pageViewController(_ pageViewController: UIPageViewController, viewControllerBefore viewController: UIViewController) -> UIViewController?
    {
        guard let viewControllerIndex = orderedViewControllers.index(of: viewController) else
        {
            return nil
        }

        let previousIndex = viewControllerIndex - 1

        //if there is no previous page, then do nothing
        guard previousIndex >= 0 else
        {
            return nil
        }
        guard orderedViewControllers.count > previousIndex else
        {
            return nil
        }

        //if there is a previous page, then show this page
        return orderedViewControllers[previousIndex]
    }

    //code for what happens if the user swipes left (forwards)
    func pageViewController(_ pageViewController: UIPageViewController, viewControllerAfter viewController: UIViewController) -> UIViewController?
    {
        guard let viewControllerIndex = orderedViewControllers.index(of: viewController) else
        {
            return nil
        }

        let nextIndex = viewControllerIndex + 1

        //if there is no next page, then do nothing
        guard orderedViewControllers.count != nextIndex else
        {
            return nil
        }
        guard orderedViewControllers.count > nextIndex else
        {
            return nil
        }

        //if there is a next page, then show this page
        return orderedViewControllers[nextIndex]
    }

    //displays the current page with a transition
    func pageViewController(_ pageViewController: UIPageViewController, didFinishAnimating finished: Bool, previousViewControllers: [UIViewController], transitionCompleted completed: Bool)
    {
        let pageContentViewController = pageViewController.viewControllers![0]
        self.pageControl.currentPage = orderedViewControllers.index(of: pageContentViewController)!
    }
}

```

RaceData.Swift

```swift
//
//  RaceData.swift
//  RunWithFriendsComputingCoursework
//
//  Created by Joshua Graham on 02/04/2021.
//

import Foundation
import UIKit

class RaceData
{
    static let sharedInstance = RaceData()

    var speedDataArray: [[Double]] = [[], []]       //2D arrays to hold each players speed over the course of the race
    var distanceDataArray: [[Double]] = [[], []]      //2D arrays to hold each players distance over the course of the race
    var timeDataArray: [[Double]] = [[], []]      //2D arrays to hold each players time over the course of the race

    var finalTimeArray: [String] = ["", ""]       //2D arrays to hold each players final time
    var averageSpeedArray: [Double] = [0.0, 0.0]       //2D arrays to hold each players average speed
    var topSpeedArray: [Double] = [0.0, 0.0]       //2D arrays to hold each players top speed

    var raceDistance: Int = 0      //integer to hold the race distance
    var playerString: String = ""      //string to hold the current player: "player1" or "player2"
    var playerInt: Int = 0      //integer to hold the current player: 1 or 2

    var isKilometres: Bool = true      //boolean which determines whether the stats should be shown in km or m.

    //Takes the two arrays of a specific player and formats each element into an array...
    //... of tuples so it can be used to add the series to a graph.
    func arrangeToData (player: Int, xArray: [[Double]], yArray: [[Double]]) -> [(x: Double, y: Double)]
    {
        var data: [(x: Double, y: Double)] = []     //dummy array to be returned
        for i in 0...xArray[player-1].count-1   //loops to the end of the specified xArray
        {
            //appends to the data array the element at the current index in both the xArray and yArray
            data.append((x: xArray[player-1][i], y: yArray[player-1][i]*3.6))
        }
        return data
    }
}

```