Instituto Superior Técnico

Wireless Mobile Networks

Final Submission - June 3, 2018

André Ferreira - 81715

Bruno Alves - 81684

1 Introduction

The project consists on an app that corresponds to a virtual firing game, where the players can try to shoot against other players, in the pursuit of achieving the maximum score. Meanwhile, players should avoid being shot, as besides increasing the opponent's score, it kills them, preventing them from playing during a predefined dead time. In order to understand each player's position and shooting aim, the game needs to use the GPS location and the orientation given by the compass and accelerometer sensors of each player's smartphone. These individual processes of each player require a constant communication between a server that contains the information of all players and the app, which is the interface that interacts with the players.

2 Structure of the game

In order to accomplish the goals of the project, essential decisions were made about how the game would evolve. As game rules, the fixed parameters found on Table 1 were established. Then, assuming that there are a set of players in the game, the behaviour of the app will follow the flowchart shown in Figure 1. First, the player performs its log in. This process will be explained in more detail in Section 2.3. Then, there will be a verification if the player is "dead" or not. If one player is inside the firing range of another player that has shot, the target player will have a penalty in its score and it will not be allowed to shoot over another player during a certain time (10 seconds). This is like a respawn time where the player cannot be shot again and has time to reach a secure position. On the other hand, the dead player cannot shoot others as well. When this time ends, the player is not dead anymore and can return to normal playing conditions. When the player shoots, it is verified if that shot reaches someone and if it is the case, the status of the victim should be changed. However, if the player is alive, it can be shot again, and in that case the application goes to the beginning of this explanation and the process starts again.

The goal of the game is to reach the predefined score as soon as possible. So the first player to reach 20 points wins and the game is over. When the game terminates the top scores are displayed and each player has the opportunity to start a new game.

These individual processes of each player require a constant communication between something that contains the information of each player and the app which is the interface that communicate with the player.

Fixed Parameters	
Maximum score	20 points
Maximum shooting distance	30 meters
Respawn time	10 seconds
Pop up time	1 second

Table 1: Table with the fixed parameters that set the game rules.

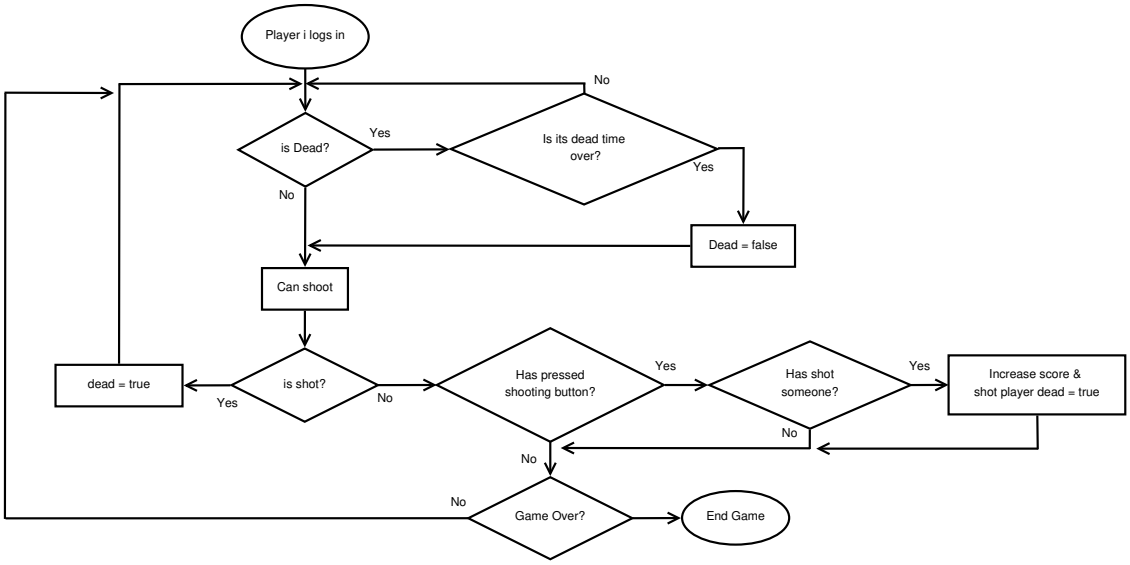


Figure 1: Flowchart of each player.

2.1 Communication of the data

In this application, there are different data transactions between the server, database and app, as stated in Figure 2. The app of the corresponding player should be constantly sending the information of its position and orientation to the database. Theoretically, this rate of transmission should be the highest possible so that the information of the player would be always updated in the database. However, it is important to think about the consequences of that strategy. For instance, if the user was concerned about the energy consumption of the app, more transmissions would mean more battery used. Furthermore, the Firebase platform has a limited amount of data transmitted for free users, which consists on a clear limitation. This detail will have more prominence in Section 4.

When the player pulls the trigger, the app will request the list of players. After performing the calculations specified on Section 2.4, a notification must be sent to the player or players that were shot. If there was a GPS signal without precision errors, only one player could be damaged on a single shot. As the app of each player is frequently requesting the *isDead* status, if the player is shot the app must notify it and not allow it to shoot.

Considering the user experience, it is essential for the player to have the information of the current score printed on the app interface, so the app should request that data from the database, according to a specific rate.

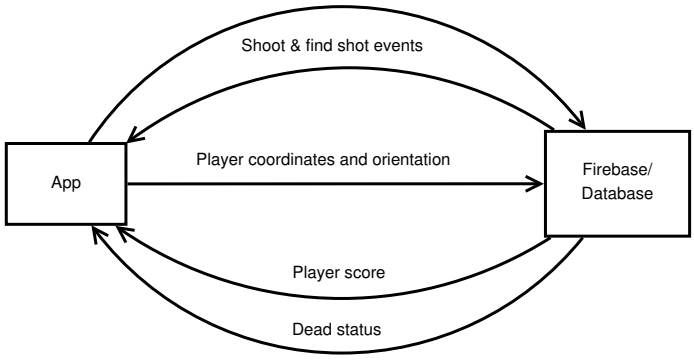


Figure 2: Communication diagram designed.

2.2 Database and server

Before implementing the database, it was necessary to establish the required data for this application and the corresponding organization. In the context of the project, it was decided that it would be used three entities, the Player, the Coordinates and the Orientations, as it is shown in Figure 3 . The first one contains the information of the player, namely its name, id, score, an indicator of either being alive or dead and an indicator of either being logged in or not. The other ones save three values, the latitude, longitude and altitude for the Coordinates, and the azimuth, pitch and roll for the Orientations. Each Player will also have one Coordinates and one Orientations association, and each Coordinates and Orientations can only belong to one Player.

At the beginning, this database was implemented using *mySQL*. However, after a few research about how to perform the communication between the smartphone and the server, the *Firebase* platform by *Google* was found. This platform can perform the role of the database and the server at the same time, which brings some advantages and drawbacks that are discussed in Section 4.

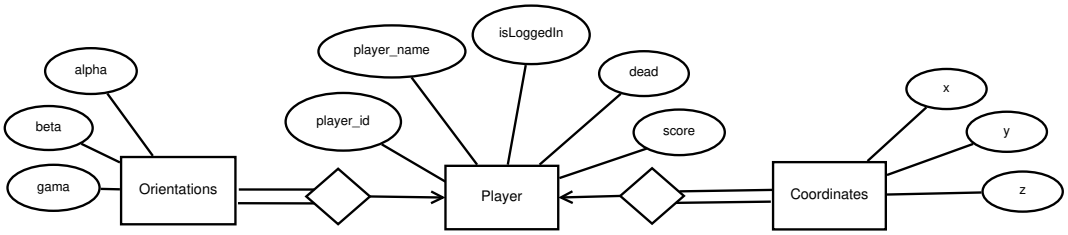


Figure 3: Structure of the database used in the implementation of the game.

Considering this logic, an example of the a game running with a Firebase implementation can be found on Figure 4. Firebase Realtime Database works with the JSON format, which stands for JavaScript Object Notation. This way, data is represented as text, structured by nodes and possible children nodes, which resemble objects in Javascript. This format for data as advantages, as it's discussed on Section 4.

In this particular case, shown in Figure 4, there are 2 players active on the game and it's possible to see their detailed information. Looking at the second player's current status, one can see his coordinates, know that he is currently alive and logged in, his id, his name, his orientation and his score, which is far away from 20, where he would win the game (see Table 1). The yellow attributes correspond to the attributes

Figure 4: Structure of the database viewed on Firebase Realtime Database.

that are currently being updated. In contrast, the first player does not have yellow attributes because he is logged out of the game and their coordinates and orientations cannot be updated. When comparing the scores of both players, one can see that the second player is closer to win the game. In the case of winning the game, the database should be reset, ending the current session of the game and wait for players to begin the game with a score of zero points. That being said, it's possible to store the scores of the game that ended in separate data nodes, in order to have a history record of the previous scores. This correspond to an extra feature to introduce in the game (see Section 5).

2.3 Authentication and Security

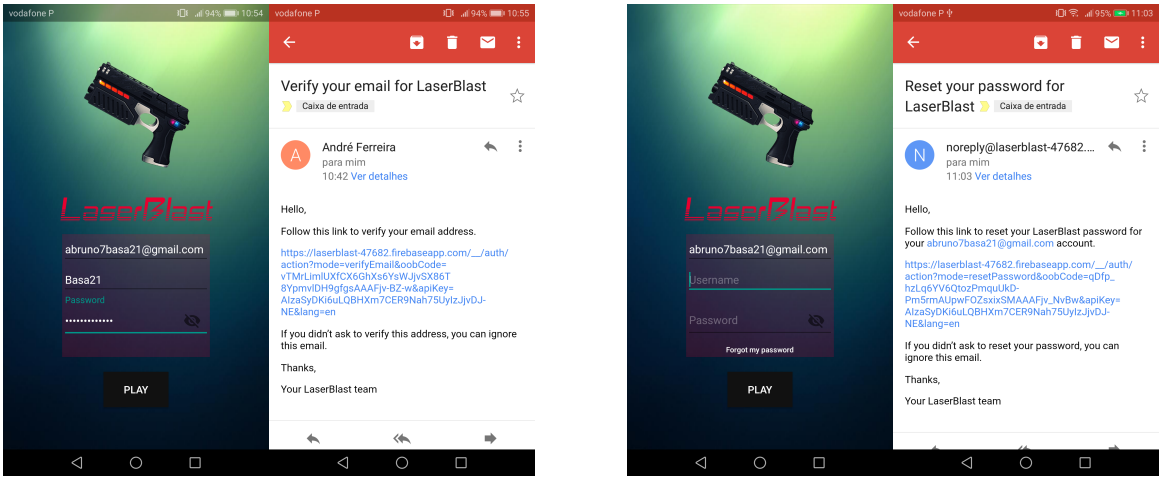
The intermediate delivery contained the first version of the application. For this version, in the log in activity, it was asked from the user a nickname to enter the game. Then, the app requested the list of current players from the Firebase's database. If that user name was already taken, the user would be notified and asked to try another one.

For the final delivery, new features were included in the authentication part. Now it is asked if the user wants to sign up, corresponding to creating a new account in the application, or to sign in, corresponding to loading an existing profile. If a user wants to sign up, it is requested an email, an username and a password to create a profile. Then, if that email is not already being used, the password is larger than 6 characters, and the username is not already taken, the procedure continues and it is sent a verification email. Afterwards, the

user can verify its email as it's shown on Figure 5a. During the sign in process, there is also the possibility to reset the password of the account (see Figure 5b), as it is common in every app nowadays. For the sign out functionality, it's done either when there's a gameover or when the user closes the app, for which we use the activity's *OnDestroy*, a method that is invoked every time the activity is closed, except when changing between activities.

Although, in this new authentication part, two more attributes were requested (email and password), it was decided not to introduce them in the structure of the player, as presented in Figure 12, to avoid saving it to local memory. As this information is still accessible from the Firebase's server, it is ensured more safety for these crucial attributes while keeping access to the information.

From now on the users have the possibility to sign out from the game and came back without having to create a new player. They can also restart the game with the same score if no other player reached the maximum score.



(a) (b)

Figure 5: Demonstration of the (a) verification email and (b) resetting password procedure.

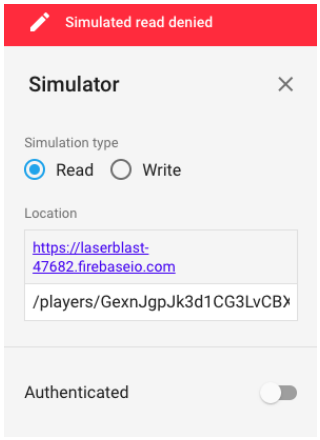
In order to prevent data privacy issues and reduce the risk of attacks to delicate information, Firebase security rules were implemented for the final version of the app. This feature allows developers to set conditions to determine who and when can access certain parts of the database, in terms of reading and writing operations. As such, through a script format that resembles JSON, we defined that only users that are currently logged in, with an authenticated account, can read or write players data, and any attribute of the class as seen on figure 12, except for a special case in the writing operations. Considering the way that the data is structured, where every player is referred to by it's unique identification, the code searches for a node with the name of the UID of the person using the app (auth.uid) and, if there is such a node, it will see if it has the attribute "isLoggedIn" with a "true" value.

When adding a new player, he or she still doesn't have an authenticated account, which would prevent adding the player's object to the database because of the security rules. To solve this situation, we add a non readable boolean called "FirstAdd" (figure 4) that allows writing operations when it has a value of "true". This way, whenever a player is being added, this boolean is made to be "true" in the very instant of uploading the player to the database, and turned back to "false" afterwards. While this does mean that an

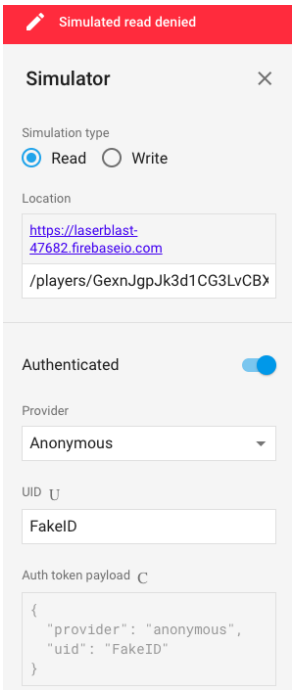
hacker could attack on the precise moment that a player is being created, he could only do writing operations under the "players" node, without ever being able to read either players' information or the value of the "FirstAdd" boolean. The script of the Firebase security rules can be seen below.

```
{
  "rules": {
        "FirstAdd": {
                ".read": "false",
                ".write": "true"
      },
        "players": {
                ".read": "root.child('players').child(auth.uid).child('
                   isLoggedIn').val() == true",
                ".write": "root.child('players').child(auth.uid).child('
                   isLoggedIn').val() == true || root.child('FirstAdd').
                   val() == true"
        }
  }
}
```

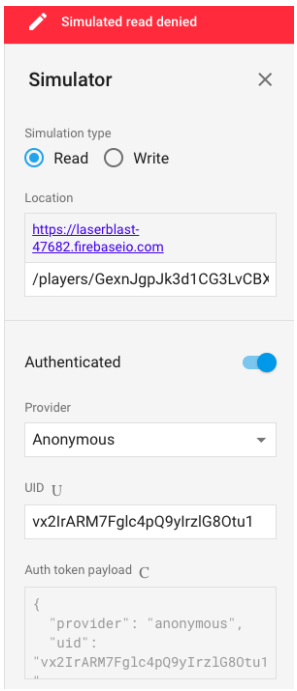
In the Firebase console, there's an interesting simulator that allows to test the security rules through reading and writing commands. This constitutes a fast method to debug the app's data permissions and experiment with real data that is currently in the database, without ever changing its values. The results of simulations for reading the player "Enemy"'s score (from the data of figure 4) are shown in figure 6, demonstrating the desired functionality. Namely, it isn't possible to read the player's data if the person doing the query isn't authenticated (figure 6a), if it's using a fake or incorrect ID (figure 6b) and if it's using a valid ID but from a player that currently isn't signed in (figure 6c). As such, it will only grant acess to "Enemy"'s score if the user is authenticated and logged in, as is the case of the player "Andre" (figure 6d).



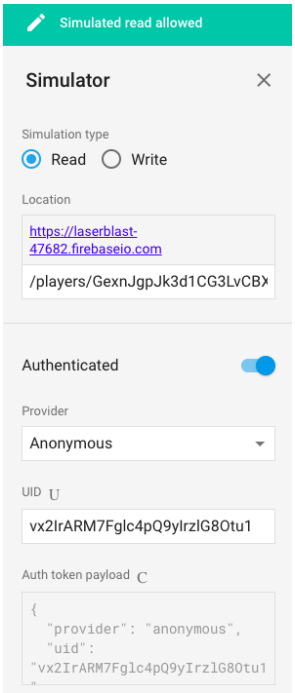
(a) Simulation without authentication.



(b) Simulation with incorrect authentication.



(c) Simulation with correct authentication of a logged out user.



(d) Simulation with correct authentication of a logged in user.

Figure 6: Security simulations, through reading the score of a player.

2.4 Shooting algorithm

As explained before, the game must have access to location and orientation information about each player. To do so, GPS and a mix of compass/accelerometer are used, respectively. These sensors give access to the following variables:

$$lat = \text{Latitude in degrees}$$

lon = Longitude in degrees
h = Height above sea level in meters
$azimuth$ = Angle in the XY plane, in radians.
$roll$ = = Angle in the XZ plane, in radians.
$pitch$ = Angle in the YZ plane, in radians.

Visually, these variables can be represented as shown on Figure 7.

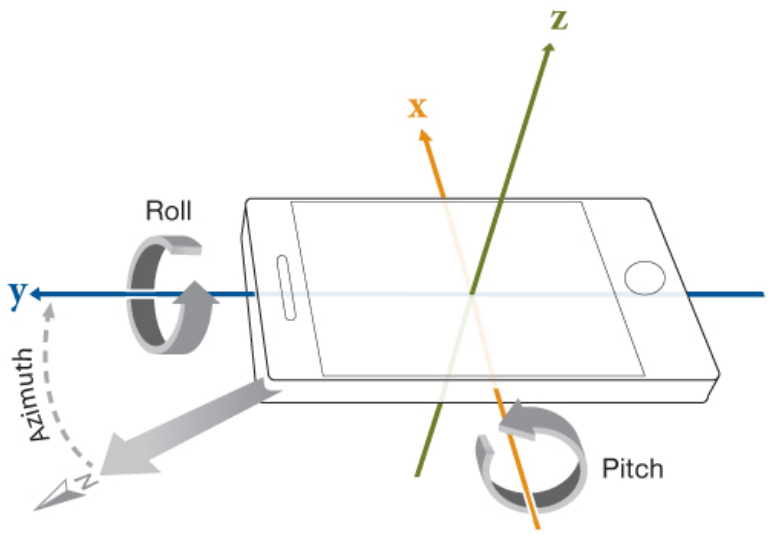


Figure 7: Location and orientation variables of a smartphone.

Having this data, it's set that a player B is successfully shot by player A if the following conditions are met:

- Player B is within a maximum distance of 30 meters of player A.
- The XY angle (bearing) between the locations of player B and A corresponds to player A's azimuth.

Although it might seem a very simple problem, due to Earth being round it's not possible to directly use latitude and longitude to calculate the distances and angles in basic Pythagorean equations. However, as suggested by Chris Veness [1], using a flat Earth approximation for small distances, it's possible to calculate the distance d and the XY angle $brng$ between a player A and a player B using the following equations:

$lat_{A_r} = lat_A \frac{\pi}{180}$ = Latitude of player A converted to radians (same goes for player B and for longitude).
$R = 6371000 \quad [m]$ = Earth's radius in meters.

$$d = \sqrt{((lon_{B_r} - lonA_r)cos((lat_{A_r} + lat_{B_r})/2))^2 + (lat_{B_r} - lat_{A_r})^2} R$$

$$brng = atan2\left(\frac{sin(lon_{B_r} - lon_{A_r})cos(lat_{B_r})}{cos(lat_{A_r})sin(lat_{B_r}) - sin(lat_{A_r})cos(lat_{B_r})cos(lon_{B_r} - lon_{A_r})}\right) \quad [rads] \qquad (1)$$

Using this equations, it's possible to detect if the distance between two players is less than 30 meters (the predefined maximum shooting distance) and if the XY angle is similar to the azimuth, the XY orientation, of player A (the one that shoots). Due to GPS, compass and accelerometer precision errors, this method is only acceptably accurate outdoors and away from objects with a big magnetic influence. Even if these conditions are met, there's still about 5 meters of location error just considering the current smartphone GPS problems, which makes it difficult to have a satisfying functionality in terms of the shooting calculations.

As it is shown in Figure 7, the azimuth measures the angle of the smartphone in that configuration with the North direction. However, the game was developed to shoot according to the configuration presented in Figure 8. From this, by performing a simple transformation such as adding $\pi/2$ to the measured azimuth, the desired orientation was obtained. This transformation creates a new problem related to compatibility between the bearing angle and one given by the azimuth. Before changing the referential, the angles used from the azimuth were between $-\pi$ and π. After adding $\pi/2$, interval changes to $-\pi/2$ and $3\pi/2$. In order to have the same interval for both variables, bearing and azimuth plus transformation, the last ones were modified according to

$$\begin{cases} azimuth + \frac{\pi}{2} - 2\pi, & azimuth + \frac{\pi}{2} > \pi \\ azimuth + \frac{\pi}{2}, & otherwise \end{cases}.$$

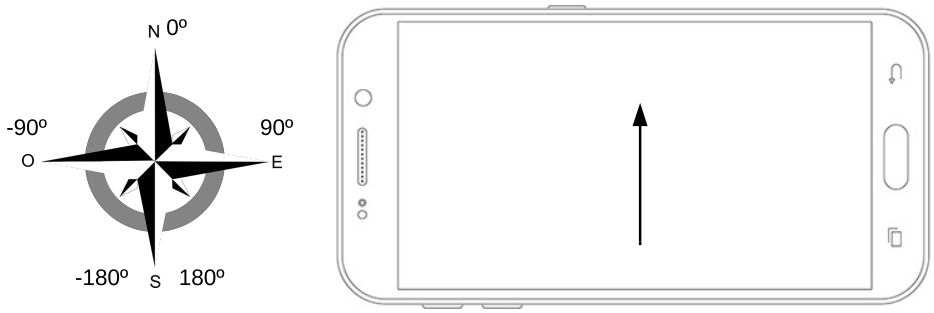
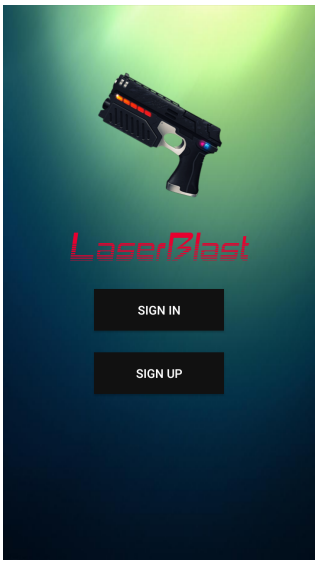


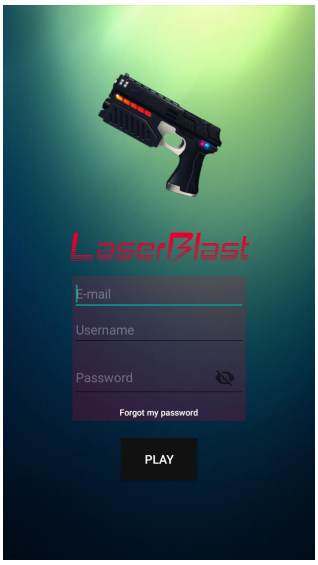
Figure 8: Transformation in the considered orientation.

3 Structure of the app

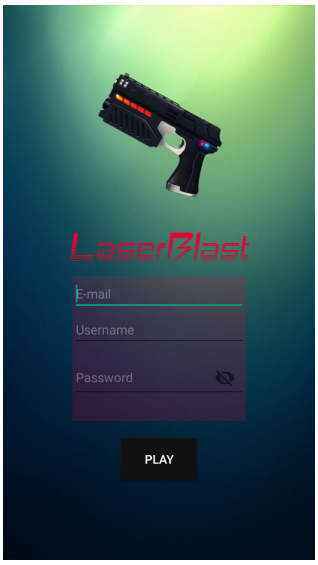
The android code is composed by three activities. The initial activity is the login activity (see Figure 9), where there are the options to either sign in to an already existing account or sign up and create a new one. Regardless of the selected option, an UI will appear, where the user can enter his or her credentials, by selecting an username, different from the rest of the players of the game, inputting the e-mail to be associated to the account and choosing a password that needs to be at least six characters long. To allow privacy for the user, the password field is defined to show each inserted character as a dot, to hide the password, although this can be disabled by the player. In the case that the user selects the sign in option, there's a "forgot my password" button, that sends an e-mail to reset the password associated to the inserted e-mail. Afterwards, the player can hit the Play button to enter the game and the main activity, as long as the credentials are correct. Previously, this activity only had a text space to fill with the username and a button to start playing the game.



(a) Initial layout of the login activity.



(b) Sign in UI

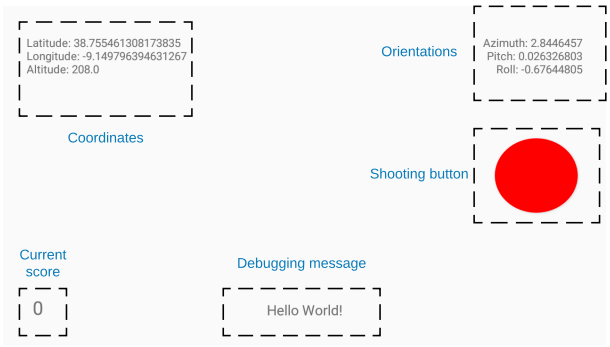


(c) Sign up UI

Figure 9: The three possible UIs of the login activity, with the sign in and sign up options.

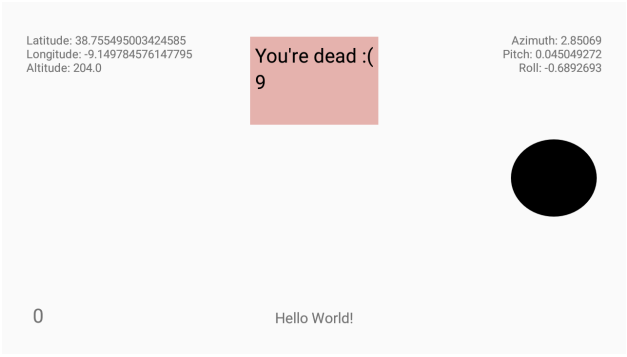
The second one is the game activity (Figure 10) where the player can shoot others, as well as have access to information related to its life status, shot players, current coordinates and orientation, and now also the current score. The player has a button to shoot, which will print a message in the screen informing if it was a clean shot or not, as well as the names of those that were hit by it. When the player is not allowed to shoot, due to being dead, the shoot button is disabled until respawn, which happens 10 seconds after dying.

As the app isn't in a production setup ready to be published to the general public, there's now a string in the lower center that initially shows an "Hello World!" message but, every time the shooting button is



(a) Standard view, with no events registered. The dashed boxes and blue texts serve as a reference for the meaning of every part of the screen.

(b) Layout with a temporary window indicating an effective shot and the name of the shot players.



(c) Layout when the player is dead, having the shooting button disabled and a message with a timer indicating the seconds left to respawn.

Figure 10: Different possible views inside the main activity.

pressed, it switches to the azimuth, the bearing and their calculated difference, between the current player and the last analyzed player of the database. This isn't something to be present in a final version, but rather a debugging feature for us to see if the angles and the shooting method are functioning correctly.

The last activity (Figure 11) is invoked when the game is over, and it prints the scores of the players, in a descending order. It also has a button that allows to start another game, with all the scores reset to 0, by invoking the login activity.

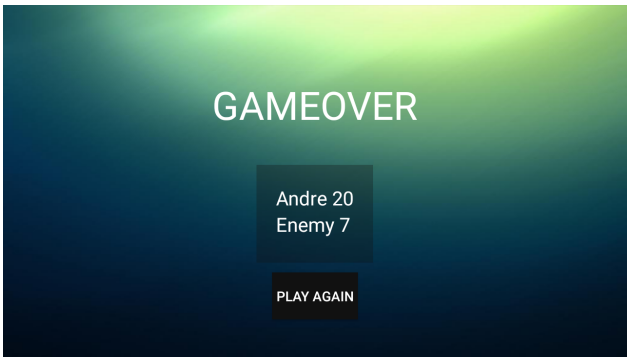


Figure 11: Layout of the gameover activity, indicating the players sorted by their score in a descending order.

The implementation of the game requires the class diagram presented in Figure 12. The Player has all the fields as indicated in the database. The ThreeDCharact has three fields that can be used to represent Coordinates or Orientation. This class has an unused reference to a date object, which could be used in the future to allow having an record of the coordinates in which each player has been in throughout the game.

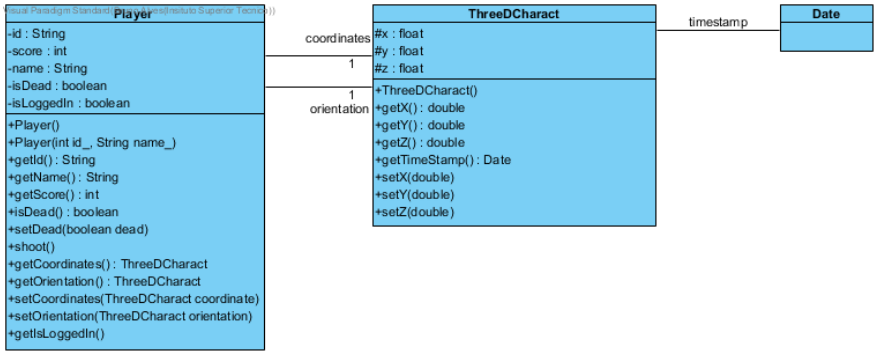


Figure 12: Design of the class diagram required for the application.

4 Analysis of the *Firebase* platform

Initially, it was studied the possibility of using the usual SQL solution to build the database. This would require the development of the database in SQL, setting up a server, the communications between the database and the server, using PHP, as well as the communications between the server and the app. As such, it implied building a database and server solution from a low level, implementing from scratch a significant part of this infrastructure and requiring a vast knowledge and experience to build a complex, interesting and safe product. Meanwhile, when searching for possible alternatives, it was found Firebase. This platform, besides many other app and web related services, has available a database solution called Firebase Realtime Database. As was said in section 2.2, data in this platform is structured according to the JSON format. Due to its shape of nodes and children nodes, it makes it very easy to use in object oriented languages such as Java. For example, when handling the login of a new player in the game, it's just a one line code to create and upload all the structured information regarding the new player to the database. In fact, this format allows to handle the information inside the database practically in the same way as local variables are handled in the Android application. With this ease of data access from Android, it's possible to put the main processing load on each smartphone, decrease the number and duration of downloads and uploads, as well as decrease the risk of server overload.

Having their own website, Firebase allows an intuitive and effective view of the database, as seen on Figure 4. As it all works in realtime, the second a change is uploaded from some smartphone connected to the server, it immediately appears on the database and can be viewed in a dynamic way.

Firebase enables the setting of security rules, which consist on a simple way to configure private and correct use of the data. Just by writing readable expressions directly on the Firebase console, it's possible to establish read access, write access and data validation rules. Besides that, being it a Google product, it's likely to be more resistant to server bugs and overloads. Although it wasn't yet explored for this project, there's also the possibility to interact with the database in a serverless way, writing functions in JavaScript that are able to change behaviours of the app according to the data, without having to change the app's code itself. Among additional features, it's also possible to use the data offline, syncing the data when back online.

```
{
  "rules": {
    "users": {
      "$user_id": {
        // grants write access to the owner of this user account
        // whose uid must exactly match the key ($user_id)
        ".write": "$user_id === auth.uid"
      }
    }
  }
}
```

Figure 13: Example of security rules in Firebase Realtime Database.

However, there are disadvantages in using Firebase Realtime Database. As a service that allows a more intuitive and high level solution to develop and manage databases, the free plan available has some limitations. As seen on figure 14, it's only allowed to have up to 100 simultaneous connections to the database, it can only

store up to 1 gb and the total downloads per month can't exceed 10 gb. In order to have less restrictions and access to more features, it's necessary to buy a monthly plan or a pay as you go plan that can still become a big economic constraint for a project with low revenue. As such, for small projects that need more connections, storage, higher download limits or other features, it's likely better to have a custom SQL solution.

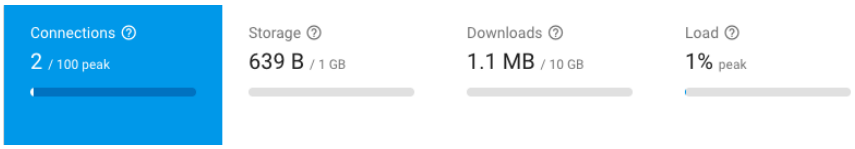


Figure 14: Limitations of the free plan, viewed on the Firebase console.

In summary, after spending some time implementing the game, a set of advantages was found such as:

- There is a huge ease to see the actual contents in the database. It is also possible to manage the data directly from the Firebase console;
- The security in the database is reliable and easy to implement. It would be impossible to construct such a secure database in *mySql*, for instance, as it is done in the *Google* platform. It's used by large scale app and web companies such as Shazam and Skyscanner;
- Handling data in JSON format, improving information processing in Android;
- Straightforward e-mail authentication functionality, with few lines needed to implement verification e-mails and password reset e-mails, and also has additional options to allow a user to enter through its Google, Facebook or Twitter account.
- Access to more features and greater scalability if using a paid plan.

In contrast, the drawbacks found are

- Limits in simultaneous connections in the free plan;
- Limits in storage capacity in the free plan;
- Limits in downloads per month in the free plan;
- Not as much customization when compared to a lower level solution such as SQL paired with PHP.

5 Future Improvements

After spending some time thinking about the game, it is possible to find several ways to crack the behaviour of the game. For instance, a player can try shooting someone by just constantly pressing the shoot button while rotating to all the directions. In order to avoid this, a charging time could be added. In other words, if a player spends all its "bullets", it must wait a certain time, corresponding to the time needed to recharge its gun.

Although the accuracy of the GPS signal is not the best possible, the shooting function could possibly be more accurate. Besides this improvement, the game could become more interesting by adding different types

of weapons, such as land mines, where the player could mark a certain region of the map and, if another player enters in that zone, it dies; grenades where the player can victimize more opponents but with the cost of a shorter range; or guns with different ranges, precision and damage capability.

Considering that each user can have coordinates associated with a timestamp, and interesting idea would be to have a list of the coordinates of the places where each player has been throughout a game, to allow to check the chronology of player's location. Furthermore, every time that a game ends, the final score of each player could be stored in a scores list of the database, to see and compare the performances of the players.

6 Evaluation of the application

The assessment of this application will be based on these topics,

- Scalability of the app;
- Obtained precision during tests;
- Applicability of the app in real life.

One of the issues that was not so much shaped, was the scalability of the game. The game was designed for a small set of players playing with each other. In other words, the game is global, there are no parties or sub servers (small set of players). Furthermore, when any player reaches the maximum score, the game is over and the scores of every player are reset. As the game is global, there would be some problems because, for instance, a set of players in Lisbon would cause the resetting for a set of players in Porto, which does not make sense since they are not playing directly against each other. This could be arranged by adding another level of data nodes to the database, that would store the lists of players that are playing an independent game with each other. However, there would always be a total maximum number of 100 players, counting with all the parties at the same time, due to the limits of Firebase's free version. In the paid version of the Firebase service, it's possible to have more players and to build separate databases for each game.

Relatively to the precision of the game. One can consider two different regions. When the players are close to each other, more precisely, a distance less than 5 meters, which corresponds to the GPS uncertainty, the GPS can crash the shooting algorithm. According to Figure 15, due to localization imprecision, both configurations can be found in this scenario, and both cause completely different results for the bearing angle. For instance, if the real positions are the ones on the left, but the GPS gives the positions on the right, the app does not have information about the reality. When the players are in have distance between each other higher than the GPS uncertainty, the shooting algorithm performs better. Despite the uncertainty of the GPS, it was introduced a threshold for the difference between the bearing angle and the angle related to the azimuth. This threshold was empirically established to 25 degrees, so that the majority of the shots in the right direction could be identified. This value can be considered as significant, regarding the fact the weapon's aperture is considering almost 15 per cent of the total aperture (25/180). In addiction to the optimal values given by the GPS, the influence of being closer to buildings can also deteriorate its measures.

There is another component that has an influence in the precision of the game. The angles given by the sensors of the smartphones, with special prominence to the compass one (azimuth), are susceptible to the environment surrounding the player. When the user is near sources of magnetic fields such as electrical wires, electronic devices, usually presented indoor, the measures given by the sensors start to diverge from

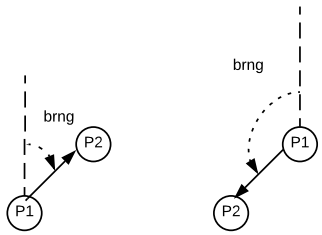


Figure 15: Configurations showing the problem of having two players really close to each other.

the real ones. This influence, which was not possible to quantify objectively, will definitely not help the performance of the game.

After playing the game, it was identified some problems that could hamper the introduction of this app in real life. One of them is the possibility of killing players that are covered behind the walls. Using only GPS and orientation sensors, it's not possible to overcome this difficulty. One possible solution could be adding infrared sensors as it is already used in similar applications in the market, or add computer vision features. The main problem consists of the uncertainty given by the GPS nowadays. The context of the game requires a constant proximity between players that is often below the boundary of the GPS uncertainty. This means that in the majority of the cases, the game will operate in the region where the performance is not good. Thus, as the GPS is not good enough, this application is not sufficiently robust to enter in real market.

References

[1] Chris Veness, Calculate distance, bearing and more between Latitude/Longitude points, 2002-2017 `http://www.movable-type.co.uk/scripts/latlong.html`