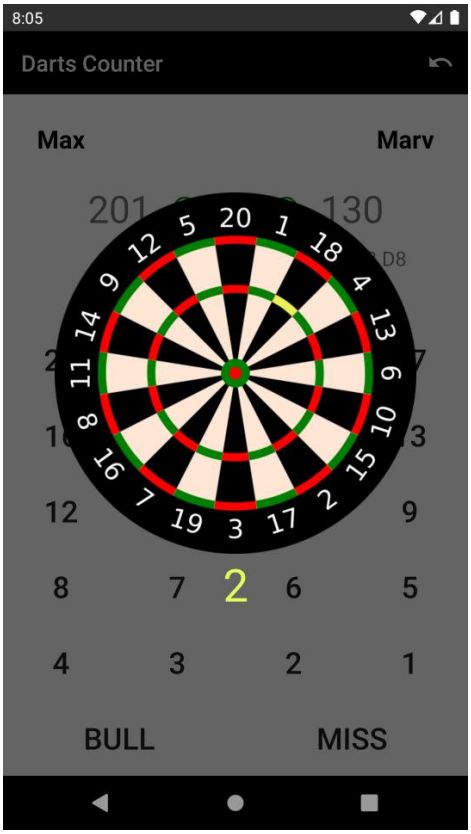
Mobile Application Development with Android

Developing a dart counting app



Adrian Kröger (21872150)

Marvin Kühn (21767036)

adrian.kroeger@stud.uni-goettingen.de

marvin.kuehn@stud.uni-goettingen.de

Table of contents

1 Project proposal

1.1 Short description

Our idea for this project was to develop a darts counting app. This means that we do not develop a game, but rather a helping tool for counting (e.g. with default rules) from 501 points to 0 points, while respecting the logics of the game like three darts per throw, busting if you throw too much points or handling the needed double finish to end the game. Both of us play darts as a hobby in our free time, but we weren't fully happy with the existing counting apps. The main problem for most apps is the paywall behind statistics and databases. We want to track our progress for free and with great detail, like average scores, average finishes and highest checkouts. This inherently involves player profiles so you are motivated to compete with your friends with this app. Furthermore, if there is enough time left, we would like to provide some settings for different preferences or even game modes.

1.2 Functional specification

We want the app to be fast and to the point. This means we will have a main menu, from which you can reach the different activities, such as the settings, the statistics and the actual counting activity. The statistics will be realized through a local room database, while in the counting activity, there will be suggestions for finishes, based on the given data. The goal is to distract the player as little as possible and make the input of the points scored intuitive, so they can focus on the game. After the game is finished, we want to provide some options like saving the current game to a file or directly sharing it with others.

1.3 List of features

1.3.1 Must have features

Core functionality

- intuitive UI to input scores
- display of ways to finish current score
- choosing between several common gamemodes (x01, double-in)

Statistics

- keeping track of match data
- maintaining this in a database
- straightforward display of statistics for each player

Recording of Match

- generating simplified representation of match
- saving or sharing match outcome

1.3.2 Nice to have features

Interaction between apps

- online mode via P2P score communication
- importing/exporting match data to another user's database

Additional gamemodes

- implementation of more sophisticated gamemodes (Cricket, Clock, …)

Customization

- multiple language support
- choice of different UI themes

1.4 Timeline

Description	Deadline	Member
Project setup	07.07.2020	Member 2
Visuals & UI	14.07.2020	Member 2
Game Logic	14.07.2020	Member 1
Recording of match & sharing	17.07.2020	Member 1
Database implementation	17.07.2020	Member 2
Cleanup & assessment of nice to have features	20.07.2020	Member 1 & Member 2
Implementation of optional features	12.08.2020	Member 1 & Member 2
Final cleanup and documentation	21.08.2020	Member 1 & Member 2

Where "Member 1" is Marvin and "Member 2" is Adrian.

2 Project Structure

2.1 MainActivity

The top of our MainActivity is used as a main menu, where the toolbars function is to either navigate towards the settings or the statistics. The rest of the top half is setup in a way to quickly start a match between two players. You will find all necessary settings for a darts match, like choosing between game-modes, setting up the points or even a handicap. You will also notice the fields where you should put the player names and a button to add new players. This is connected to a database. Players participating in a game will have their stats tracked throughout their played matches.

The bottom half of the MainActivity is utilizing a ListView to show games, which were stored to a file. Saved games should include all necessary data to recreate a full game played from the first dart to the last. To navigate towards the actual game, the “play” button should be pressed while all information is filled in.

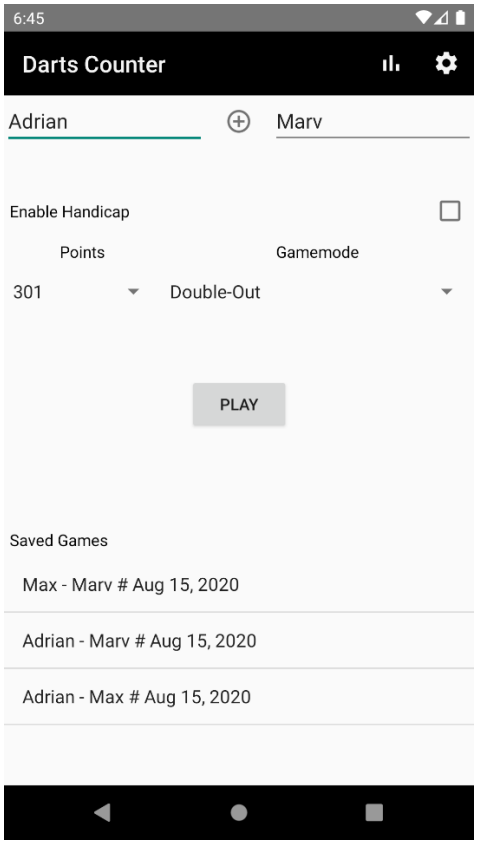


Fig 1: MainActivity with some saved games

2.1.1 MatchHistoryDialogFragment

If you click on a saved match, this fragment will open up and show the game as it is standard in official matches. On the right side of a player we show how many points the player has left and on the left side we show the points which were scored. A horizontal line indicates that a turn has ended, which means either 3 darts were thrown or the player has ended the turn in another way (e.g.: finish was not scored with a double).

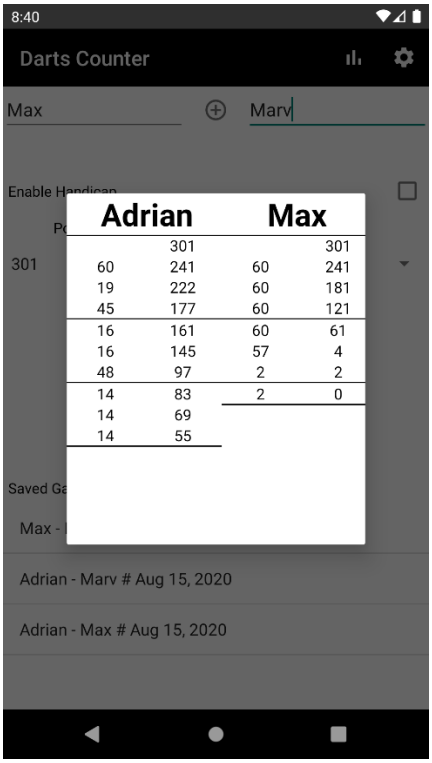


Fig 2: MatchHistoryDialogFragment where the right player has won

2.2 ScoreboardActivity

The ScoreboardActivity is where the actual games take place. The screen is again split up vertically into two parts. The bottom fragment is for gesture inputs of the current thrown score. We have initialized a straightforward 4x5 grid to put the numbers 1-20. Below you can also find the “miss” (0 points) or “bull” button (bullseye: 25 points). Because of the fact that you can score double or triple points when you manage to throw into the corresponding fields of the dartboard, we would need many more buttons or some kind of EditText to input the scores. This is why we came up with the idea to use “up swipes” for x2 the score and “down swipe” for x3 the score. This means if you score a triple 20, you would just swipe down on the “20”.

The top part is used to display the current score of the player, who's turn it is currently (RadioButton) as well as a throw history of the last turn in the middle and finishing suggestions for each player individually, based on the current points, darts left and obviously the game mode. There is also an "undo" button in the TaskBar to undo mistakes made while putting in the scores. The input in the bottom half and the output of the top half are basically connected through the game logic. Every time a button is pressed, the state of the game gets updated and will change accordingly.

The live finishing suggestion (below a player's score) is a clickable TextView and will open up a DartboardDialogFragment (Section 2.2.1) of the current finish to highlight the needed fields to finish (or setup the finish).

After one player has won the game a GameFinishedFragment shows up, which allows the user to externally share the game or save it into a file so it will show up on the MainActivity. You can also quit to the main menu from there.

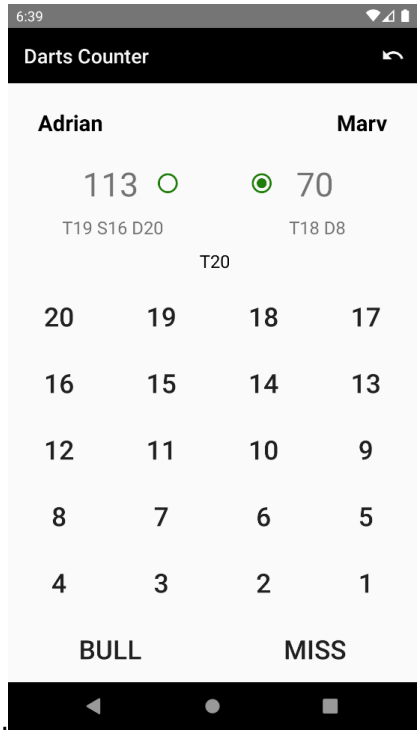


Fig 3: ScoreboardActivity with finish suggestions and throw history (only one throw currently)

2.2.1 Creating the button grid

The score-input grid comprises over 20 buttons, so adding them manually would have been a pain. We improvised and created the grid programmatically via a ConstraintLayout and arranged them next to and above each other with the constraints.

```
//create score button grid
ConstraintLayout.LayoutParams params;

for(int y = 0; y < 5; y++) {
    for(int x = 0; x < 4; x++) {
        Button b = buttons[y*4+x];
        params = new ConstraintLayout.LayoutParams(0,0);
        if(x == 0)
            params.startToStart = layout.getId();
        else if(x == 3)
            params.endToEnd = layout.getId();

        if(y == 0)
            params.topToTop = layout.getId();
        else if(y == 4)
            params.bottomToTop = buttons[20].getId();
            //params.bottomToBottom = layout.getId();

        if(x != 3)
            params.endToStart = buttons[y*4+(x+1)].getId();
        if(x != 0)
            params.startToEnd = buttons[y*4+(x-1)].getId();

        if(y != 0)
            params.topToBottom = buttons[(y-1)*4+x].getId();
        if(y != 4)
            params.bottomToTop = buttons[(y+1)*4+x].getId();
        b.setLayoutParams(params);
        layout.addView(b);
    }
}
```

Fig 4: Programmatic creation of the button grid

As you can see, we used two for-loops to create a 5x4 grid. There are special rules in place for the outer buttons, because they can not constrain to other buttons for every direction depending on the location. The inner buttons are always constrained to the buttons around them.

2.2.2 GestureListener

From the beginning we wanted the score-buttons to make use of swipes to register double and triple hits. But we quickly ran into the problem that there is no built-in way of registering a simple left/right or up/down swipe. So we created our own that calculates if there was a swipe, based on the coordinates of the first touch and lift of the finger.

```
public ScoreGestureListener(Context context, int threshold) {
    gestureDetector = new GestureDetector(context, new GestureListener());
    this.threshold = threshold;
}
```

(a) Constructor of our GestureListener with a threshold

```
b.setOnTouchListener(new ScoreGestureListener(this,
        Resources.getSystem().getDisplayMetrics().heightPixels/10) {
```

(b) Initialization of the GestureListener for every button

To accommodate for small variations in the clicks we introduced a threshold that depends on the height of the display. This way we avoided a strong hardcode. We obviously still had to determine a percentage of the screen by ourselves, but we feel that our current threshold will not lead to many mistakes and feels natural to use on every device.

```
public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
    float dy = e1.getY() - e2.getY();

    if(Math.abs(dy) > threshold) {
        if(dy > 0)
            onSwipeUp(view);
        else
            onSwipeDown(view);
        return true;
    }

    return false;
}
```

(c) Using the threshold to determine if the swipe was “strong” enough

2.2.3 DartboardDialogFragment

We have visualized the finishes on a dartboard, because it can be hard for beginners to find every spot they should hit, especially if we use the standard notation (e.g. T19 = “triple 19”). We used a predetermined graphic for the board and flashes. But since a flash can occur on all of the 20 segments, we had to find a way to rotate them along the middle point of the board to address every segment. The overlays will flash in order at the fields you need to score.

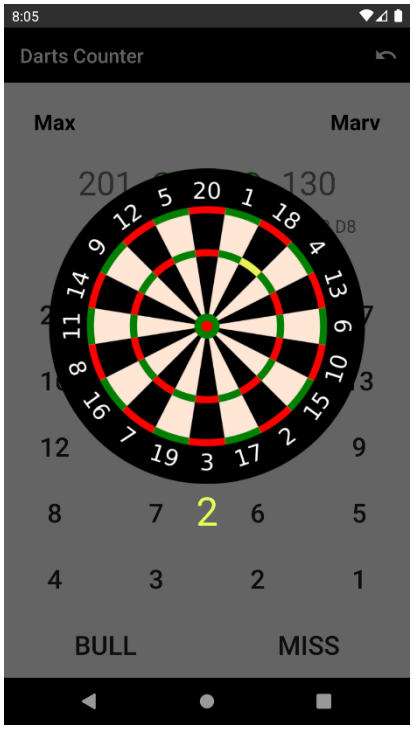
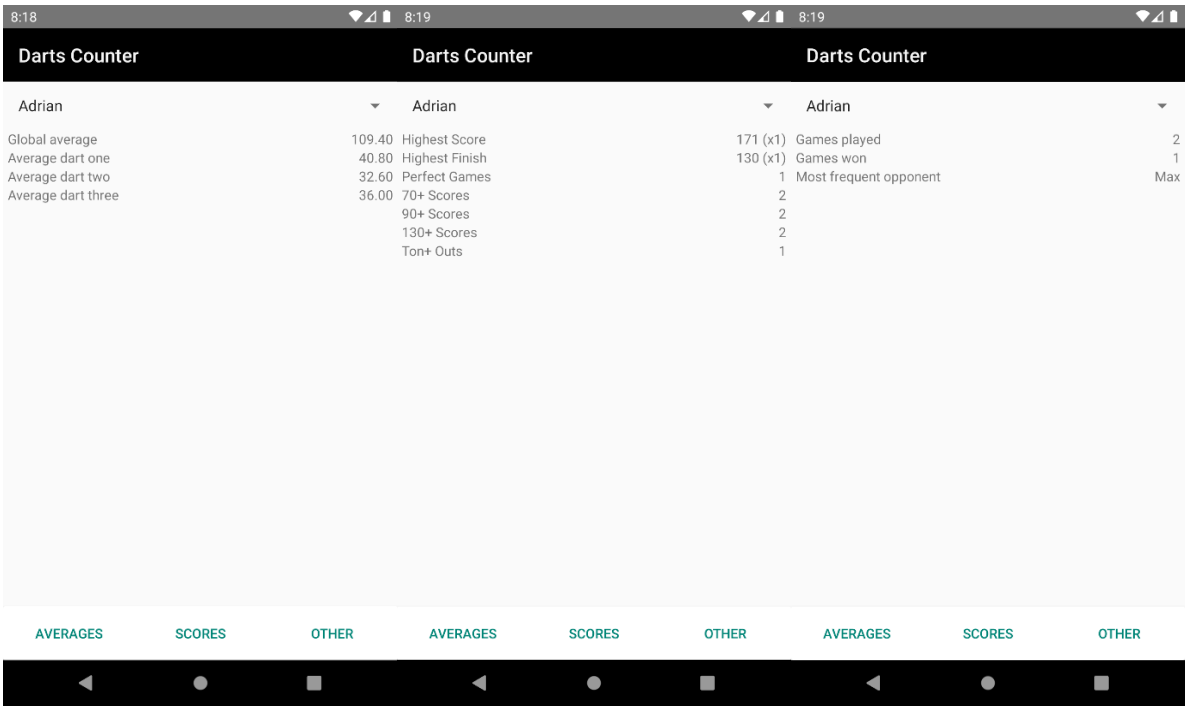


Fig 5: DartBoardDialogFragment which flashes at the triple 18

2.3 StatisticsActivity

We have implemented statistics for every player, which get updated after every game played. The activity is divided into three fragments. These are “averages”, “scores” and “other”, each of which holds interesting data like “highest finish”, “global average” or “most frequent opponent”. After a game is finished, we extract the needed information from the game automatically (no need to save the game on the aftergame popup) to update the database with new stats.



(a) AveragesFragment (b) ScoresFragment (c) OtherFragment

2.4 SettingsActivity

The SettingsActivity uses a SettingsFragment, which holds two settings. One of them is the aesthetical appearance of the top bar by choosing its color. The other is the language setting between English and German. A restart is needed if you switch between languages.

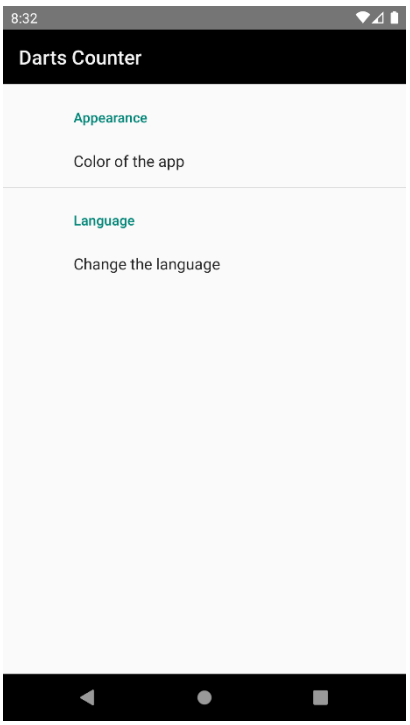


Fig 6: SettingsFragment

2.5 Sharing Games

As said before, we have the opportunity to externally share a played game in the GameFinishedFragment. We utilize the Android Sharesheet to send a string representation of the match to the receiving app. The string of the match gets written actively while the game is played, so you don’t have to wait for it to be generated. We put every important information about the environment of the game in the first half of the string (player names, game mode, etc.) and separate it by a dot.

The second half is used for the actual game. We decided to only save “scored points” and not the points history because it simplifies the string and it will give every information we need. On receive, the string gets separated into substrings to recreate the game that was played. After we have extracted everything, the game gets saved to show up in the MainActivity.

2.6 Game Logic

While in the ScoreboardActivity, the app waits for the input via the GestureListener and loops its logic until the game is finished. We use global variables to keep track on who's turn it is and how many darts were thrown. We also save the points history (points remaining) and the throw history (points scored) as an ArrayList to be able to handle the "undo" action.

In darts you are able to score points in different ways, for example you can score "18" with "single 18", "double 9" and "triple 6". Because it is necessary to differentiate between that sometimes (think of finishes) we established a standard throughout our app which follows this rule: Points scored are saved as a three-digit integer, where the first digit indicates the multiplier and the last two digit indicate the field that was hit (e.g.: "312" is "triple 12", "100" is "miss", "125" is "single bull").

There is also a slightly simplified diagram of one cycle through the game logic attached to this file (or in the folder on GitLab), which shows the essential steps of what happens for one player, from when the app awaits an input to after the cycle is completed.

2.7 Database

The database keeps track of matches played by a player and various statistics throughout the games. It has two tables, one for the matches and one for the players (see the ER-diagram), where the statistics are embedded into the match. The players are also embedded because they are needed when saving a game which has no connection to the database.

For every player a unique id is stored and additionally a name. A match also has a unique id, a boolean value of who won the game (true means player one won, false when player two won) and an instance of the Statistics class where the stats are saved.

We implemented the database with Android Room so we have DAOs for the players, which includes operations for inserting/updating/deleting and also queries to get either all players or a specific player by their id, and for the matches, which supports the insert/update/delete operations and a query to get a list of all matches where a specific player competed.

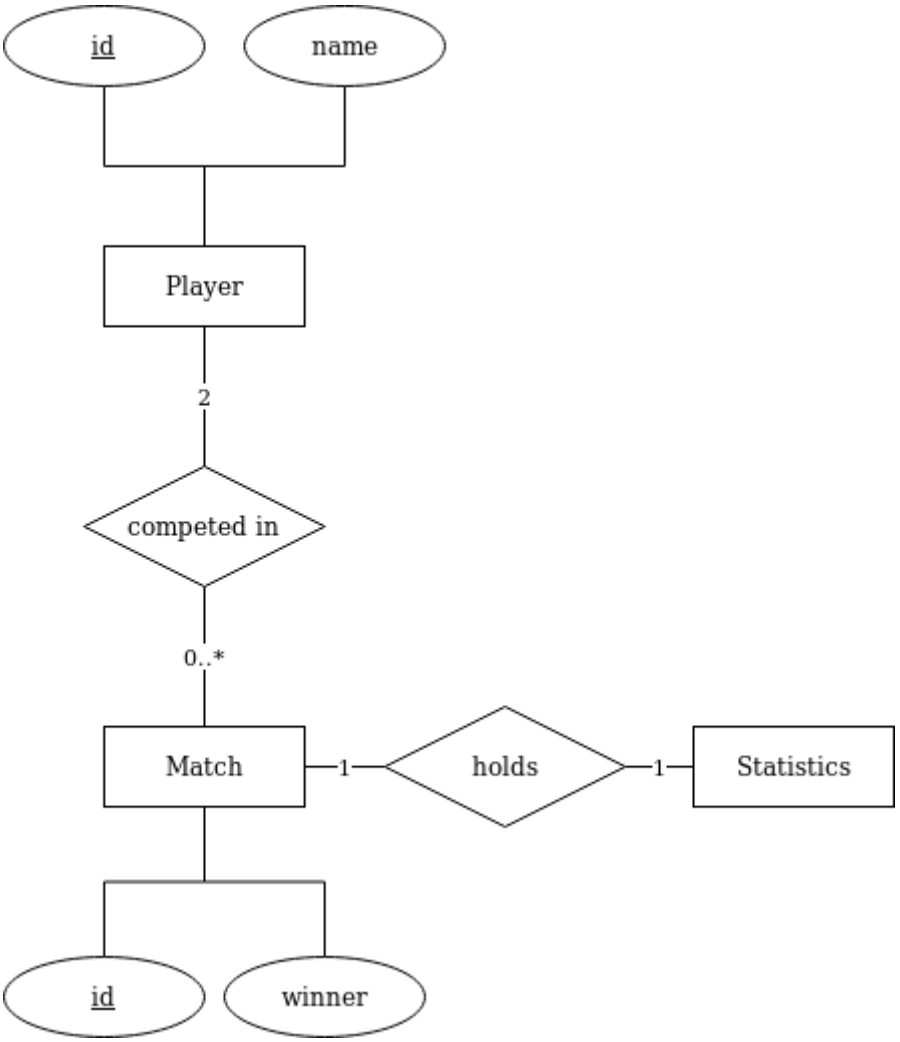


Fig 7: ER-Diagram of the database

2.8 CheckoutMap

The CheckoutMap class stores the standard checkouts for the wanted game modes as two dimensional static int arrays. This gets accessed from the ScoreboardActivity to update the suggested finishes for the current score.

2.9 Miscellaneous

We structured our project so that a package contains classes that serve similar purposes. So, we created packages for UI-elements, activities and one for all database related classes.

Graphically, apart from the standard icons, we have designed some elements ourselves, like the dartboard or the flashing overlays on finishing suggestions.

3 Encountered Problems

3.1 Gesture Input

When we started our implementation of the swiping inputs, we started with left and right swipes. This was not feasible, because of the layout for our input scores (screen limitation for swiping left on numbers on the left and vice-versa). We have then switched to up and down swipes. This is working because at the top there is obvious space and at the bottom, we only have the buttons “miss” (no down or up-swipe needed) and “bull” (no triple possible, so only up-swipe).

3.2 Future Statistics

One obvious problem with our statistics is the difficult task to implement new statistics. Because we don’t save every game played for space reasons, we can’t extract the needed information for the new statistics. The easiest solution is to have every important statistic implemented from the start. It still hinders the ambition if we really want to flesh out the statistics part, because we probably need to either ignore the older games, suggest the player a split of statistics or just start the statistics from zero after every update (not ideal).

3.3 Game Logic

While the logic of standard darts is easy to understand, the turn-based throws, busting, undo-button and finishing nuances (for different modes) complicated the game logic, especially because you need to start somewhere and you probably will not find a perfect solution from the start. The ArrayLists of scored points and points left were a big help but required some serious index work, especially while implementing the undo function and resetting points after you bust.

4 Further development

We do not have every nice to have feature implemented that we wanted, but as the core functionality is the most important, we are happy with the results of this app and as both of us play darts as a hobby, we can comfortably say that the app is useful. Maybe we will flesh out the UI and make some things flow more naturally so the time spent navigating the app while playing will be minimized even more. It is also quite possible that there are still some things to improve in the finishing suggestions which will probably only show after you have played for a while (for example personal preferences for finishes).