

BOP-IT!



תיכון עודד

ע"ש עודד ראור
מחנך אותך בדיוק שלך

בית ספר: תיכון עודד קדימה צור

תלמיד מגיש: הראל גרינברג

ת"ז: 214561359

שם המנחה: ריקי יפה

תאריך הגשה: 16.5.22

תוכן עניינים

2.....	מבוא
3.....	תיאור תחום הידע
4-18.....	מבנה
19-36.....	מימוש הפרויקט
37.....	מדריך למשתמש
38.....	רפלקציה
39-40.....	ביבליוגרפיה

מבוא

רקע לפרויקט

שם פרויקט: BOP-IT!

תיאור הפרויקט: BOP-IT נעשה בהשראת המשחק המכני BOP IT שבו יש מוזיקה קצבית ויש קול שנותן הוראות שצריך לבצע. לכל הוראה יש חלק שונה במשחק לדוגמה לפעולה PULL IT יש חלק כחול שאותו צריך למשוך כדי להמשיך הלאה. ההבדל בין המשחק לפרויקט הוא שהמשחק עושה שימוש בסנסורים של הטלפון כמו חיישן התנועה והג'ירו.

קהל יעד: אנשים שמשחקים משחקי טלפון, יכול להיות ילדים ומבוגרים כאחד.
הסיבות לבחירת הנושא: יום לפני הגשת נושא העבודה יצא לי לשחק במשחק BOP IT וזה נתן לי השראה לבנות את האפליקציה.

תהליך מחקר

לפני שהתחלתי לבנות את הפרויקט הכנתי אפליקציה ראשונית שמטרתה הייתה לבדוק אם זה בכלל אפשרי להשתמש בכל כך הרבה חיישנים ביחד. לאחר הרבה ניסוי וטעייה, הורדת פעולות והחלפת פעולות מסוימות הגעתי לרמת דיוק משביעת רצון עבורי שעלייה לבסוף התבססתי להכנת הפרויקט הסופי.

אתגרים מרכזיים

בפרויקט התמודדתי עם קשיים ואתגרים רבים. אחד מהקשיים הכי גדולים איתם התמודדתי היה אילו פעולות יכנסו לפרויקט הסופי. תחילה רציתי שהפעולות יהיו Slice, Flip, Swipe, Shake, Press אבל לאחר התנסות גיליתי שפעולות הניעור והחיתוך מתנגשות אחת בשנייה, ניסיתי לסדר ככה שהן לא יעשו זאת אך לבסוף הורדתי את פעולת הניעור. לאחר מכן הוספתי פעולה הנקראת Blow (לנשוף) אך גם איתה היו קשיים והיא לבסוף הוחלפה בפעולה Cover (לחסות). לאחר שהחלפתי טלפון נייד גיליתי שהפעולה לכסות לא עובדת באותה צורה כמו שהיא עבדה בטלפון הקודם ולכן לבסוף גם היא נמחקה ובמקומה הוספתי Swipe(Direction) כלומר הפעולה החלקה אך לכל ארבעת הכיוונים. לבסוף הפעולות שאיתן המשכתי לפרויקט הסופי היו Press, Swipe(Direction), Flip, Slice כלומר 7 פעולות.

תיאור תחום הידע

ייצוג מידע

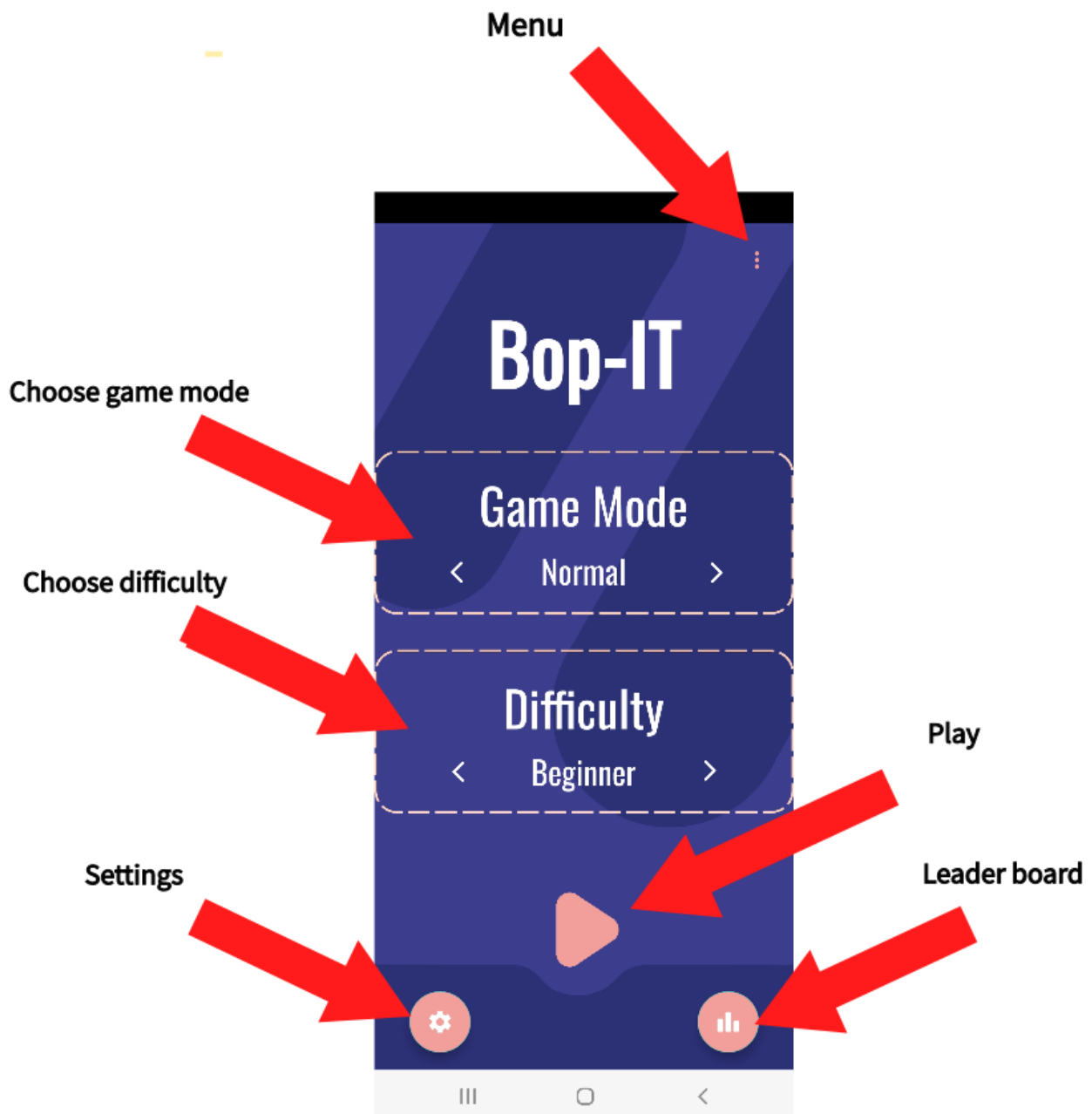
לוח המשחק מורכב מכמה חלקים, אזור בו השחקן יכול להחליק (Swipe), כפתור בו השחקן ממש את הפעולה לחיצה (Press), טקסט שבו מוצג הזמן, טקסט שבו מוצג התוצאה, טקסט שבו מוצגת הפעולה שנעשתה ושני תיבות טקסט אשר מחזירות פידבק חיובי.

ייצוג מבנה הנתונים

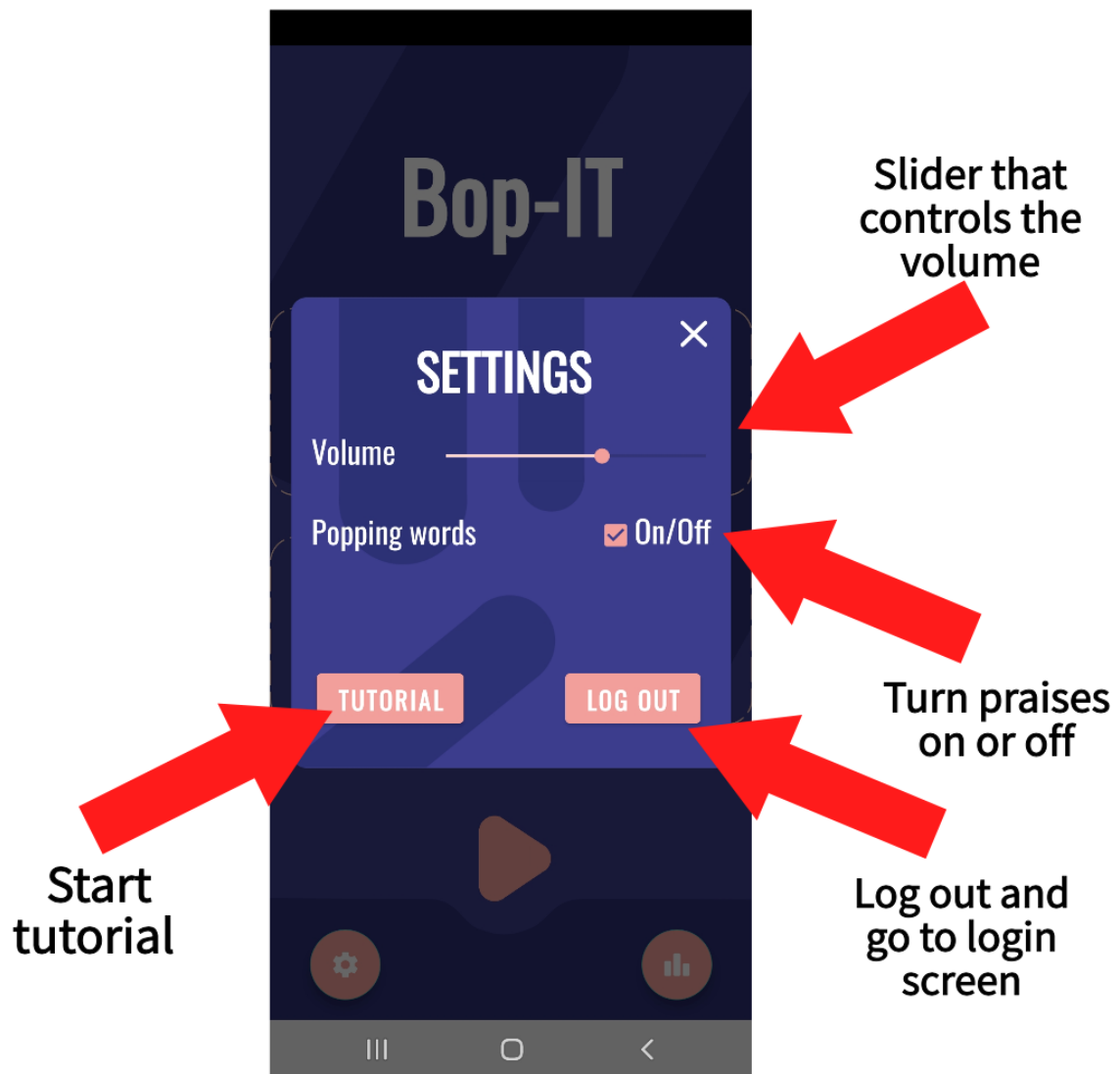
מבנה הנתונים שבו השתמשתי הוא Firestore של Firebase למבנה הנתונים יש 6 תכונות אימייל, שם משתמש וניקוד הכי גבוה עבור ארבעת סוגי במשחק. בהפעלת המשחק נפתח הדף הראשי ונבדק האם המשתמש נרשם בעבר במידה וכן הוא נשאר בדף הראשי ובמידה ולא הוא מופנה לדף ההרשמה. בעת ההרשמה מוכנס לבסיס נתונים השם אשר בחר לעצמו השחקן, האימייל שלו ומתאפסים כל הניקודים שלו. במסך הבית נשלפים מן הבסיס נתונים 4 רשימות של ניקוד לפי סוג משחק ו4 רשימות של שמות אשר מתאימות לכל אחת מהרשימות ניקוד. כדי לסמן איזה שחקן מסתכל כרגע על הלוח תוצאות נעשית בדיקה האם האימייל של האדם הרשום כרגע שווה לאחד שמסתכלים עליו כרגע (עוברים על כל השמות), במידה והם שווים מגדירים את הרקע להיות בצבע שונה מן השאר כדי להבליט לשחקן. בעת סיום משחק נעשית בדיקה אם השחקן עשה שיא חדש, במידה וכן מוכנס לבסיס נתונים השיא החדש.

מבנה

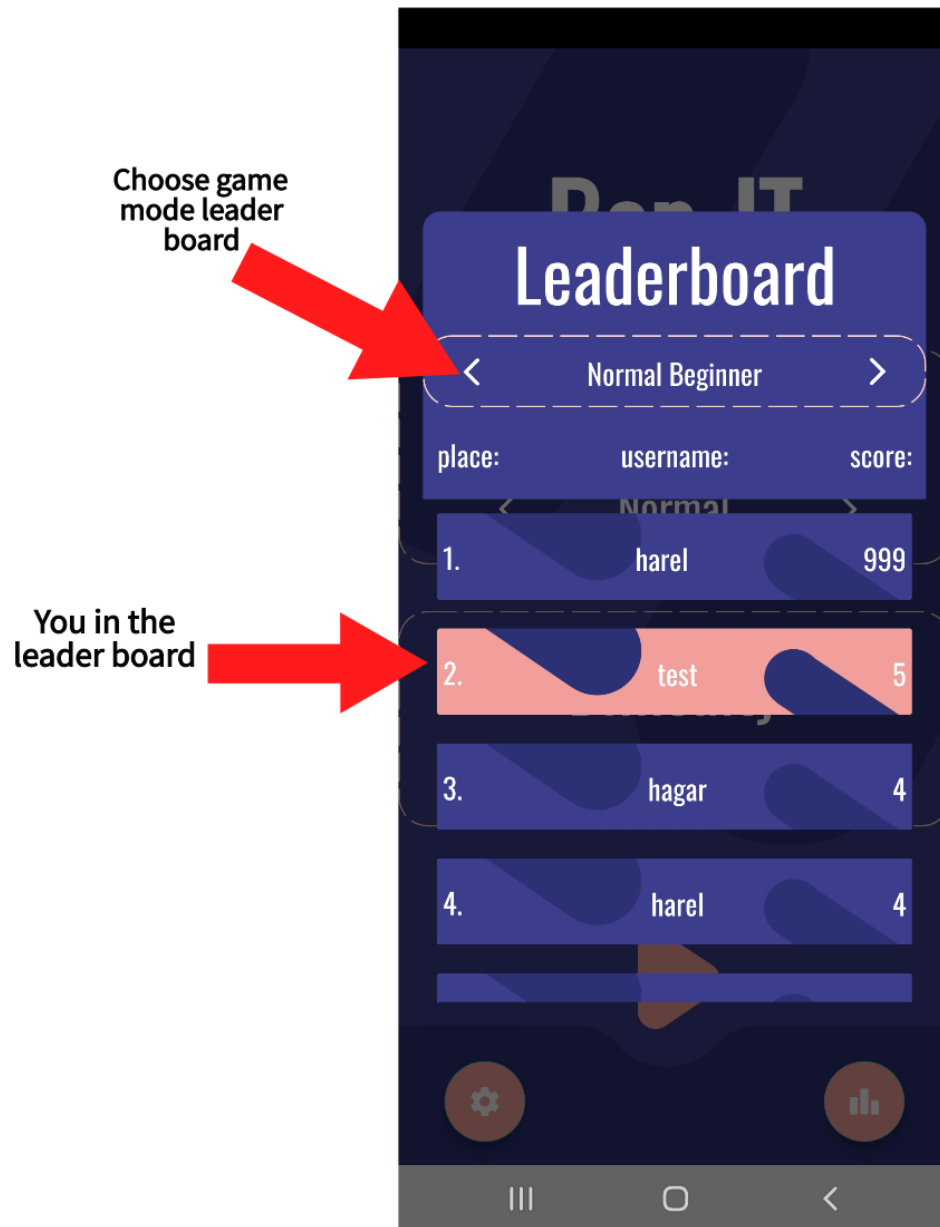
מסך בית



דיאלוג הגדרות



דיאלוג לוח תוצאות

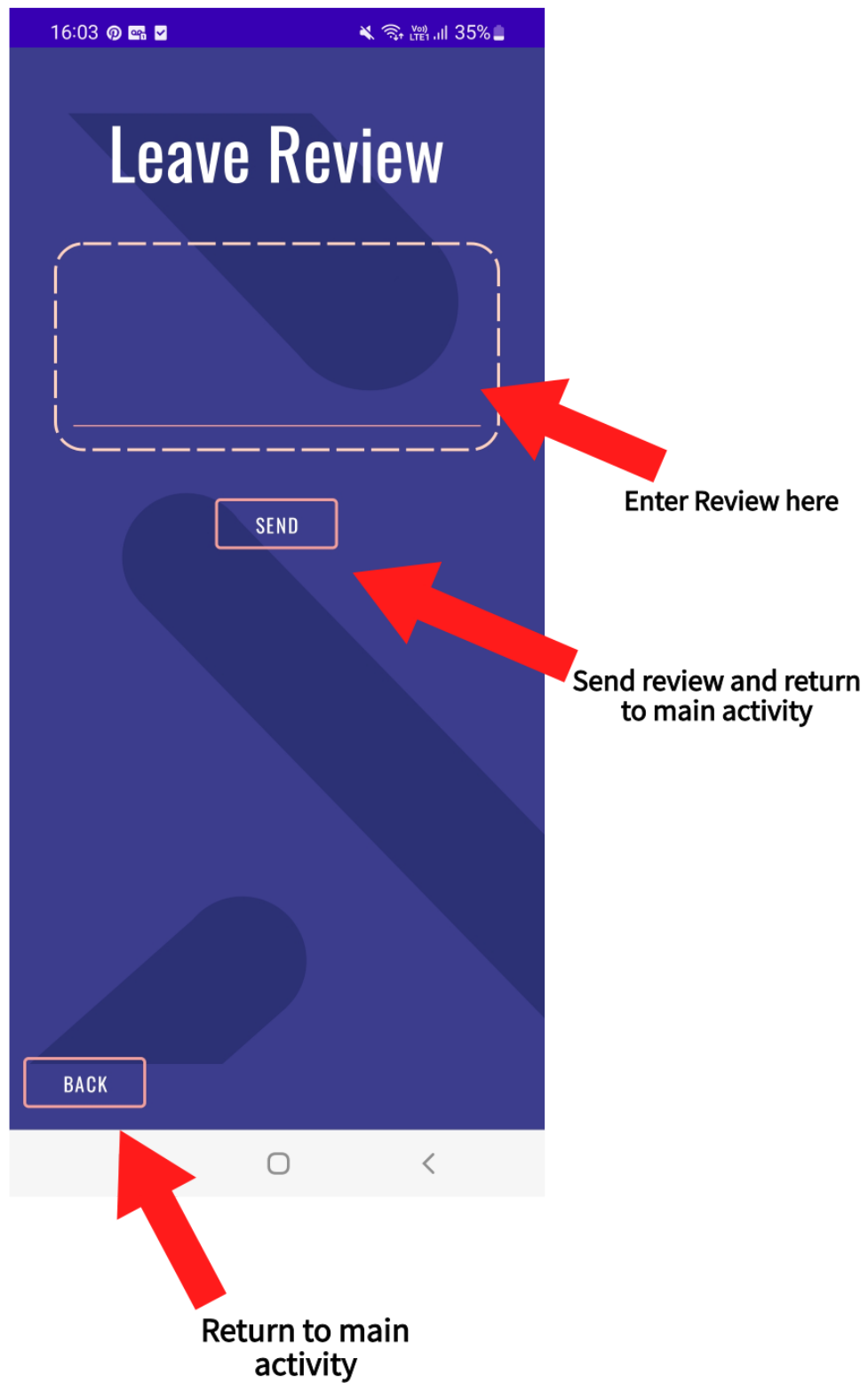


תפריט

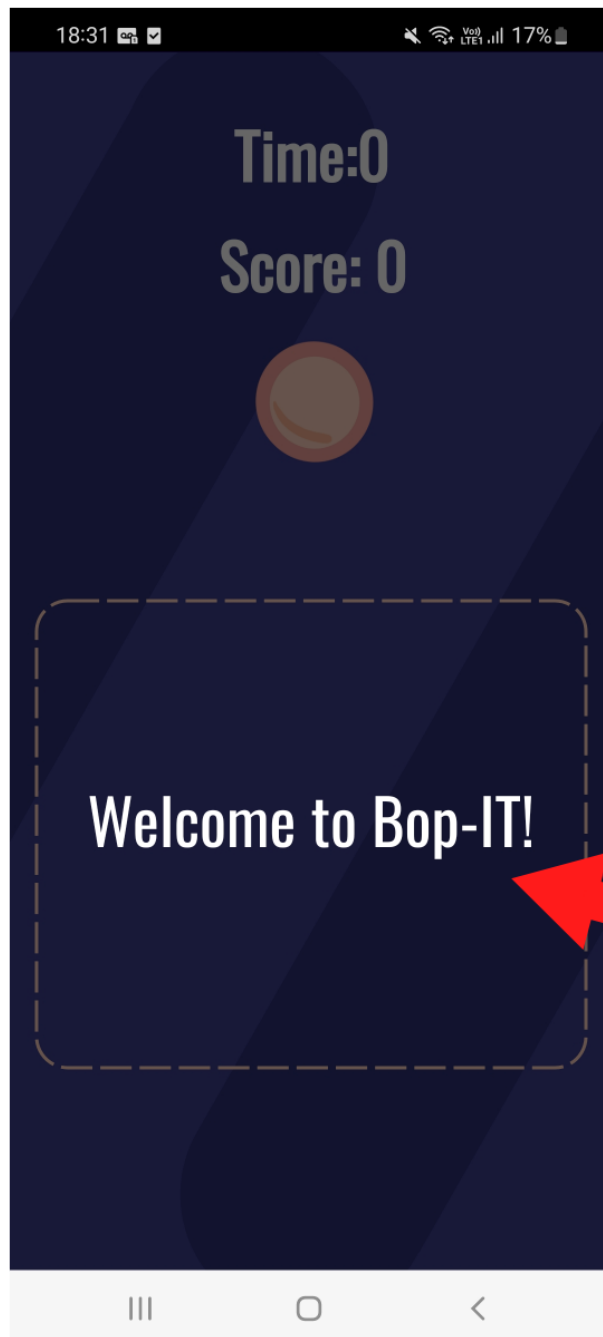


menu with 2 options, leave game and send feedback

מסך השארת תגובה

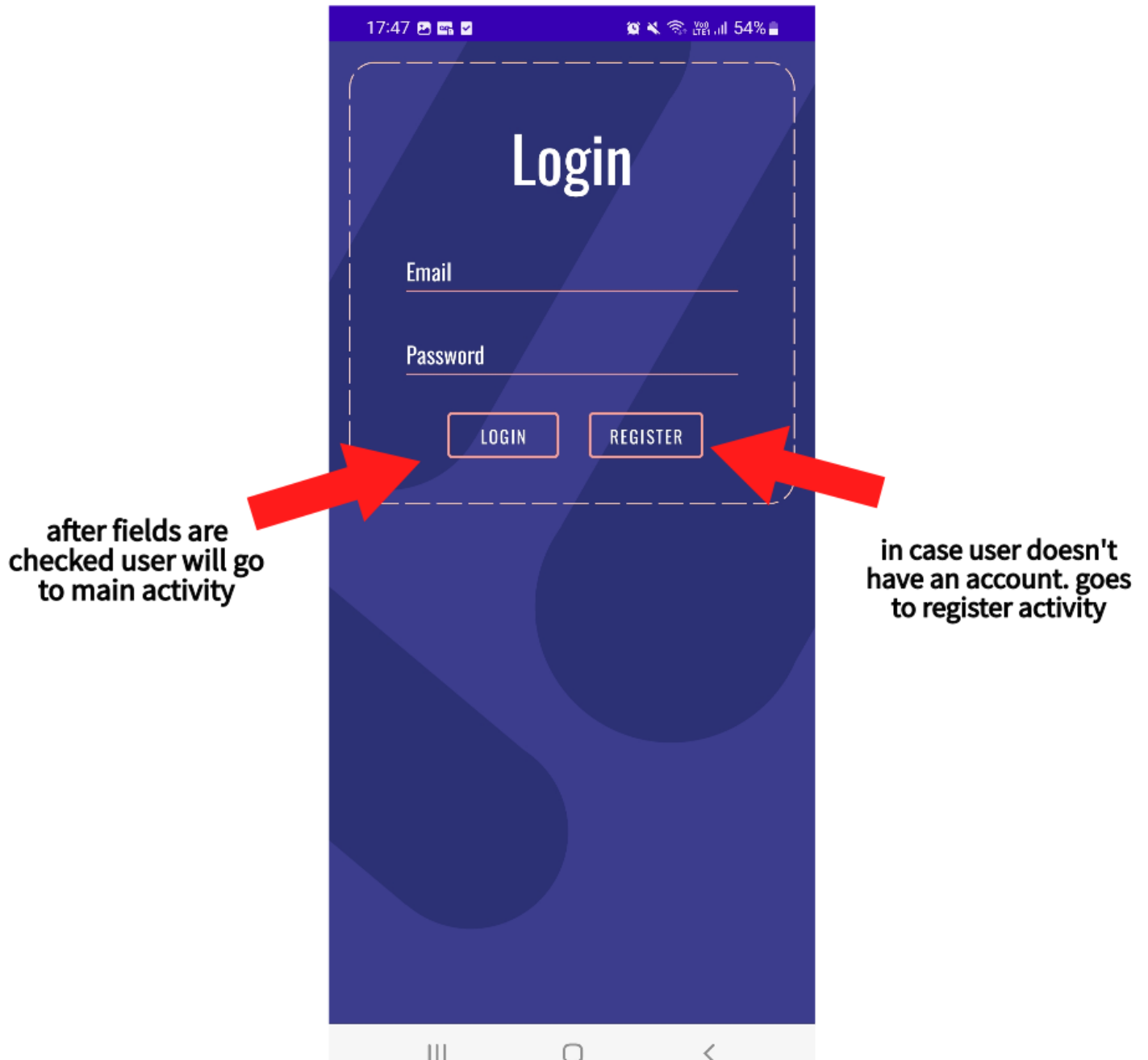


מסך הדרכה

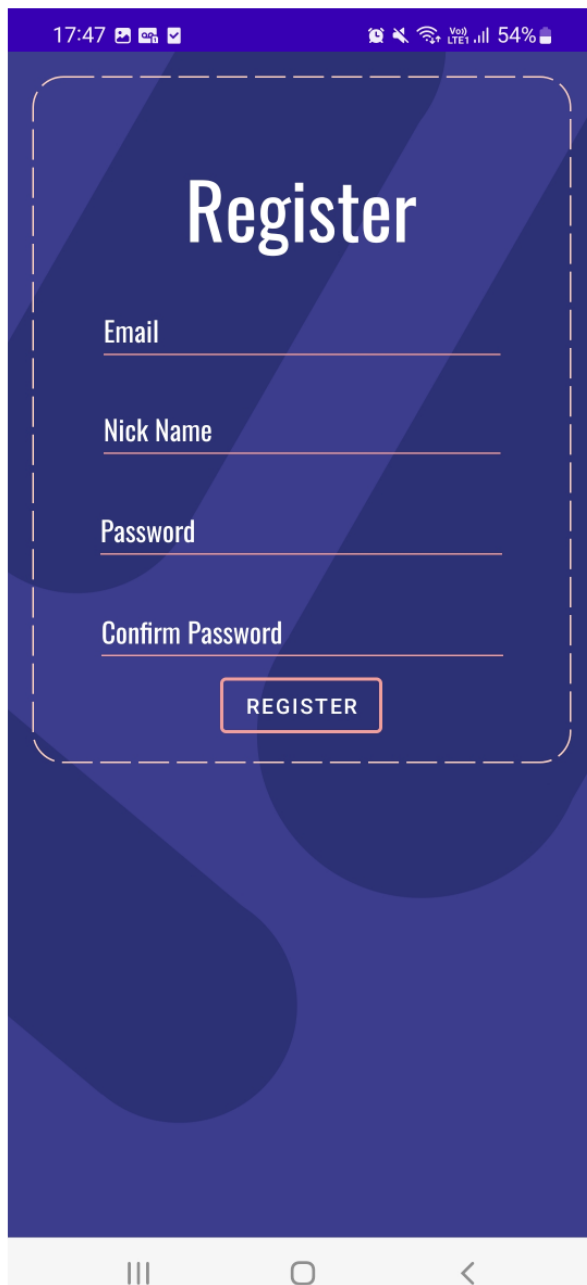


Text that guides
the player and
changes each tap

מסך התחברות



מסך הרשמה

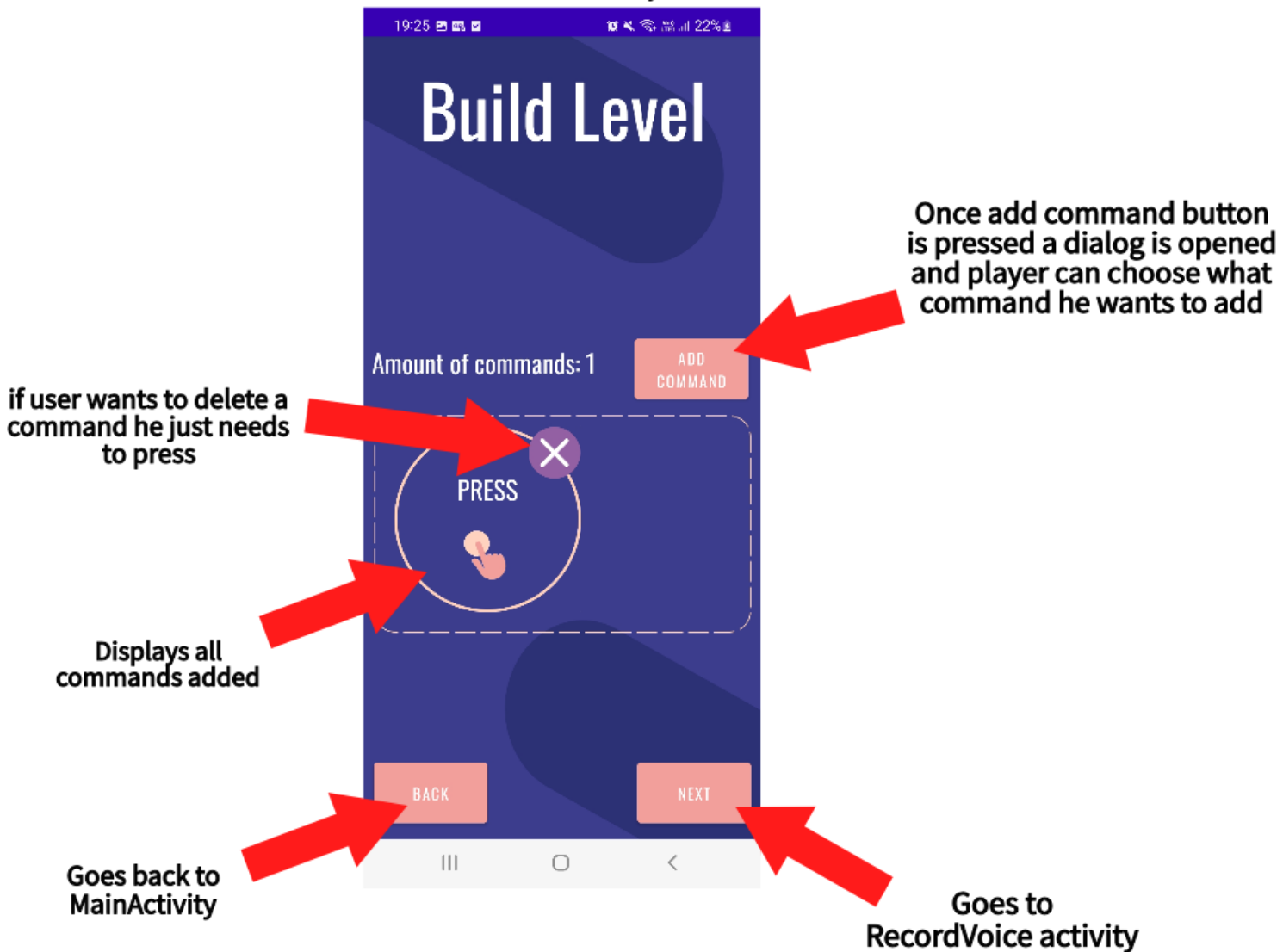


A mobile app mockup of a registration screen. The screen has a dark blue background with a lighter blue abstract shape. At the top, a status bar shows the time 17:47, signal strength, and 54% battery. The main content is enclosed in a dashed white border. It features the title 'Register' in large white text, followed by four input fields labeled 'Email', 'Nick Name', 'Password', and 'Confirm Password' in white text. Below the fields is a white 'REGISTER' button with a red border. At the bottom, there is a white navigation bar with three icons: a hamburger menu, a home circle, and a back arrow.

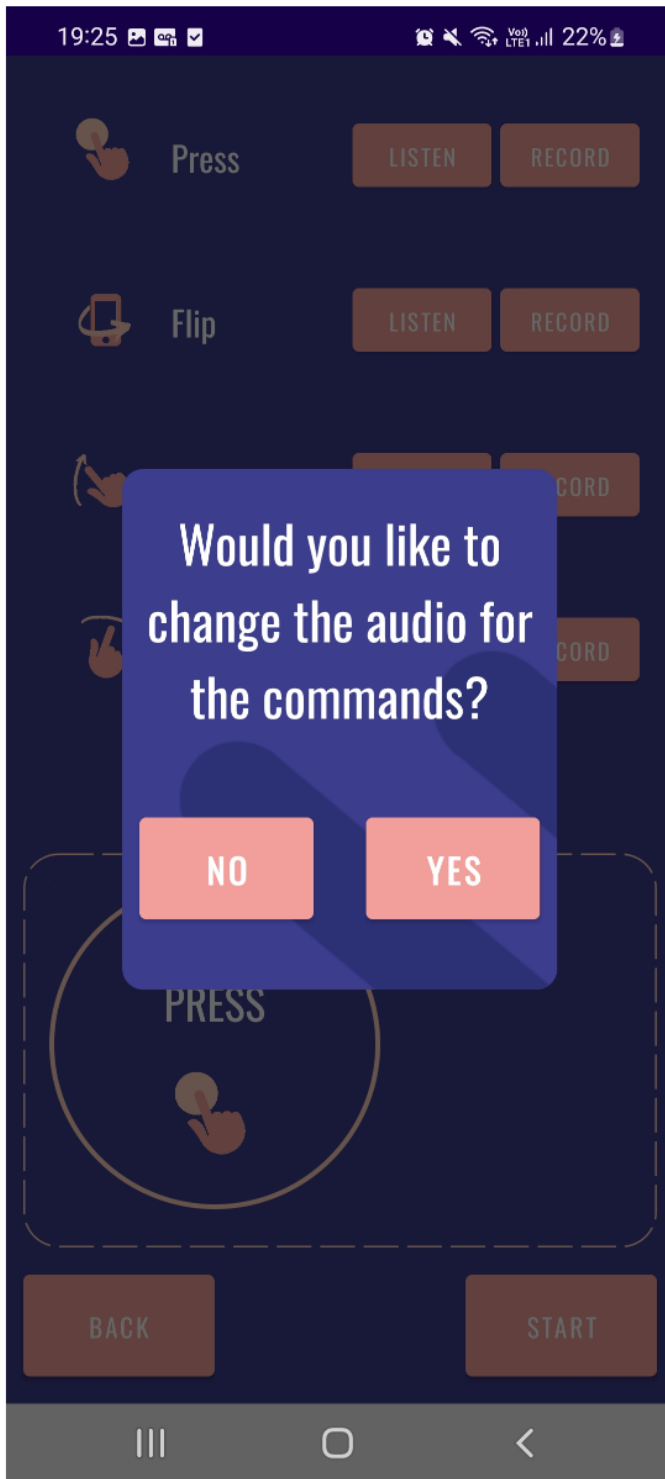
Each field is checked and confirmed to be valid before registering a new user. On register user is sent to tutorial.

מסך בניית שלב

If the game mode that was selected was "Build Level" user is sent to BuildLevel activity



דיאלוג הקלטה

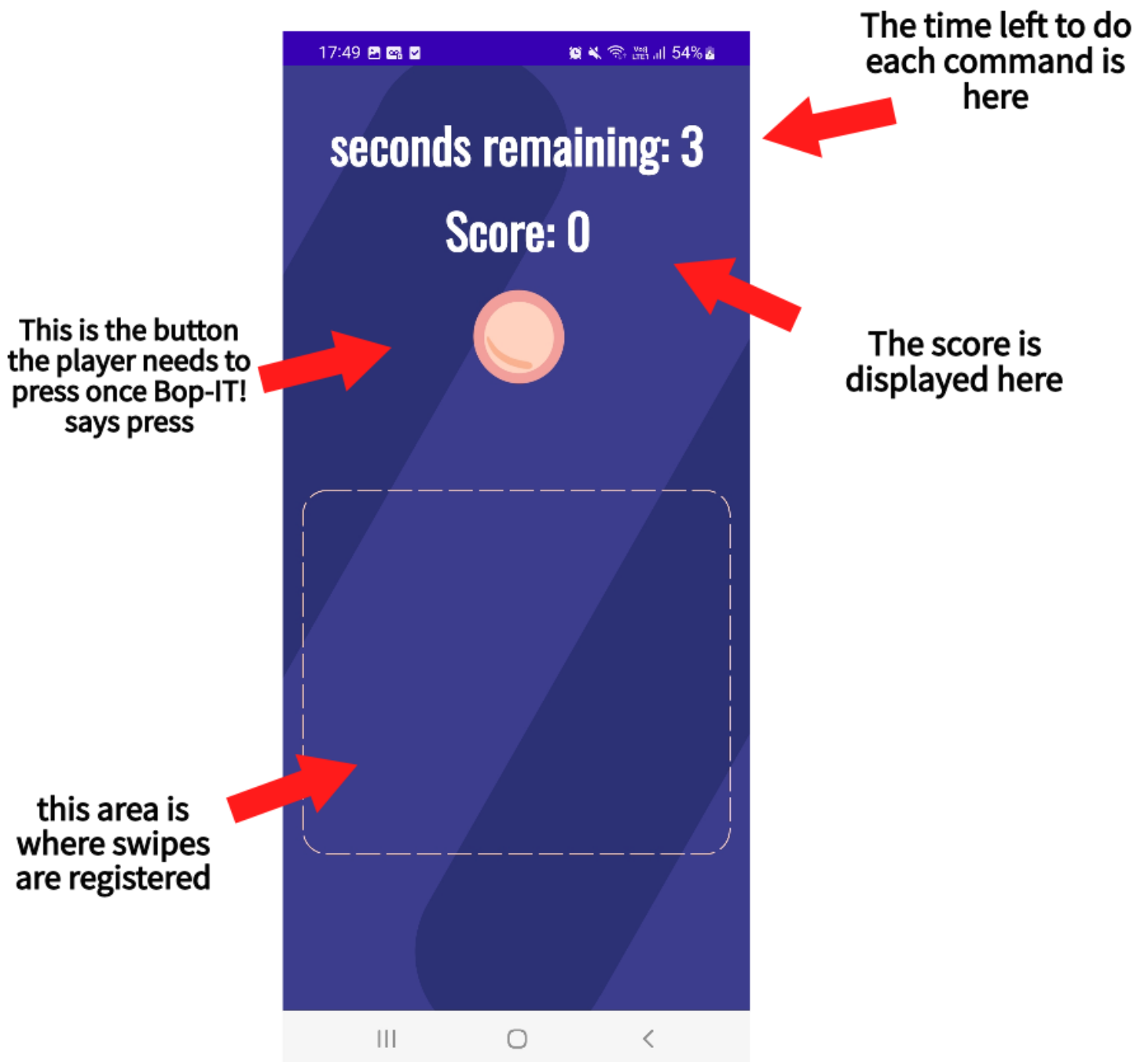


If player presses yes, he is directed to the the RecordVoice activity where he can change the audio for each command. If the player presses no he is directed to the CustomGame activity to play the game he created

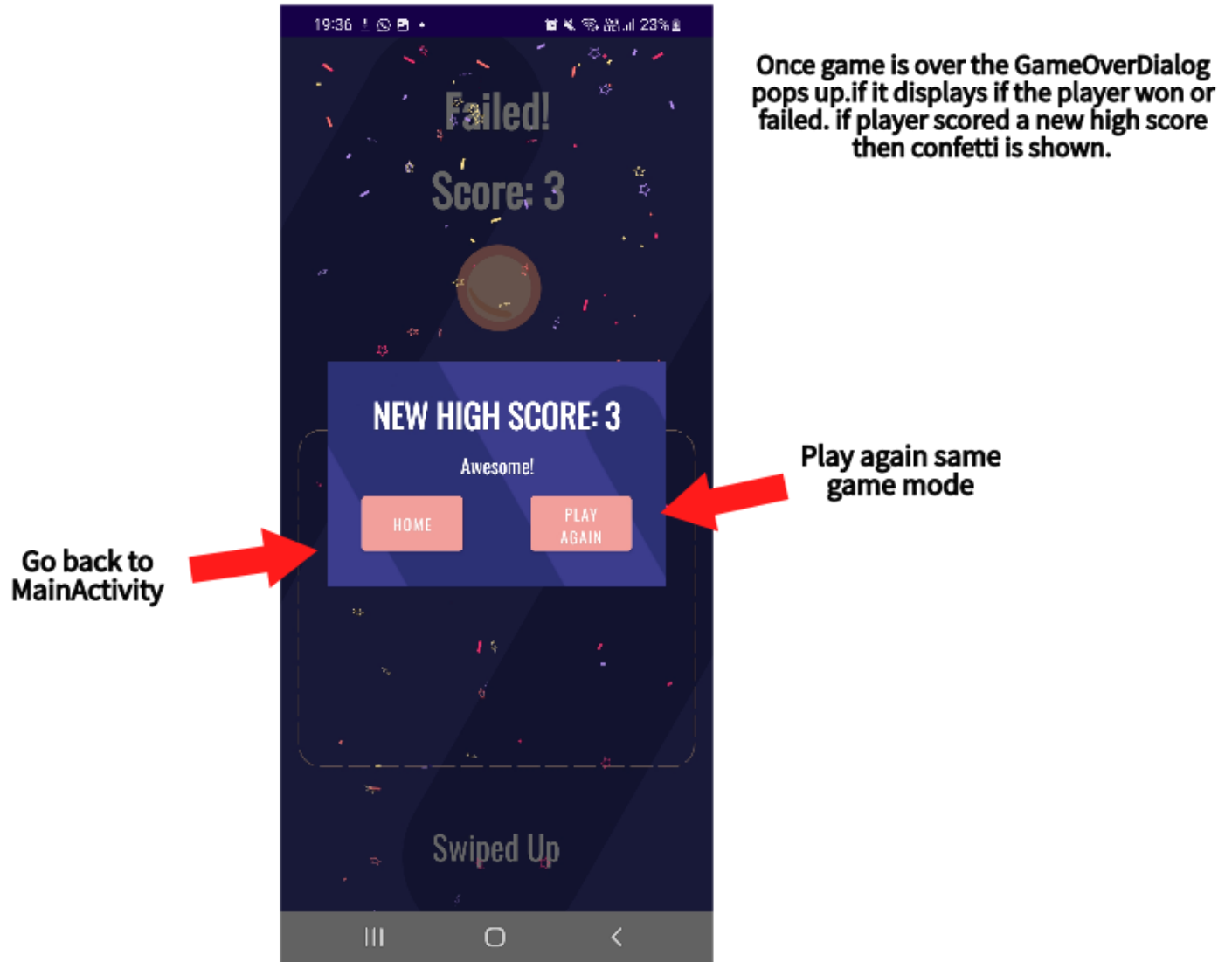
מסך הקלטה



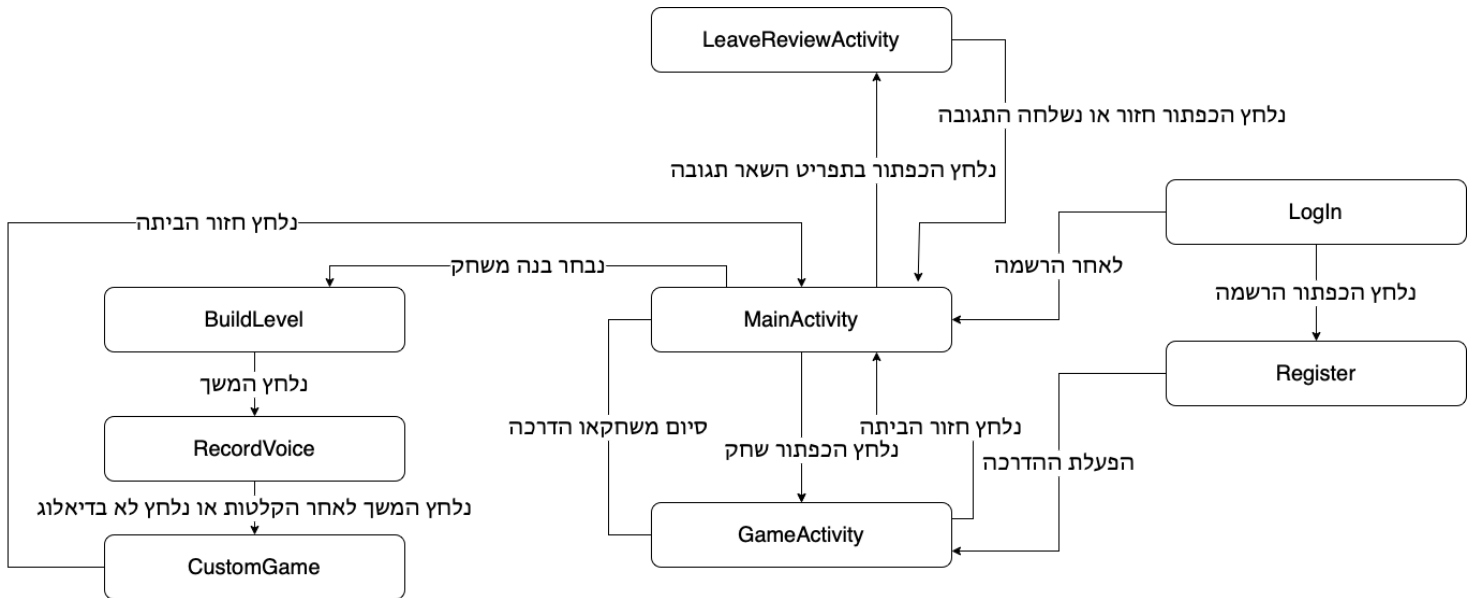
מסך משחק



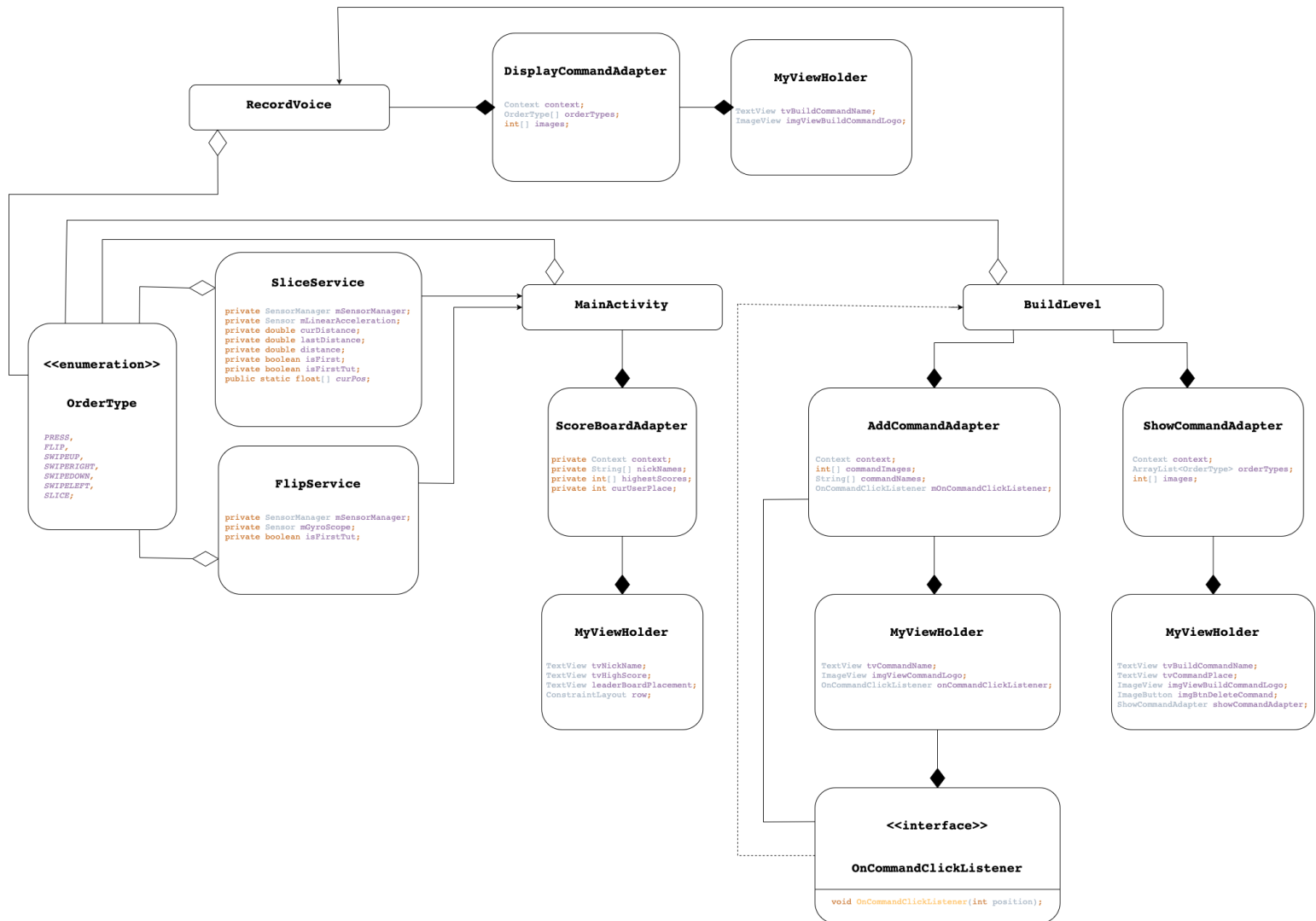
דיאלוג סוף משחק



תרשים זרימת המסכים



תרשים UML של המחלקות



מימוש הפרויקט

FlipService

תפקיד המחלקה:

המחלקה אחראית על הפעלת הגיירו סנסור ולבדוק כאשר הטלפון עושה פעולה של סיבוב. כאשר נקלטת הפעולה של הסיבוב המחלקה אחראית ליידע את האקטיביטי האחר שנעשתה פעולה והיא סיבוב.

תכונות המחלקה:

```
private SensorManager mSensorManager;  
private Sensor mGyroScope;  
private boolean isFirstTut;
```

mSensorManager ו-mGyroScope אחראים על הפעלת הגיירו סנסור. isFirstTut אחראי על כך שכאשר הסרוויס מופעל בזמן הדרכה הפעולה תיקלט פעם אחת ותפסיק לקלוט אחרי זה.

פעולות:

```
public int onStartCommand(Intent intent, int flags, int  
startId)
```

הפעולה אחראית על אתחול כל המשתנים בעת הפעלת הסרוויס.

```
public void onSensorChanged(SensorEvent sensorEvent)
```

בפעולה הזאת נעשית בדיקה שהסיבוב שהטלפון עשה נעשה בכיוון הנכון ושהזווית שנוצרה מספיק גדולה. בתוך הפעולה נעשית בדיקה נוספת אם הסרוויס הופעל מהדרכה משחק שהשחקן יצר או משחק רגיל ובהתאם לכך נעשים צעדים שונים.

```
public IBinder onBind(Intent intent)  
public void onAccuracyChanged(Sensor sensor, int i)
```

פעולות שנרשמות מ-SensorEventListener ומ-Service ולא עשיתי בהם שינוי.

SliceService

תפקיד המחלקה:

המחלקה אחראית על הפעלת הסנסור אקסלרציה ולבדוק כאשר הטלפון עושה פעולה של חתיכה. כאשר נקלטת הפעולה של החתיכה המחלקה אחראית ליידע את האקטיביטי האחר שנעשתה פעולה והיא חתיכה.

תכונות המחלקה:

```
private SensorManager mSensorManager;  
private Sensor mLinearAcceleration;  
private double curDistance;  
private double lastDistance;  
private double distance;  
private boolean isFirst;  
private boolean isFirstTut;  
public static float[] curPos;
```

mSensorManager ו-mLinearAcceleration אחראים על הפעלת הסנסור אקסלרציה. curDistance, lastDistance, distance, isFirst, isFirstTut אחראי על כך שכאשר הסרוויס מופעל בזמן הדרכה הפעולה תיקלט פעם אחת ותפסיק לקלוט אחרי זה. curPos אחראי על שמירת המיקום הנוכחי של הטלפון, התכונה היא סטטית כיוון שהיא מקבלת ערכים מ-GameActivity ואין עצמים שדרכם אפשר לפנות לתכונה הזאת.

פעולות:

```
public int onStartCommand(Intent intent, int flags, int  
startId)
```

הפעולה אחראית על אתחול כל המשתנים בעת הפעלת הסרוויס.

```
public void onSensorChanged(SensorEvent sensorEvent)
```

בפעולה הזאת נעשית בדיקה שהכיוון שאליו נעשית האקסלרציה היא ואלידית והמרחק מספיק גדול למען עדכון המשחק שהפעולה התבצעה. בתוך הפעולה נעשית בדיקה נוספת אם הסרוויס הופעל מהדרכה משחק שהשחקן יצר או משחק רגיל ובהתאם לכך נעשים צעדים שונים.

```
public IBinder onBind(Intent intent)
```

```
public void onAccuracyChanged(Sensor sensor, int i)
```

פעולות שנדרשות מ-SensorEventListener ומ-Service ולא עשיתי בהם שינוי.

```
public float[] getCurrentPosLinear(SensorEvent event)
```

הפעולה מחזירה את המיקום הנוכחי של הטלפון

OrderType (enum)

תפקיד המחלקה:

המחלקה אחראית על זיהוי הפעולות בקלות.

תכונות המחלקה:

```
PRESS,  
FLIP,  
SWIPEUP,  
SWIPERIGHT,  
SWIPEDOWN,  
SWIPELEFT,  
SLICE;
```

פעולות:

```
public static OrderType fromInteger(int x)
```

פעולה המקבלת מספר ומחזירה את ה-enum התואם למספר.

ScoreBoardAdapter

תפקיד המחלקה:

המחלקה אחראית על קישור ה-recyclerview בדיאלוג leaderboard לתכונות והנתונים שהוא צריך לייצג.

תכונות המחלקה:

```
private Context context;  
private String[] nickNames;  
private int[] highestScores;  
private int curUserPlace;
```

context אחראי על מה שיקבל ה-inflater.
nickNames ו-highestScores אחראים על התכונות שיוצגו.
curUserPlace הוא המיקום הנוכחי של השחקן המחובר בלוח התוצאות. המיקום של השחקן מסומן בצבע שונה על מנת שיהיה לשחקן יותר קל להתמצא.

פעולות:

```
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup  
parent, int viewType)
```

הפעולה אחראית על יצירת inflater ויצירת ViewHolder חדש.

```
public void onBindViewHolder(@NonNull MyViewHolder holder, int  
position)
```

הפעולה הזאת אחראית לקשירת כל הוויג'טים למשתנים ולשינוי הצבע של המיקום הנוכחי של השחקן.

MyViewHolder

תפקיד המחלקה:

המחלקה אחראית על יצירת משבצת ייצוג אחד במחלקה `ScoreBoardAdapter`.

תכונות המחלקה:

```
TextView tvNickName;  
TextView tvHighScore;  
TextView leaderBoardPlacement;  
ConstraintLayout row;
```

`tvNickName`, `tvHighScore` ו-`leaderBoardPlacement` מייצגים את כל הוויג'טים מהם מורכב `ViewHolder`. `row` אחראי על שינוי הרקע לצבע שונה למען הבהרה לשחקן היכן הוא נמצא בלוח תוצאות.

ScoreBoardAdapter

תפקיד המחלקה:

המחלקה אחראית על קישור ה-recyclerview בדיאלוג leaderboardDialog לתכונות והנתונים שהוא צריך לייצג.

תכונות המחלקה:

```
private Context context;  
private String[] nickNames;  
private int[] highestScores;  
private int curUserPlace;
```

context אחראי על מה שיקבל ה-inflater.
nickNames ו-highestScores אחראים על התכונות שיוצגו.
curUserPlace הוא המיקום הנוכחי של השחקן המחובר בלוח התוצאות. המיקום של השחקן מסומן בצבע שונה על מנת שיהיה לשחקן יותר קל להתמצא.

פעולות:

```
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup  
parent, int viewType)
```

הפעולה אחראית על יצירת inflater ויצירת ViewHolder חדש.

```
public void onBindViewHolder(@NonNull MyViewHolder holder, int  
position)
```

הפעולה הזאת אחראית לקשירת כל הוויג'טים למשתנים ולשינוי הצבע של המיקום הנוכחי של השחקן.

MyViewHolder (ScoreBoardAdapter)

תפקיד המחלקה:

המחלקה אחראית על יצירת משבצת ייצוג אחד במחלקה ScoreBoardAdapter.

תכונות המחלקה:

```
TextView tvNickName;  
TextView tvHighScore;  
TextView leaderboardPlacement;  
ConstraintLayout row;
```

tvNickName, tvHighScore ו- leaderboardPlacement מייצגים את כל הוויג'טים מהם מורכב ViewHolder. row אחראי על שינוי הרקע לצבע שונה למען הבהרה לשחקן היכן הוא נמצא בלוח תוצאות.

AddCommandAdapter

תפקיד המחלקה:

המחלקה אחראית על קישור ה-recyclerview בדיאלוג addCommandDialog לתכונות והנתונים שהוא צריך לייצג.

תכונות המחלקה:

```
Context context;  
int[] commandImages;  
String[] commandNames;  
OnCommandClickListener mOnCommandClickListener;
```

context אחראי על מה שיקבל ה-inflater.
commandImages ו-commandNames אחראים על התכונות שיוצגו.
mOnCommandClickListener נעשית המחיקה של הפעולות מן הרשימה שהשחקן יצר.

פעולות:

```
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup  
parent, int viewType)
```

הפעולה אחראית על יצירת inflater ויצירת ViewHolder חדש.

```
public void onBindViewHolder(@NonNull MyViewHolder holder, int  
position)
```

הפעולה הזאת אחראית לקשירת כל הוויג'טים למשתנים.

MyViewHolder (AddCommandAdapter)

תפקיד המחלקה:

המחלקה אחראית על יצירת משבצת ייצוג אחד במחלקה AddCommandAdapter.

תכונות המחלקה:

```
TextView tvCommandName;  
ImageView imgViewCommandLogo;  
OnCommandClickListener onCommandClickListener;
```

tvCommandName ו-imgViewCommandLogo מייצגים את כל הוויג'טים מהם מורכב ViewHolder. onCommandClickListener אחראי על מחיקת פעולה מהרשימה הייצוגית שנוצרת כשהמשתמש בוחר פעולה.

ShowCommandAdapter

תפקיד המחלקה:

המחלקה אחראית על קישור ה-recyclerview בדיאלוג showCommandDialog לתכונות והנתונים שהוא צריך לייצג.

תכונות המחלקה:

```
Context context;  
ArrayList<OrderType> orderTypes;  
int[] images;
```

context אחראי על מה שיקבל ה-inflater.
orderTypes ו-images אחראים על התכונות שיוצגו.

פעולות:

```
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup  
parent, int viewType)
```

הפעולה אחראית על יצירת inflater ויצירת ViewHolder חדש.

```
public void onBindViewHolder(@NonNull MyViewHolder holder, int  
position)
```

הפעולה הזאת אחראית לקשירת כל הוויג'טים למשתנים.

MyViewHolder (ShowCommandAdapter)

תפקיד המחלקה:

המחלקה אחראית על יצירת משבצת ייצוג אחד במחלקה ShowCommandAdapter.

תכונות המחלקה:

```
TextView tvBuildCommandName;
```

```
TextView tvCommandPlace;  
ImageView imgViewBuildCommandLogo;  
ImageButton imgBtnDeleteCommand;  
ShowCommandAdapter showCommandAdapter;
```

tvBuildCommandName ,tvCommandPlace ,imgViewBuildCommandLogo ,imgBtnDeleteCommand
מייצגים את כל הוויג'טים מהם מורכב ViewHolder.
showCommandAdapter אחראי על מחיקת פעולה מהרשימה שנוצרת כשהמשתמש בוחר פעולה.

DisplayCommandAdapter

תפקיד המחלקה:

המחלקה אחראית על קישור ה-recyclerview בדיאלוג displayCommandDialog לתכונות והנתונים שהוא צריך לייצג.

תכונות המחלקה:

```
Context context;
```

```
OrderType[] orderTypes;  
int[] images;
```

context אחראי על מה שיקבל ה-inflater.
orderTypes ו-images אחראים על התכונות שיוצגו.
הסיבה ש-orderTypes הוא מערך ולא רשימה כמו ב-ShowCommandAdapter היא שלא יעשה שינוי במערך כיוון שהוא נוצר לאחר שהשחקן סיים ליצור את הרשימה.

פעולות:

```
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup  
parent, int viewType)
```

הפעולה אחראית על יצירת inflater ויצירת ViewHolder חדש.

```
public void onBindViewHolder(@NonNull MyViewHolder holder, int  
position)
```

הפעולה הזאת אחראית לקשירת כל הוויג'טים למשתנים.

MyViewHolder (ShowCommandAdapter)

תפקיד המחלקה:

המחלקה אחראית על יצירת משבצת ייצוג אחד במחלקה ShowCommandAdapter.

תכונות המחלקה:

```
TextView tvBuildCommandName;  
ImageView imgViewBuildCommandLogo;
```

tvBuildCommandName ו-imgViewBuildCommandLogo מייצגים את כל הוויג'טים מהם מורכב
.ViewHolder

אלגוריתמים

יצירת רשימה רנדומלית של פעולות:

הפעולה יוצרת מערך מסוג `OrderType` באורך שהתקבל והפעולות שמוכנסות לרשימה הן על פי רמת הקושי שהתקבלה.

הפעולה יוצרת מספר רנדומלי מ-0 עד 3 ברמה הקלה. במידה ויוצא 3 (Swipe) מגרילים מספר חדש כדי לקבוע את כיוון הפעולה. זה נעשה מתוך מחשבה שלא יהיו יותר מדי מהפעולה Swipe. אם הרמת קושי היא מתקדמת אז נוספת עוד פעולה להגרלה. בנוסף לזה נעשית בדיקה אם פעולה הופיעה פעמיים רצוף, במידה וכן היא לא יכולה להופיע פעם שלישית.

```
public OrderType[] BuildOrderArray(int arrLength, int
levelDiff)//builds order array with random orders according to
difficulty
{
    OrderType[] orderArray = new OrderType[arrLength];
    Random random = new Random();
    int lastOrder = 50;//so it wont be equal to any order
    boolean beenDouble = false;// if same order was twice it
cant be a third time
    if (levelDiff == 1) {
        for (int i = 0; i < arrLength; i++) {
            int temp = random.nextInt(3);
            if (beenDouble)//if order was twice in a row
            {
                while (temp == lastOrder)//while current order
is equal to last 2
                {
                    temp = random.nextInt(3);//pick a new order
                }
                beenDouble = false;
            }
            if (temp == lastOrder) {
                beenDouble = true;
            }
            if (temp == 2)//in order to not make most orders
swipe(direction) only if random is equal to 2 the order will
be swipe(direction)
            {
                temp = random.nextInt(4) + 2;
            }
            orderArray[i] = OrderType.fromInteger(temp);
            lastOrder = temp;
        }
    } else {
```

```

        for (int i = 0; i < arrLength; i++) {
            int temp = random.nextInt(4);
            if (temp == 2)//in order to not make most orders
swipe(direction) only if random is equal to 2 the order will
be swipe(direction)
                {
                    temp = random.nextInt(4) + 2;
                } else {
                    if (temp == 3)//if equal to three make the
order slice
                        {
                            temp = 6;
                        }
                    }
                if (beenDouble)//if order was twice in a row
                {
                    while (temp == lastOrder)//while current order
is equal to last 2
                    {
                        temp = random.nextInt(4); //pick a new order

                        if (temp == 2)//in order to not make most
orders swipe(direction) only if random is equal to 2 the order
will be swipe(direction)
                            {
                                temp = random.nextInt(4) + 2;
                            } else {
                                if (temp == 3)//if equal to three make
the order slice
                                    {
                                        temp = 6;
                                    }
                                }
                            }
                        beenDouble = false;
                    }
                    if (temp == lastOrder) {
                        beenDouble = true;
                    }
                    orderArray[i] = OrderType.fromInteger(temp);
                    lastOrder = temp;
                }
        }
        return orderArray;

```

}

מסד נתונים

אני השתמשתי במסד הנתונים `firestore` של `firebase`. בחירה זאת נעשתה מתוך מחשבה על פשטות ההכנסה/הוצאה והאלידציה של הנתונים והמשתמשים שנעשית בפשטות יתרה על ידי כמה פעולות פשוטות. בנוסף על כך `firebase` אינו משתמש ב-`sql` ולכן בחרתי בו.

כאשר משתמש חדש נרשם למשחק, למסד נתונים נכנס מסמך חדש לקולקציה Users שבו נמצאים הנתונים הבאים: אימייל, שם משתמש ו4 התוצאות הכי גבוהות של השחקן (מוכנס 0 לכל אחת). בנוסף על כך מוכנס לקולקציה שונה בשם Reviews את שם המשתמש ואת כמות התגובות (מוכנס 0). בעת ההרשמה נעשית בדיקה בעזרת האימייל והסיסמא של השחקן לראות האם הוא קיים. בעת הכנסת תגובה חדשה מוכנס מסמך חדש תחת הקולקציה reviews שנמצאת תחת המסמך של השחקן שנמצאת בקולקציה Reviews. במסמך נמצאת התגובה החדשה ובנוסף מספר התגובות באותו קולקציה עולה באחד. בעת הכניסה ללוח התוצאות מוחזרים כל התוצאות של כל השחקנים ברשימה ממוינת ובנוסף הם ממוינים לפי סוג משחק ורמת קושי, ואז המשתמש יכול לדפדף בין הלוחות לראות את התוצאות. בתחילת כל משחק נשלפות התוצאות הכי גבוהות של השחקן מן המסד נתונים וכאשר משתמש מסיים את המשחק נעשית בדיקה האם תוצאתו יותר גבוהה, במידה וכן היא מוכנסת למסד נתונים על פי רמת הקושי וסוג המשחק.

מדריך למשתמש

אני השתמשתי ב android studio arctic fox 2020.3.1 patch 4 כסביבת העבודה לבניית הפרויקט והשתמשתי בטלפון האישי שלי (samsung +21) בשביל להריץ אותו.

לאורך כל הפרויקט השתמשתי בהודעות מסוג Toast כדי לתת פידבק למשתמש שמהו קרה. לדוגמא לאחר שהמשתמש משאיר תגובה הוא מקבל הודעה "תודה על התגובה!" ובדומה לכך גם כאשר המשתמש נרשם למערכת הוא מקבל הודעה, במידה וההרשמה הצליחה הוא יקבל הודעה כזאת: "משתמש הוכנס למערכת בהצלחה" ובמידה והייתה תקלה המשתמש יקבל הודעה ובה כתובה התקלה בהתאם.

רפלקציה

העבודה על הפרויקט הייתה עבורי מאתגרת, אך בו זמנית מהנה במיוחד. אני חושב שהחלק שהכי אהבתי בפרויקט, היה שלב ה-"ניסויים" ושלב הכנת ממשק המשתמש ותכנות המשחק עצמו.

המסקנה שלי בנוגע לכל התהליך היא שכתובת הפרויקט הזה פיתחה את צורת המחשבה שלי, מכיוון שהיום אני חושב בצורה יותר הגיונית ולוגית מאשר לפני הכתיבה שלו וגם הכתיבה הקנתה לי המון כלים שגיליתי בצורה עצמאית.

אילו הייתי מתחיל לעבוד על הפרויקט היום הייתי משנה את הגישה שהייתה לי לקוד, צורת הכתיבה שלי הייתה מבולגנת ולא יעילה.

עמדו בפניי קשיים – חלקם לוגיים, חלקם היו יותר נוגעים לסביבת העבודה ולאופן התכנות, אך בסופו של דבר, אני מאמין שהצלחתי להתגבר על הקשיים וליצור פרויקט מצוין. המסקנות שלי מן הפרויקט הן – לא לוותר בקלות, להמשיך וללמוד, גם כשקשה, ובסופו של דבר – להגיע להצלחה. אני חושב שהכלי החזק ביותר שקיבלתי מהכנת הפרויקט היה חיזוק יכולת הלמידה העצמית – השימוש בגוגל ככלי לפתרון בעיות וקבלת מענה לשאלות. שכן – כל שאלה הנוגעת לתכנות ובה נתקלתי, מצאתי לה תשובה בגוגל. לא תמיד הפתרון היה הפתרון שלו ציפיתי, אך בהחלט חלק מהפתרונות שהיו לקהילת המתכנתים להציע עבדו.

במידה והיה לי עוד זמן הייתי מוסיף פיצ'ר של מולטיפלייר כלומר ששני אנשים יוכלו להתחרות אחד בשני מ2 טלפונים שונים. למרות שאני מרגיש שהשקעתי המון זמן בפרויקט אני מרגיש שאם אשקיע עוד שעות אוכל להפיק תוצאה מרהיבה הרבה יותר.

ביבליוגרפיה

RecyclerView - Android Studio Tutorial | Part 1

https://www.youtube.com/watch?v=18VcnYN5_LM

Get data with Cloud Firestore

<https://firebase.google.com/docs/firestore/query-data/get-data>

Firestore | Sorting and Limiting documents

https://www.youtube.com/watch?v=NTkloBcBy_U

Voice recorder in android studio | how to create voice recorder app in android studio | Audio record

<https://www.youtube.com/watch?v=z3Gx4whgWcY>

MediaPlayer documentation

<https://developer.android.com/reference/android/media/MediaPlayer>

Add and Remove Items From Recycler View | Android Studio Tutorial

<https://www.youtube.com/watch?v=LQmGU3UCOPQ>

RecyclerView OnClickListner (Best practice way)

<https://www.youtube.com/watch?v=69C1ljfDvl0>

Keep a User logged in

<https://youtu.be/5uZjGpwzOkE>

How to Create a Custom Button (With Images) in Android Studio

<https://www.youtube.com/watch?v=Nn4-Vn7qk9k>

How to Create Instruction Dialog in Android Studio | AlertDialog | Android Coding

<https://www.youtube.com/watch?v=qdLmWPXQPnk>

Firebase Android Tutorial 3 - Firebase Login Authentication and Sign Out

<https://www.youtube.com/watch?v=ItqsK5uhLBg>