Dart counting app

by Adrian Kröger and Marvin Kühn

General concepts of darts and our ideas

- Starting points of “301”, “501” etc. for both players
- Trying to reduce points to zero, you have three darts per round
- Outer field counts x2 and inner field counts x3
- You need to reduce the points exactly to zero (with a double for standard game modes) or else you will be reset to the points you had before the round

- Counting down is hard without help

→ automating the counting

- Thinking of finishes is hard without help

→ automating the finishing suggestions

- Stats are tracked only through payed apps most of the time

→ automating the tracking of stats

Game Logic

- global variables to keep track on who’s turn it is
- global variables to keep track on the players points and throw history

Game Logic

- global variables to keep track on who's turn it is
- global variables to keep track on the players points and throw history

int turnCounter
- 0, if start of one players turn
- 1, if player has thrown one dart
- 2, if player has thrown two darts
- 3, if player has thrown three darts

int turn
- 1, if it is player1's turn
- 2, if it is player2's turnh

increment the turnCounter after every legal throw

if the turnCounter is 3 --> switch player via "turn" and reset to 0

Game Logic

- global variables to keep track on who's turn it is
- global variables to keep track on the players points and throw history

```
private ArrayList<Integer> pts_history1 = new ArrayList<>();
```

271	250	190	-1	130	70	38	-1	32	38	-1	16	0

```
private ArrayList<Integer> throw_history1 = new ArrayList<>();
```

310	307	320	-1	320	320	216	-1	106	318	-1	211	208

- the true game history can be deduced via throw history
  but not via the pts_history

Buttons

- We had to add over 20 buttons for the score-input grid

→ Adding them manually would have been exhausting

- We programmatically aligned the buttons via a ConstraintLayout
- The 5x4 grid is reflected in the two for-loops
- Extra rules are in place for the buttons on the outer edge

(Next slide will show how the button grid turns out)

```
//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);
    }
}
```

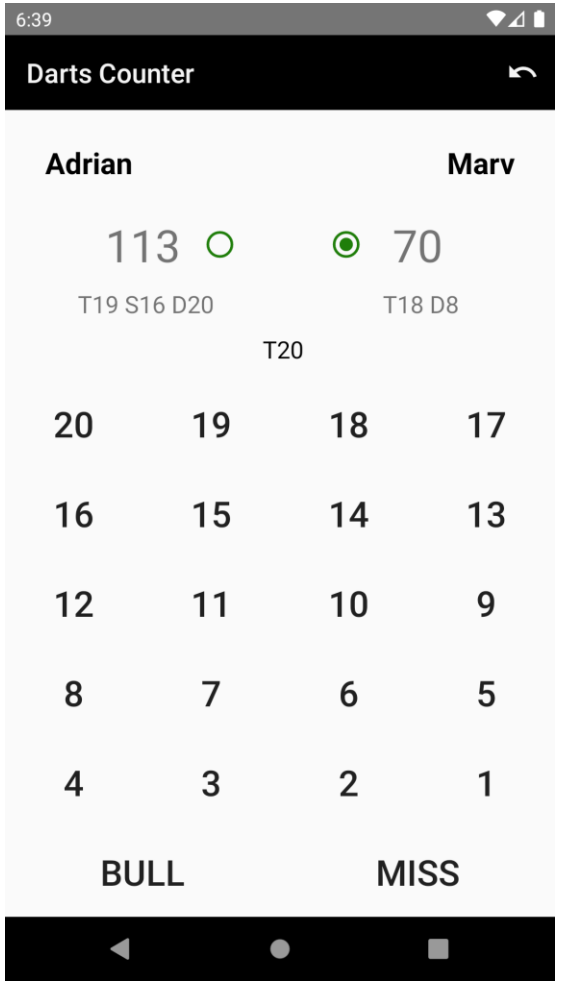
Swiping mechanic

- using swipes to speed up the input process
- we started with left and right swipes

→ problems with dimensions of the layout

- Switching to up and down swipes

→ up is “double” and down is “triple”



Swiping mechanic

- Every button initializes a GestureListener with a threshold

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

- That threshold is based on the displays height

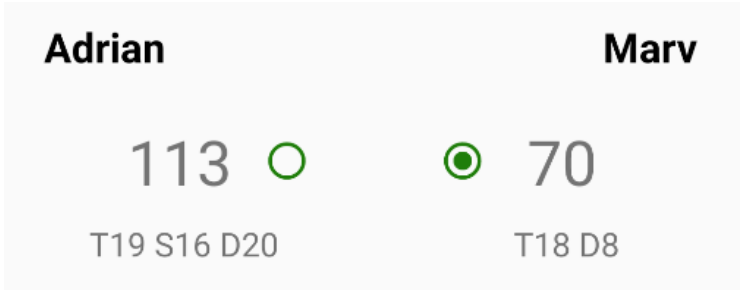
```
b.setOnTouchListener(new ScoreGestureListener(this,
        Resources.getSystem().getDisplayMetrics().heightPixels/10) {
```

- We use that threshold to determine if a swipe should be registered as such (so that a small accidental movement will not register as a swipe)

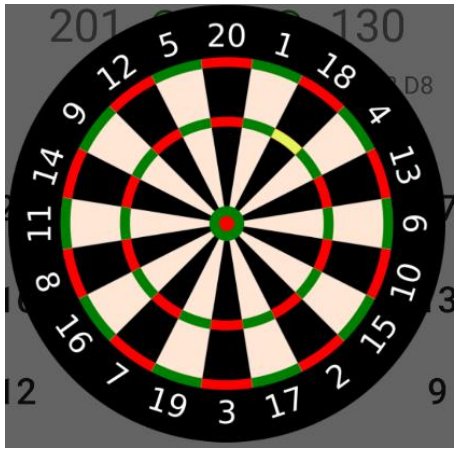
```
public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
    float dy = e1.getY() - e2.getY();

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

    return false;
}
```

Finishing suggestions

- When the points of one player get into a range where a finish is possible directly OR it is possible to prepare a finish, we suggest the player the throws he should make
- Short indication in standard notation of darts below players points (e.g. “T19 S16 D20”)
- Or longer indication as a clickable TextView to show where on the dartboard you should aim to hit



Short indication for finishes below players points



Longer indication with graphics

Rotation of finishes to display

- We used a predetermined graphic to flash, but changed the orientation based on the scores

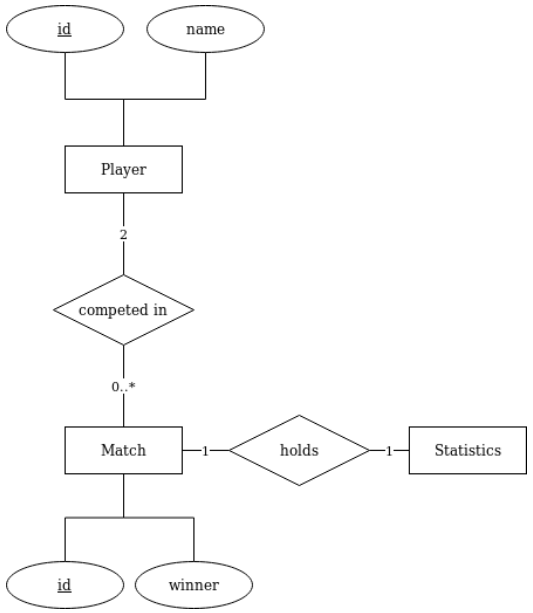
```
/** defines rotations of 18° needed to reach field with score n+1*/
private final int[] rotations = {1, 8, 10, 3, -1, 5, -8, -6, -3, 6, -5, -2, 4, -4, 7, -7, 9, 2, -9, 0};
```

```
if(args != null && (scores = args.getIntegerArrayList("scores")) != null) {
    final ImageView[] blinks = new ImageView[scores.size()];
    final AlphaAnimation[] anims = new AlphaAnimation[scores.size()];
    for(int i = 0; i < scores.size(); i++) {
        if(scores.get(i)%100 != 25)
            blinks[i] = getOverlay(resources[scores.get(i)/100 - 1], 18*rotations[scores.get(i)%100 - 1]);
        else
            blinks[i] = getOverlay(resources[scores.get(i)/100 + 2], 0);
        layout.addView(blinks[i], -1, params);
        anims[i] = (AlphaAnimation) AnimationUtils.loadAnimation(getContext(), R.anim.blink);
    }
```

Database & statistics implementation

- Every match and every player has a unique ID and is saved to the database
- Statistics are attached to a match which are attached to the players
- We used Android Room, so we have DAOs for the players and matches with standard operations (insert/update/delete/etc.) and queries
- Every finished game will update the players statistics in the database automatically

Example of some statistics



ER-Diagram of database

Sharing of games/string representation

- We used the standard Android Sharesheet to be able to share a game which has been finished
- The app can receive these kind of strings and will extract the needed data
- After extracting the data, it will put the recreated match into our match history
- The string representation of the match will be written in real time while the match is played and will generally look like this:

Sharing of games/string representation

For extracting data: On receive, split array at "."
Then split on ";" for both sides
For every turn split on ","