TAPPY

A series of minigames to cure boredom during quarantine

Group 3

Dai Dung Nguyen 300319276

Tarane Talebi 300309630

Victor Suleiman 300315653

Link for video presentation: https://youtu.be/qn4kM1qwLs8

Link for app repository: https://github.com/JamesDaiD/TappyForAndroid-Group3

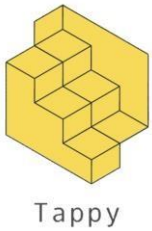


Figure 1 – Project logo.

Motivation for the App

Due to current restrictions implemented by the world's recent events, our group decided that a modular minigames app to play during the boredom hours that a quarantine may impose would be interesting to users, and at the same time an effective way to be more creative, test our knowledge during class, and expand upon it, both as individuals and in a group. By using GitHub, we kept the project up to date and committed all the changes with descriptions to keep our teammates on track. We also held meetings every week to touch base and discuss some difficulties that were challenging to overcome on our own.

Individual Member Contribution

As two team members made 2 games each and one team member made 1 game but also the leaderboard database implementation, the group decided that the contribution was to be divided equally amongst the 3 group members.

Table 1 – Individual Member Contributions.

Member	Contribution (%)
Dai Dung Nguyen	33.4%
Tarane Talebi	33.3%
Victor Suleiman	33.3%

Activities and General Concepts

Tappy is a minigames app that consists of 10 activities:

1. Splash activity
2. Main Activity
3. Level Selector
4. TapTorial
5. Tic Tap Toe
6. Tapman
7. Tap That Song
8. Simon Tap
9. Reaction Tap
10. Leaderboard

Some general concepts used in these activities are:

- Graph view
- List view
- Grid view
- TaskTimer
- Database implementation
- OnClickListeners: Spinner, Buttons, Images
- Animation
- MediaPlayer
- Methods
- DBhelper
- Relative layout
- EditText

On each game described below, we will go into more details about their main functionalities, both learned in class and applied on them and extensions from our learnings in class.

Splash Activity

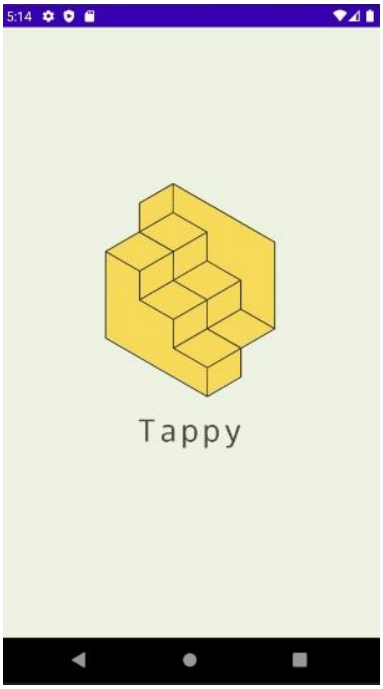


Figure 2 – Splash Activity screen.

When you enter a game, you don't want to be introduced to a page full of instructions. That is why we use splash activity to start the application to introduce the client to the theme and feel of the app.

Splash activity has an activity of its own. We create a `TimerTask` object that overrides the run method (generated by the intellisense). Inside the `run()` we will tell once the `TimerTask` is done `finish()` , then Start activity, which in our case is the Main Activity. We will also create a `Timer` object so that we can schedule the duration for our splash activity after the `run()` is done.

Key point for splash activity is to ensure that in the manifest the launching activity is set to Splash activity as our MAIN launcher.

Title Screen (Main Activity)

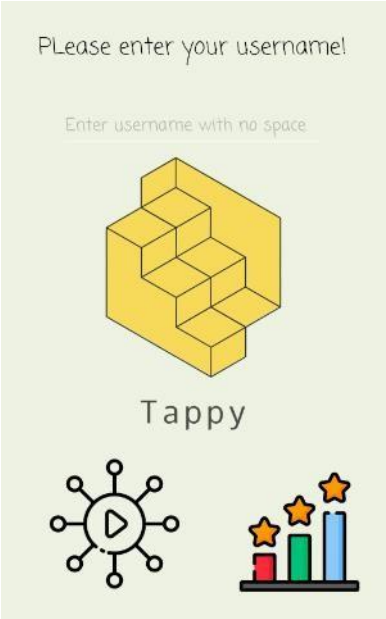


Figure 3 – Main Activity screen.

The main activity of our project consists of three main widgets.

- Edit text, which is a field for the user to input his username. Then, the username is saved into the Shared Preferences of the device, so it can be used to record the user's scores that will later be shown in the game's leaderboard If there is no text on the username field, the username "Anonymous" will be created by default.
- Play button, which uses an extension of the class learnings by implementing an Image Button on the layout. When pressed, it takes the user to the Level Selector activity and it also triggers a spinning animation, which is another class knowledge extension.
- Leaderboard button, which is displayed as a graph with stars next to the play button. This is another image button that takes the user to the Leaderboard activity, further explained in this report.
- When the user clicks the play button, a little animation will play. When this animation ends (using Animation Listener and `onAnimationEnd()`), the username entered in the EditText will be recored in the sharedPrefs to be used with other activities and recorded in our Database

Level Selector

Figure 4 – Level Selector screen.

After the main activity, we are introduced to the list of games that we can choose from. The grid view has a layout that contains the ImageView and TextView widgets and will get inflated in the GridAdapter. In our main GridList layout we only have a grid view widget inserted.

In our adapter we declare two lists, one of string and one of integer type that will be filled with the images and the name of the games. In the view method of our adapter, we will inflate the grid layout and set the text and image for current item using data from map.

In the GridList activity we will make two lists of the same nature and fill them with the game names and images for the games that we will use to input into the adapter constructor. Then we declare the setOnItemClickListener that will follow a switch case that will lead to a start activity of the respective game that needs to be played based on the users click position. For aesthetics, we input horizontal and vertical spacing.

TapTorial

Figure 5 - TapTorial screen.

TapTorial is just a tutorial game for the player to understand how the rest of the games work. The player will be welcomed to the game and the button will ask the player to press it 10 times. Each time the player presses the button, the color changes and the text on the button as well, showing the number of taps left. When the player finishes tapping the button 10 times, a toast will be displayed, informing the time the player spent to press the button and encouraging him to go play the other games. Another toast after that mentions the player can go back using the back button on the device.

Tic Tap Toe

Figure 6 - Tic Tap Toe screen.

A minigame that stands out and has existed since the dawn of humanity is Tic Tac Toe, and it is always a good exercise to try to implement it on different platforms using different language implementations. On this project, Tic Tap Toe is a game of Tic Tac Toe in which users play against the computer and need to win 3 rounds for the game to be over. The player is always the "X", so he/she always starts. CPU is therefore always the "O" and it will always play after the player, until the player wins or there are no more free tiles to play, in which the round will be a draw, where no one scores. When the player presses a button for the first time, the game will be considered started and the timer will be started as well, using a support class called `TimeRecorder` (which also was used on the Simon Tap game). If the CPU wins the game, the timer and score will be reset and the player will have lost. On the other hand, if the player wins the game, the score will also be reset and the time and scoreboard will be added to the scores table in the database.

The layout was made using a Linear Layout, which has a Relative Layout and 3 Linear layouts inside it, the former for displaying title, scoreboard and win condition, and the latter for displaying each button in a row, with 3 buttons each, so it has 9 in total. They have the same weight on each nest, so it can be displayed like a grid, and because it is linear, the game is also responsive and playable with a rotated device in widescreen.

The game state transitions using the events inside each button listener. If there's already a symbol on a button, it does nothing, and if it's empty, it will draw a symbol on it. After that, it checks if the player won, and if not, then the CPU plays it. After each play, the game checks if that player who just played won the game or not and if not, it switches turns. When a round is over, a runnable is issued so the user can have time to see where the diagonal, column or row win sequence is, before resetting the entire board. The scoreboard updates and a Toast is displayed showing whoever won the round. That occurs when the game is over as well, using the `TimeRecorder` object again, which also displays the time elapsed since the game started.

Two interesting features the game has are the CPU AI and the ability to maintain the game state when the device is rotated. The first one checks if the player will win by row, column or diagonal on his/her next round and blocks it by drawing the "O" on the empty space that would make the player win otherwise. If the player cannot win the next round, it will make a list of available positions and choose a random position to play. The intention of the CPU on this game is not to try to win – since the CPU is always the second to play, its strategy is to block the player and making the game not so easy, so the player must try to find a way to win the round by two ways, as one will always be blocked by the CPU. The second feature uses a bundle to save the current game values like the scoreboard, turn, game state and time, to not lose them once the instance state is restored after the device rotation – it overrides the `onSaveInstanceState` and `onRestoreInstanceState` functions to accomplish that.

Class concepts implemented

- Linear Layout
- Changing text inside a Text View
- Implementing Button Listeners
- Toasts
- Arrays and Array Lists
- String functions
- Using Log functions for debug purposes
- Bundles
- SQLite databases

Class extensions and new concepts learned

- Relative Layout
- Inserting fonts from the internet to the project
- Grabbing button ID's programmatically
- Runnables
- Timers
- Random class
- Maintaining activity state when device is rotated

TapMan

Figure 7 – TapMan screen.

TapMan is basically a hang man game. The layout has 26 buttons for every letter in the alphabet, Image view for the hang man display, one text view for messages, one text view for the word to guess, spinner for choosing category. Instead of generating an on click for all the buttons in the activity, an onclick in the XML code is declared which calls one general method.

The logic in this activity is implemented so that after a category is chosen, a word is randomly picked from the table of the respective category and dashes are generated in return based on the numbers of letters in the word. After that, based on each click the user does on the letters, if the letter selected is corrected it will fill its spot with the letter, and if not, a step is added to the image. We also have a hint option that will generate a toast displaying a message chosen from the third column of the chosen word table. If you make 6 mistakes you will lose, and a message will appear, and you can choose a new category. If you choose the word correct the win message is displayed and the word, the number of tries, and whether you have used hint or not will be imported into the Tappy database.

Methods:

- `getRandom()`

This method is used on List <String[]> to choose a random word from our database.

- `interlace()`

This method is used to create a space between the dashes that are generated per letter in a word.

- `TouchLetter()`

This method is called per each click on each letter. It goes through conditions where if the letter exists in the word, input it and if not don't input it and change the image one step further. If the user has reached the max 6 wrong tries display the final image and game over text.If there are no more dashes and the word is guessed show the win pose and the win message. It also changes the colors of the letters so that the user doesn't make the same mistake twice.

- `LetterColorBack()`

This method is used to set all the buttons colors back to their gray state after a new category is chosen.

- `CreateDB()`

Creates database.

- `CreateTables()`

Create tables and drop them.

- `ReadCSV()`

Reads the CSV file by getting the file name. Reads every line and separates each column by a coma and inputs into an array.

- `AddWordsTo2020(), addWordsToAnimals(), addWordsToActivities()`

Inserts content into the respective data (that will late be filled in the on create by going through a for loop that will the columns by reading the file that is being read through ReadCSV( )).

Class concepts implemented

- Constraint Layout
- Toast
- List of drawables
- Array Lists
- Using Log functions for debug purposes
- SQLite databases
- Spinner

Class extensions and new concepts learned

- Button onclick: in XML
- Dynamic Resources Binding
- Using SQLite Helper class
- Modifying GUI elements
- Great drawing skills!!! (Using Paint)

TapThatSong

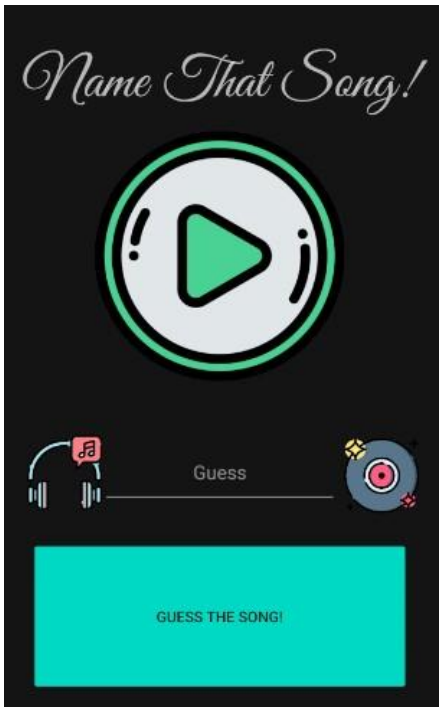


Figure 7 – Tap That Song screen.

Tap That Song's goal is to correctly input the name of the song that is currently playing. To compute that, there are a few steps implemented.

A file with data about all songs is read, this data will be parsed and put inside Song objects. These objects are stored inside a `List<Song>`. When the activity start, a random Song is selected and a timer starts running the moment the user hears the song (a boolean variable keeps track of whether the user has heard the song or not). The user enters their guess into the Edit Text. This will then be compared with the song name stored inside the object. There is some fuzzy logic and regex used to make sure close guesses are accepted.

If the User guesses right => Do a little animation, get the time they needed to guess, reset timer, get a new song. There will be an extra animation when the user guesses right on their first try.
If the user guesses wrong => Display hints (this can happen a few times). Hints include album art, artist name and trivia. Album art is displayed via as an image view with a relative layout.

The game can be repeated, the timer and song generator will work properly. Resources are accessed with dynamic names, and data is read from a CSV file and put into objects. Play/Pause button works as intended and the song always resume from where paused, unless a new song is gotten.

- Uses Relative layout and Image View to set a background that's tinted and centered.
- A method is used to make sure all resources are bound and working correctly.
- Uses quite a few animations.

Class concepts implemented

- Constraint Layout
- Button "on click" listeners
- Toast
- Array Lists
- Media Players
- Using Log functions for debug purposes
- File Reading
- SQLite databases

Class extensions and new concepts learned

- Working with XML
- Animation: XML and Android
- Image Button
- Relative Layout
- Random class
- Resuming Media Player
- Importing and using Fonts
- Timers
- Working with transparency and alpha values
- Dynamic Resources Binding
- Regular Expressions
- Overriding System Buttons' Behavior

Simon Tap

Figure 8 - Simon Tap screen.

Another minigame which is interesting to implement is Simon, a game in which a sequence of colors along with sounds is shown to the player and he/she needs to press the buttons to match the sequence that was displayed correctly. Then, the same sequence is shown but with one more color and sound, and the game goes on until the player inputs the incorrect sequence. For the Simon implemented on Tappy called Simon Tap, since one of the purposes of the app is to create a leaderboard based on time scores, the game has a sequence of 7 colors. In the first round, it shows the user just 1 color, then 2, until a sequence of 7. Then the game is won when the player correctly inputs all 7 colors shown in the right order.

As the figure shows, the layout is pretty straight-forward: a constraint layout with the title, a button to start the game and 4 diamond-shaped buttons with the classic colors of the Simon game. For the game logic, there will be 2 support classes used: `TimeRecorder`, which was also used in Tic Tap Toe, and another one called `ButtonAttribute`, which is used to record the button itself grabbed by its ID, along with its dark and highlighted colors, name and sound it does when the sequence pattern is showing or it is pressed.

If the game hasn't already started, pressing the "Start Game" button will build an array list of 7 randompicked buttons, set the listeners for the 4 buttons on the screen and start recording the timer for the score. After that, it will show the color sequence to the player, which on round 1 will be only 1 color. The showing of the pattern needs to have visual and audible cues to the player, so according to how many colors need to be shown based on the current round number, it will both highlight a button with its highlighted color, and play the button's respective sound cue. And do that one after the other. This is properly done by implementing a countdown timer which will highlight and play a sound on each of its tick.

The button listeners make sure that they don't do anything if the game hasn't started or the sequence is still being shown to the player. Once those states change, on a click, it tests which position of the sequence the player is currently in, and compares it to its own position in the random sequence order assigned when the game was started. If it's wrong, the user is displayed a toast saying the order is wrong and the game is over, only reset when the "Start Game" button is pressed again. Otherwise, it either waits for the next button press or, if there's every button that needs to be pressed, it goes to the next round by showing the same sequence with one more color, unless it's the final round, which upon completion will display a toast saying the game is won, the player's time and will add it into the scores table in the database, along with the number of tries (note that the time is not cumulative: it resets if the player gets a wrong sequence). It is worth pointing out that when the player is inputting a sequence, the buttons will also highlight and play their sound when pressed.

Class concepts implemented

- Constraint Layout
- Button “on click” listeners
- Color strings
- Toasts
- Array Lists
- Media Players
- Using Log functions for debug purposes
- SQLite databases

Class extensions and new concepts learned

- Button rotation
- Setting “on touch” listeners to buttons to change their colors and play sounds
- Random class
- Inserting fonts from the internet to the project
- Timers
- Countdown Timers
- Runnables

ReactionTap

Figure 9 – Reaction Tap start screen.

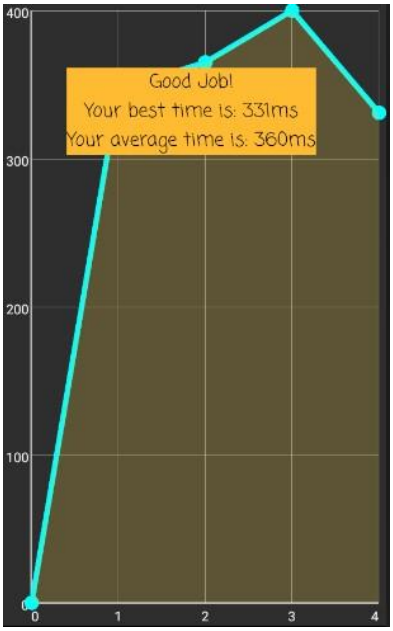


Figure 10 – Reaction Tap graph screen.

Reaction Tap is an interesting game. The concept is simple: you try to click on the green button as fast as possible, as to test one's reaction time.

The game itself has 3 states:

- Wait: red background, get a random time length to wait to transition to the ClickState
- Click: green background, tell the user to click! The button is enabled so the user can click on it. Once it's clicked, you'll be taken to the reaction state
- Reaction: Blue background, the reaction time will be recorded here. Data will then be appended to the series. This will also check if the user has tested 4 times. If yes, they will be taken to the Line Graph, where their results are displayed.

After 4 tries, the min time and avg time is calculated and to be displayed along with the line graph on the next activity.

A bundle is passed from ReactionTap to ReactionGraph, containing the data needed to display the Graph.

Class concepts implemented

- Constraint Layout
- Toast
- Bundle
- Array Lists
- Using Log functions for debug purposes
- SQLite databases and SQL commands
- Linear Layout

Class extensions and new concepts learned

- Importing and using Fonts
- Enum for states
- LineGraphView
- Using SQLiteOpenHelper

Leaderboard

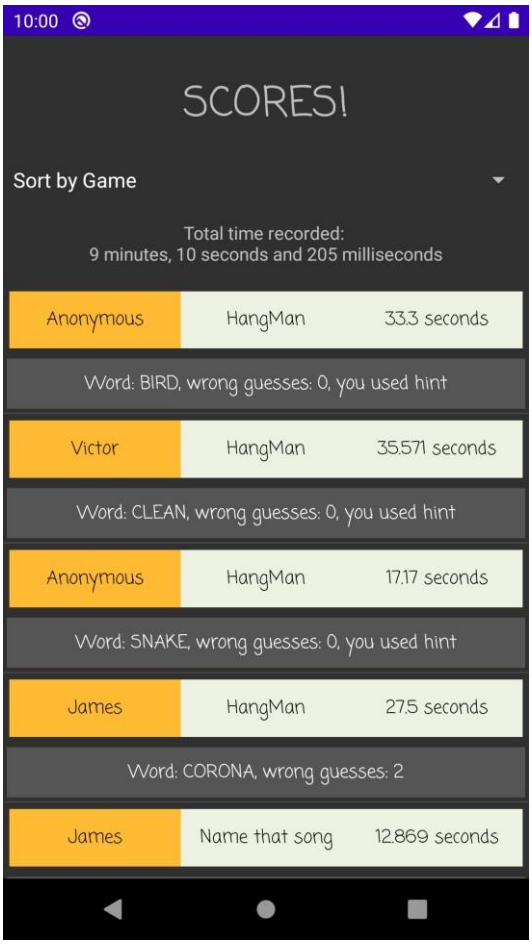


Figure 11 – Leaderboard screen.

This activity uses the DBHelper class to display what's inside our Database. The model used for this listview was created using nesting LinearLayouts, both horizontal and vertical.

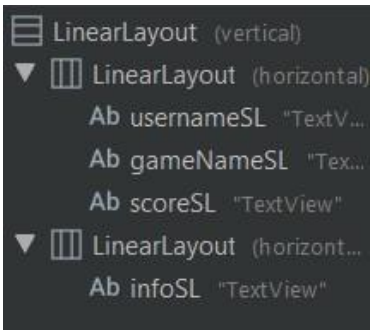


Figure 12 – Leaderboard activity layout.

The listView itself is updated every time you choose a different option. This is done via *notifyDataSetChanged()* and *invalidateViews()*.

Class concepts implemented

- Constraint Layout
- Spinners
- Toast
- Array Lists
- Using Log functions for debug purposes
- SQLite databases and SQL commands
- Linear Layout
- ListView

Class extensions and new concepts learned

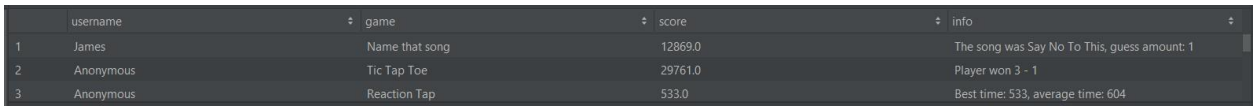
- Importing and using Fonts
- Nesting LinearLayout
- Update ListView when data changed: OnDataSetChanged()
- Using SQLiteOpenHelper

Support Classes: JamesUtilities and DBHelper

JamesUtilities contains several small methods that are used throughout the project. The most used method is formatMilliseconds(). This method converts milliseconds into a proper string. It respects plurals rules and can go up to Year, Day, Hour and Minute. The string will never contain unneeded info (0 year, 0 day etc...). This method is used throughout the app as all our time is recorded in milliseconds.

DBHelper is a class that extends SQLiteOpenHelper. This is how we access and do CRUD operations on our Database.

We have one main table:



	username	game	score	info
1	James	Name that song	12869.0	The song was Say No To This, guess amount: 1
2	Anonymous	Tic Tap Toe	29761.0	Player won 3 - 1
3	Anonymous	Reaction Tap	533.0	Best time: 533, average time: 604

Figure 13 – Leaderboard scores table example.

Each time a game is finished, we call the addUserScore() method to write to the DB. The additional info changes depending on the game and is implemented within each game. Scores value is kept in milliseconds format and converted using JamesUtilities.

Some of the main methods include:

- onCreate(): called on first creation of the DB. This will create our Score table :
    - `CREATE TABLE " + TABLE_NAME + " (username TEXT, " + "game TEXT, " + "score REAL, " + "info TEXT);`
- BrowseScoreRecs: return a List<Score> containing the result of ○ `SELECT * FROM scores;`
    - This method stems a few variations that returns ordered results and even the sum of all time recorded in our DB

Class concepts implemented

- Constraint Layout
- Spinners
- Toast
- Array Lists
- Using Log functions for debug purposes
- SQLite databases and SQL commands
- Linear Layout
- ListView
- SharedPrefs

Class extensions and new concepts learned

- Importing and using Fonts
- Nesting LinearLayout
- Update ListView when data changed: OnDataSetChanged()
- Using SQLiteOpenHelper
- Animation Listeners (for the main Method)

Conclusion

After all the group's work, either by working on the games individually or helping each other with their own games, the result is a solid app that fulfilled the intentions of the project. Class knowledge was implemented, like Button Listeners, activity transition, text editing, media player, database implementation, among others, and knowledge extension beyond class, such as animations, graphs, timers, different layouts, fonts, etc. The group's effort paid off in a fun app that creates a sense of entertainment and joy to the users when they just want to distract themselves by playing simple games on a mobile device. However, the app is challenging at the same time, encouraging players to beat a game faster and test themselves by tying all the games together through the leaderboard functionality.

References

- Android Boot Camp for Developers Using Java®: A Guide to Creating Your First Android Apps, 3rd Edition, by Corinne Hoisington
- Android official documentation: https://developer.android.com/docs
- Tic Tac Toe tutorial: https://www.youtube.com/watch?v=apDL78MFR3o
- Font: https://www.dafont.com/brushot.font
- Font: https://www.dafont.com/retro-computer.font
- Timer and TimerTask tutorial: https://www.youtube.com/watch?v=7QVr5SgpVog
- Graph tutorials: https://www.youtube.com/watch?v=VriiDn676PQ , https://www.youtube.com/watch?v=aslWfx2zT8A
- Hatchful logo creator: https://hatchful.shopify.com/
- All icons and images used are from are free to use
- This app version was made without profit purposes