NAVIGATION

WEEK 2

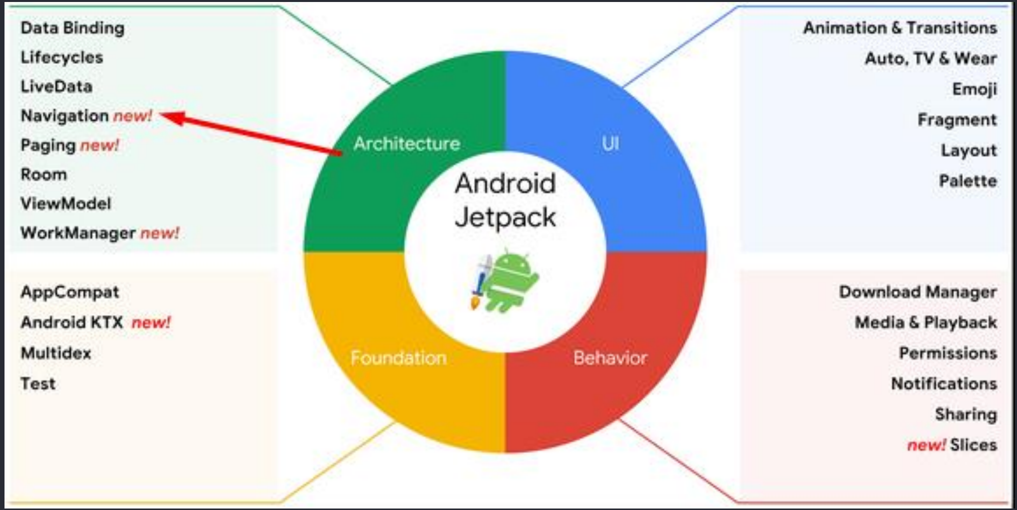
Advanced Native Mobile Programming
1604C062

Informatics Engineering - Universitas Surabaya

A BIT OF JETPACK

In 2018 Google launched the Jetpack.

Jetpack is a set of libraries, components, and tools which helps you to build a better application.



NAVIGATION

Navigation refers to the interactions that allow users to navigate across, into, and back out from the different pieces of content within your app.

Previously in Native Mobile Programming course, you achieved this by implementing “intent” or “fragment manager”

This solution requires you to write different activity/fragment classes and manage the navigation/interaction manually

NAVIGATION COMPONENT

Android Jetpack's Navigation component helps you implement navigation, from simple button clicks to more complex patterns, such as app bars and the navigation drawer.

It excels for apps with multiple Fragments.

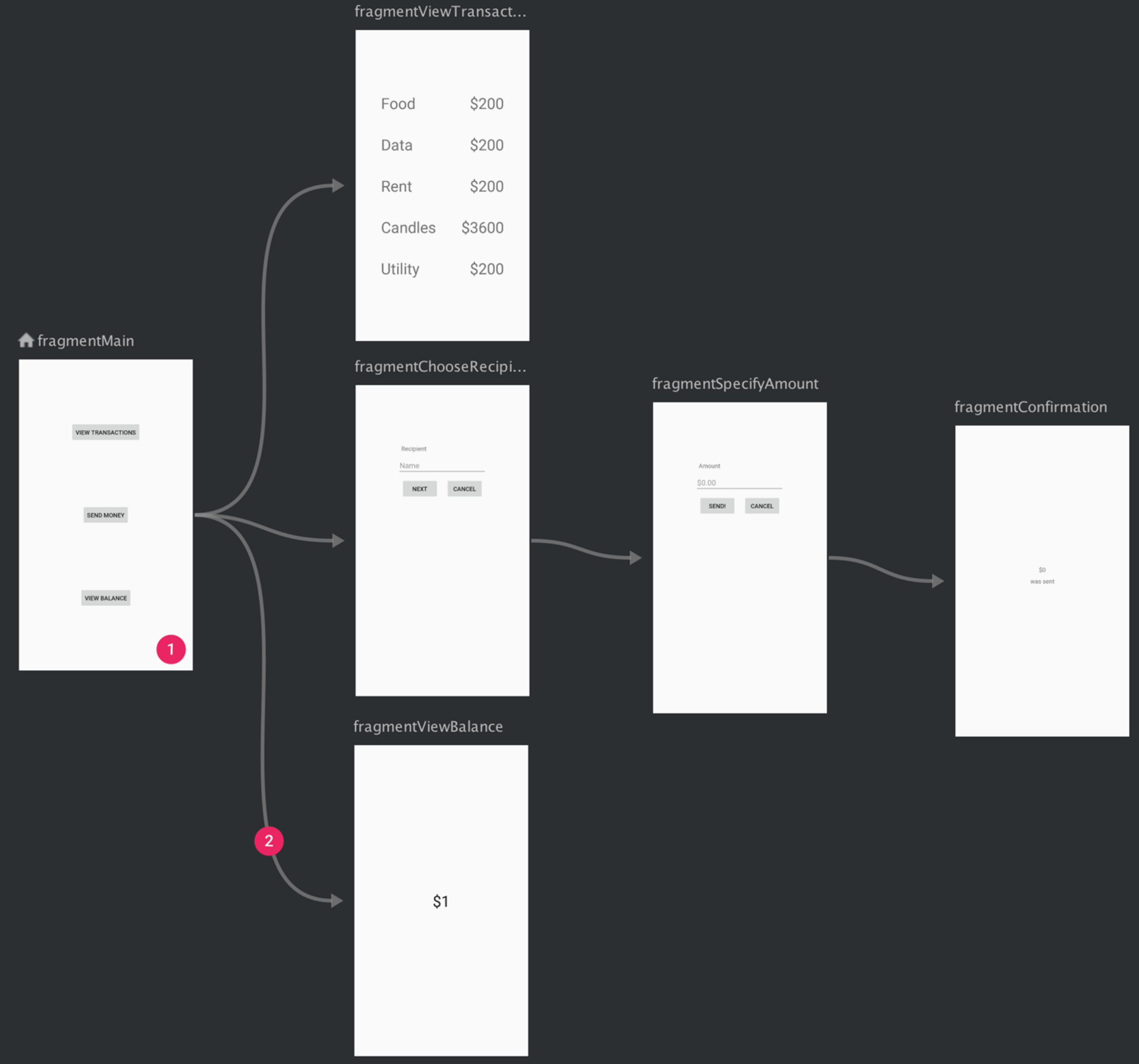
The Navigation component consists of **three key parts**:

- Navigation graph
- NavHost
- NavController

NAVIGATION GRAPH

A navigation graph is a resource file that contains all of your destinations and actions. The graph represents all of your app's navigation paths.

It represent the navigation flow from one fragment to another fragment. It could be simple or complex.



EXAMPLE

navigation graph for a sample app containing six destinations connected by five actions

NAV HOST

The navigation host is an empty container where destinations are swapped in and out as a user navigates through your app.

The Navigation component is designed for apps that have **one main activity with multiple fragment destinations**.

In an app with **multiple activity destinations**, each activity has its own navigation graph.

NAV CONTROLLER

NavController manages app navigation within a NavHost.

Navigation flows and destinations are determined by the navigation graph owned by the controller. These graphs are typically inflated from an Android resource, but, like views, they can also be constructed or combined programmatically or for the case of dynamic navigation structure.

WHY NAVIGATION?

Some additional benefits:

- Fragment transactions are handled automatically
- Back and Up buttons are handled correctly
- Adding default animations and transitions are trivial
- Implements Material Design navigation UI pattern (bottom nav, drawers) with ease
- Navigation can be visualized in Android Studio, allowing easy understanding and editing
- Passing information is easy and safe with help of SafeArgs plugin

TUTORIAL

1. Create new project, name it **as “AdvNRPWeek2”**
   Ex. Adv12345Week2

2. Choose default settings.

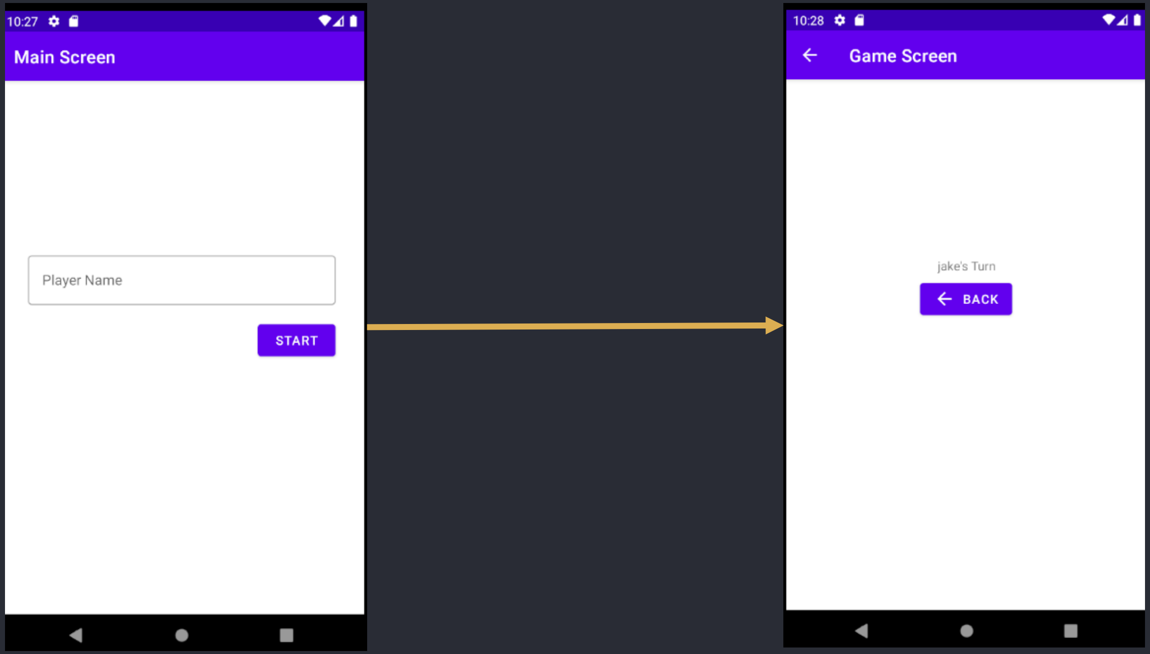
3. Integrate your project with github, and you may use any repo URL.

Tutorial

Single Activity, Multiple Fragments

TUTORIAL INTRODUCTION

In the following tutorial, we'll create single activity app with multi fragments.

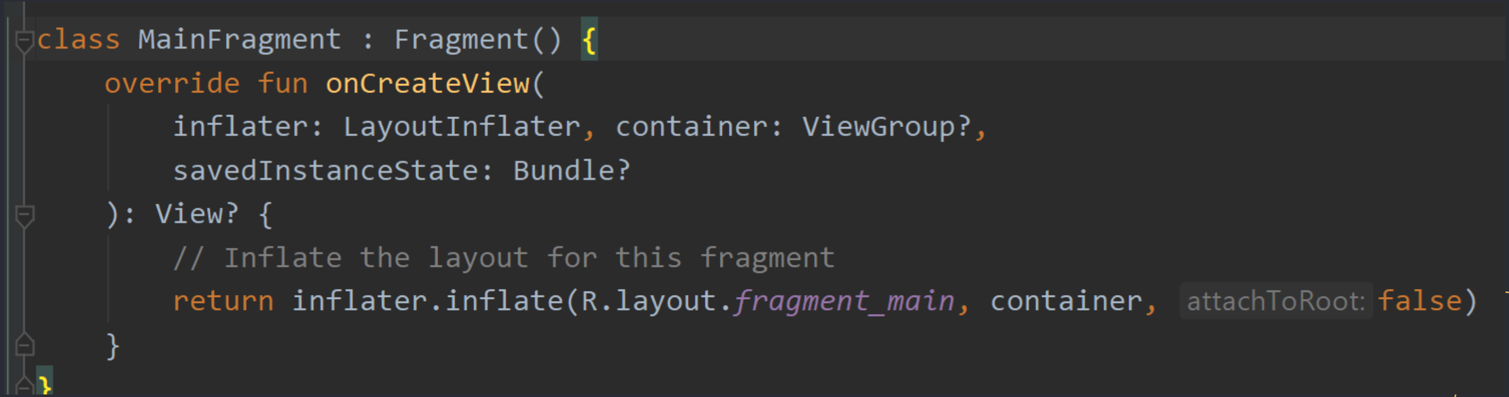


STEP 1: CREATE FRAGMENTS

Start with creating two fragments:

- MainFragment (blank)
- GameFragment (blank)

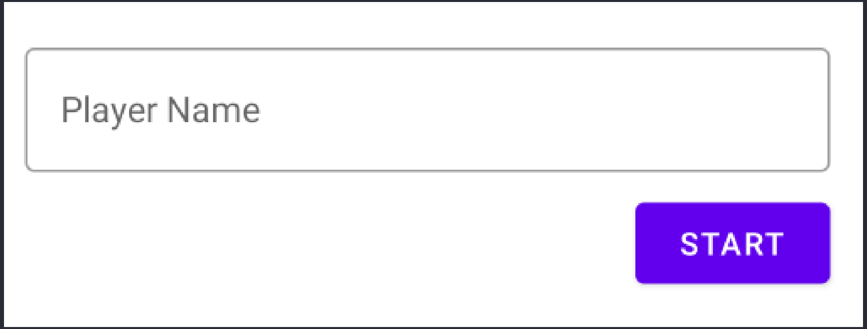
Remove **all todo, all variables, remove onCreate, remove companion object**. We don't need that in this case.



```
class MainFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_main, container, attachToRoot: false)
    }
}
```

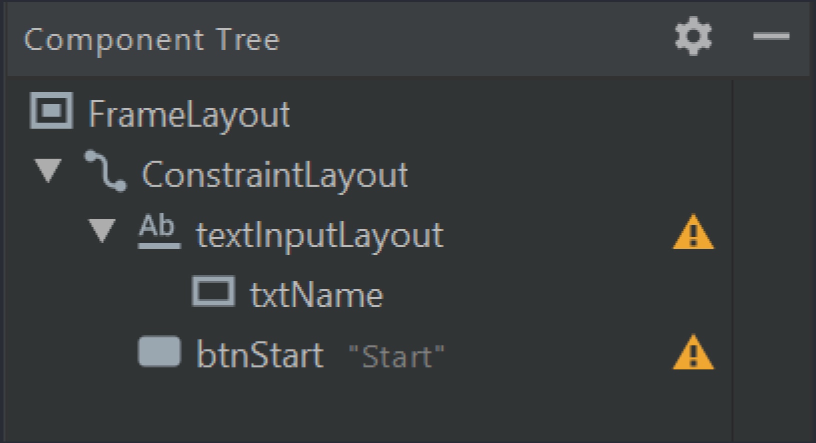
STEP 2: LAYOUT

Open fragment_main.xml, arrange UI like image below



editText inside textInputLayout

- Id = txtName
- Remove hint value

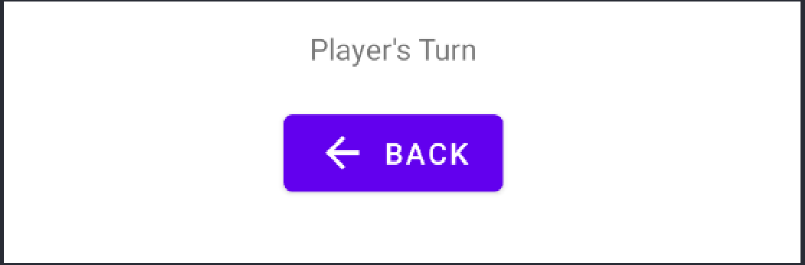


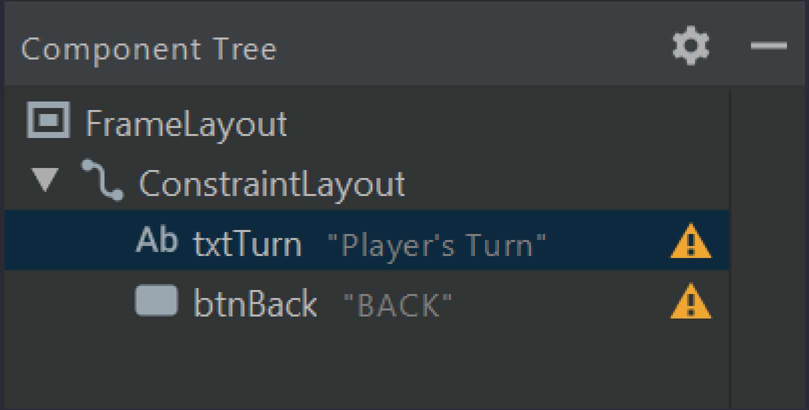
textInputLayout

- Hint = “Player Name”
- hintAnimationEnabled = true
- hintEnabled = true

STEP 2: LAYOUT

Open fragment_game.xml, arrange UI like image below





btnBack

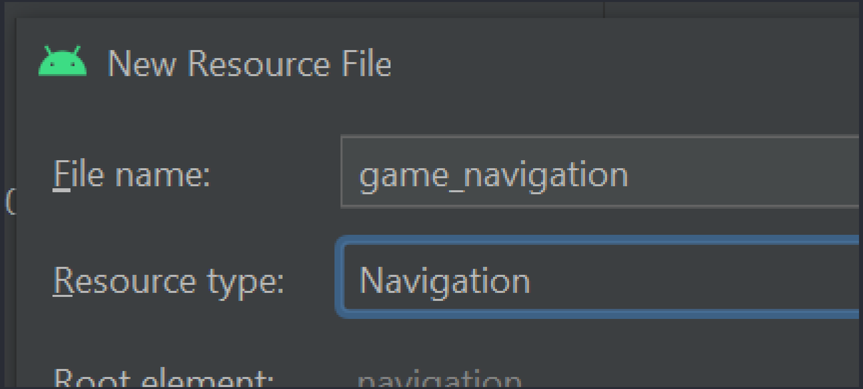
- Text = “back”
- Icon = use vector assets icon back arrow

STEP 3: NAVIGATION GRAPH

Navigation graph is a resource file. XML based, that represent how fragment navigate to other destination.

To create navigation graph:

1. Right click on res > new > android resource file
2. Enter file name to game_navigation
3. Change resource type to Navigation
4. Press OK



New Resource File

File name: game_navigation

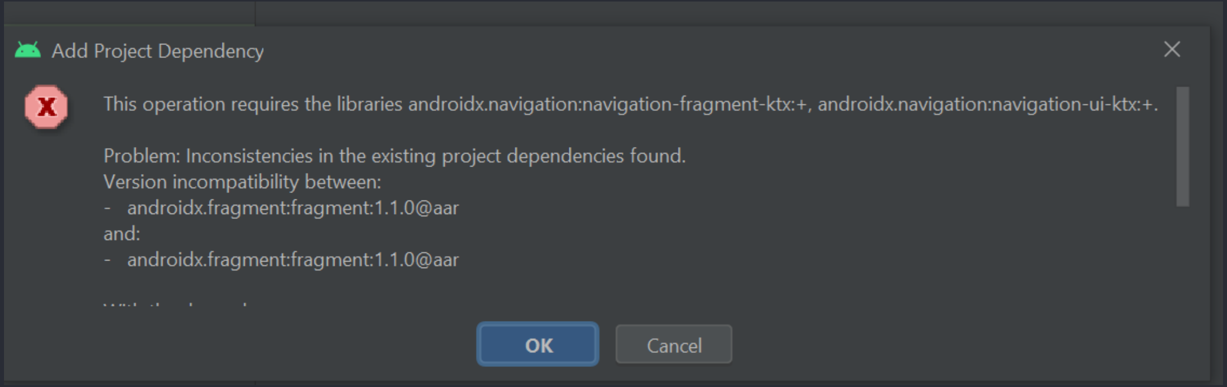
Resource type: Navigation

Root element: navigation

STEP 3: NAVIGATION GRAPH

Navigation graph needs additional dependency. This dialog will pop out and you can simply just press OK to continue. Wait until download and gradle process finished. **(notes: in recent android studio version, this dialog wont show up)**

If the navigation graph still show "infinte loading" indicator, you may restart the android studio to fix this.



This operation requires the libraries androidx.navigation:navigation-fragment-ktx:+, androidx.navigation:navigation-ui-ktx:+.

Problem: Inconsistencies in the existing project dependencies found.
Version incompatibility between:
- androidx.fragment:fragment:1.1.0@aar
and:
- androidx.fragment:fragment:1.1.0@aar

ADDITIONAL GRADLE SETTING

Open build.gradle (module), and look at the plugins object on top. Add following line of configuration:

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

Sadly, this handy feature is now deprecated...

This config enables the androidx feature to refer a view object from kotlin codes by its ID. Without this, you need to use findViewById() mechanism

INSTALLING NAVIGATION JETPACK

Still in same gradle file, locate dependencies, then add these (use the autocomplete to avoid errors):

```
implementation 'androidx.navigation:navigation-ui-ktx:2.5.3'
implementation 'androidx.navigation:navigation-fragment-ktx:2.5.3'
```

INSTALLING NAVIGATION SAFEARGS

Now open build.gradle (Project) file. At the topmost line (after the comment, but before plugins block), copy this block (get it from here):

```
buildscript {
    repositories {
        google()
    }
    dependencies {
        classpath "androidx.navigation:navigation-safe-args-gradle-plugin:2.5.3"
    }
}
```

Version number may change in future release, just autocomplete when necessary

INSTALLING NAVIGATION SAFEARGS

Go back to the other build.gradle file (the module one). Add this plugin:

```
id 'androidx.navigation.safeargs.kotlin’
```

Sync your Gradle (make sure your machine is connected to the Internet)

SAFEARGS

SafeArgs is a gradle plugin that allows you to enter information into the navigation graph about the arguments that you want to pass.

It then generates code for you that handles the tedious bits of creating a Bundle for those arguments and pulling those arguments out of the Bundle on the other side.

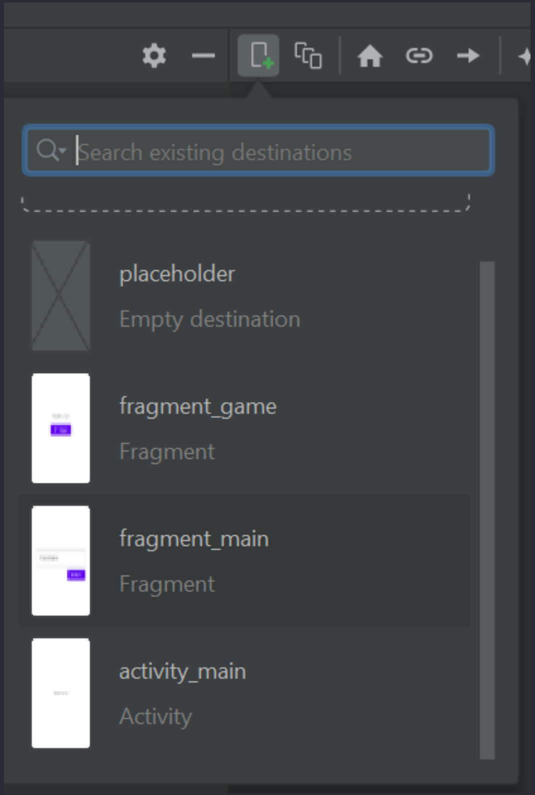
STEP 3: NAVIGATION GRAPH

Now we need to create navigation graph to define navigation in the app.

Start by clicking on **New destination** button

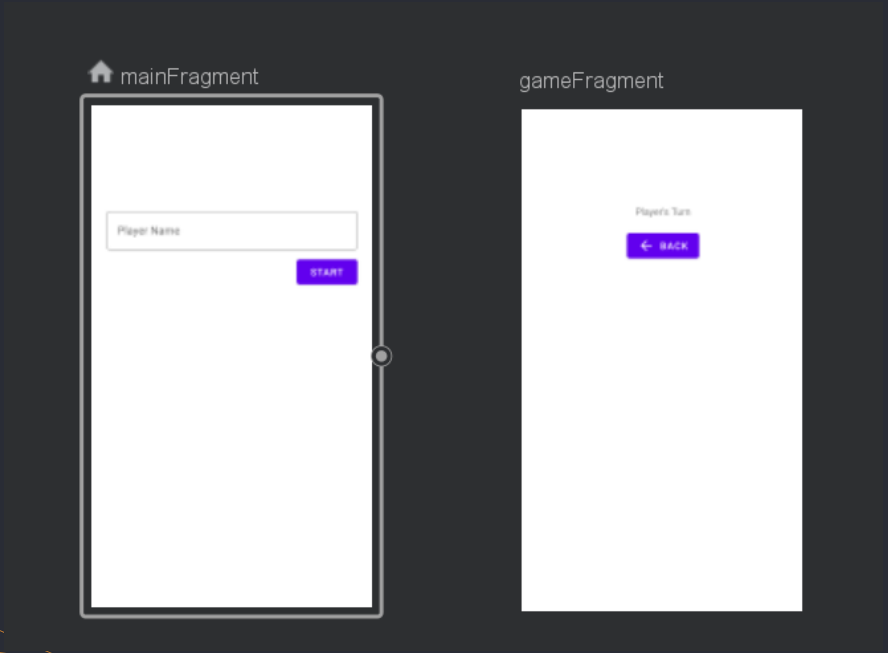
And then choose **fragment_main**

Repeat again and choose **fragment_game**



STEP 3: NAVIGATION GRAPH

Arrange the fragments like image below



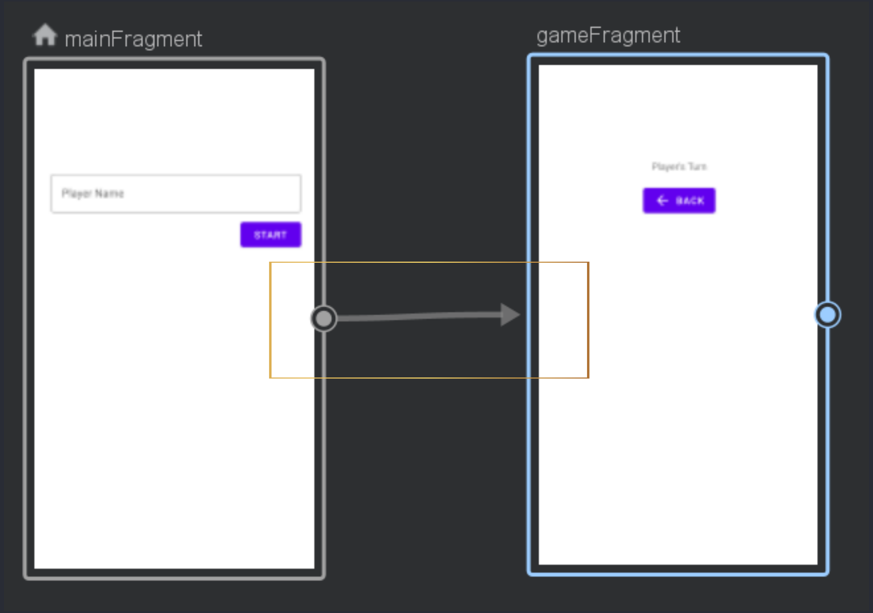
Click on mainFragment change label property to "Main Screen"

Click on gameFragment and then change label property to "Game Screen"

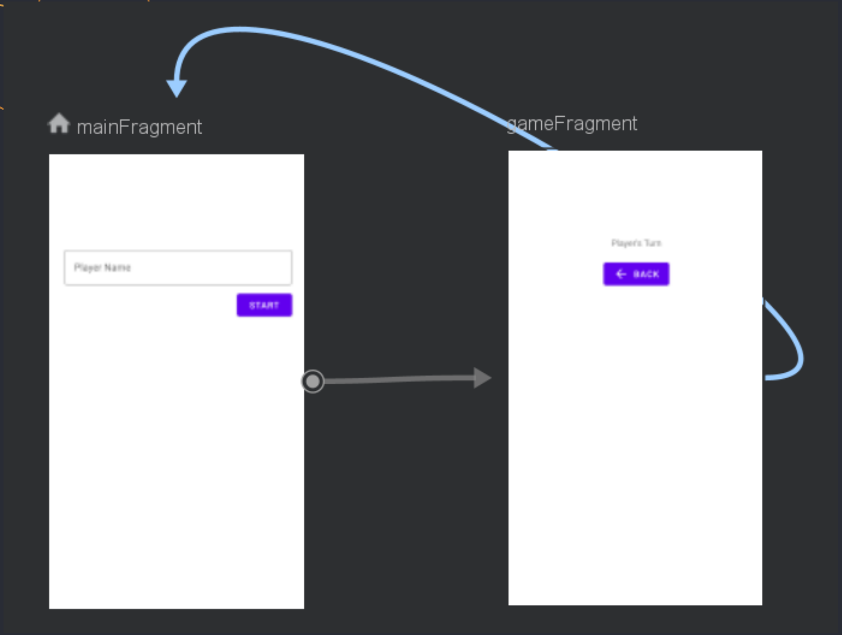
In the navigation graph, these two objects are called "destinations"

STEP 3: NAVIGATION GRAPH

Now we need to define the "actions" between destinations. Click and drag the small circle on the main fragment into the game fragment



STEP 3: NAVIGATION GRAPH

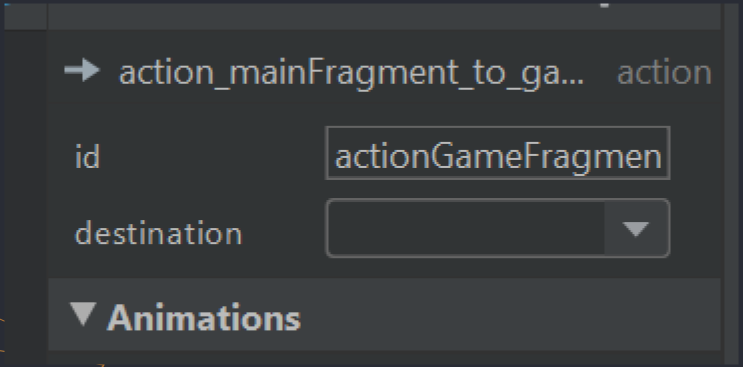


Do the opposite. Drag small circle from game fragment to main fragment

STEP 3: NAVIGATION GRAPH

Now click the arrow from main to game fragment. Rename the id to **actionGameFragment**

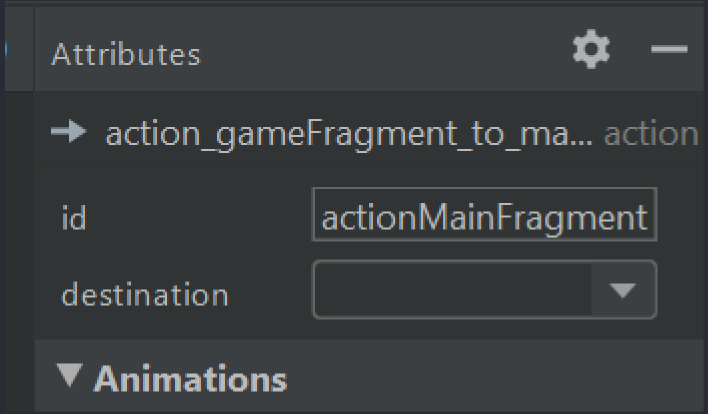
Continue to click on the arrow from game to main fragment, rename the id to **actionMainFragment**



action_mainFragment_to_ga... action

id: actionGameFragmen

destination:

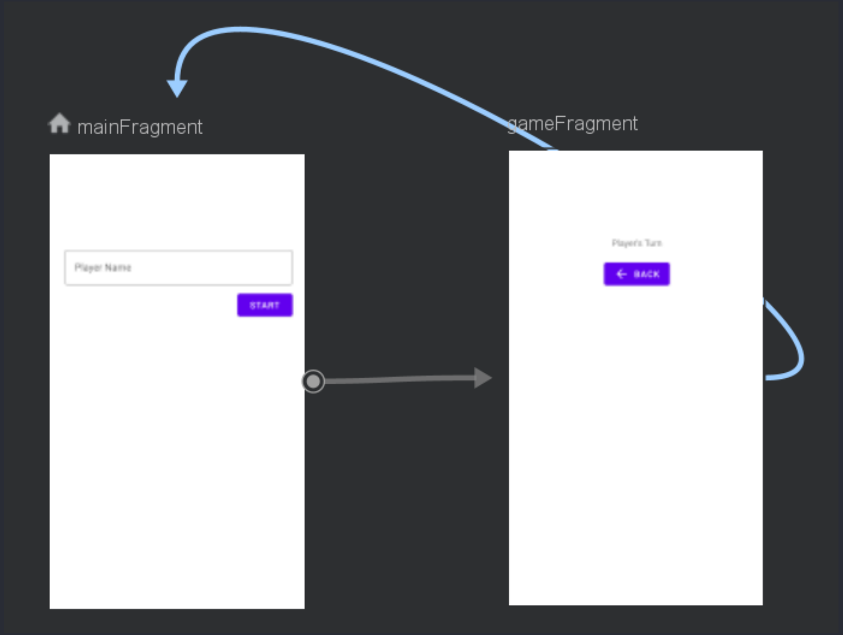


action_gameFragment_to_ma... action

id: actionMainFragment

destination:

STEP 3: NAVIGATION GRAPH



Those arrows are called "actions" to indicate navigation flow

IMPORTANT!
Every time you make a change on the navigation graph, you might need to **rebuild the project (build > rebuild project)**

It will trigger the **safeargs** library to generate necessary class needed in our kotlin codes

No need to rebuild on Bumblebee

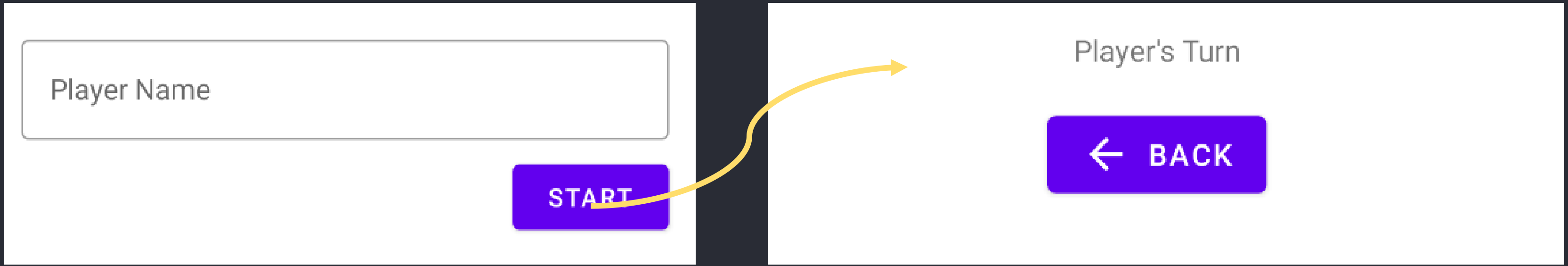
STEP 4: NAV HOST

Navigation graph must be implemented within a container, a **nav host**. To create nav host, open the activity_main.xml

1. Delete the default TextView
2. Drag and drop NavHostFragment component to screen
3. Choose **game_navigation** graph
4. Constrain it to all edges to make it full screen
5. Rename this navhost component id to **hostFragment**

STEP 5: CALL THE ACTIONS

The navigation graph only defines the navigation flows. The actions must be called manually in Kotlin codes. In this case, we want to trigger the action **actionGameFragment** when the Start button is pressed.



STEP 5: CALL THE ACTIONS

Open the MainFragment.kt. Override onViewCreated, and add btnStart click listener

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    btnStart.setOnClickListener {  }
}
```

onViewCreated will be called after onCreateView. This means we can safely access and modify the view component in this method.

STEP 5: CALL THE ACTIONS

Now write call actionGameFragment within the listener

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    btnStart.setOnClickListener {
        val action = MainFragmentDirections.actionGameFragment()
        Navigation.findNavController(it).navigate(action)
    }
}
```

MainFragmentDirections is class generated by safeargs. It holds all actions arrow that defined previously on navigation graph

To trigger navigation, simply call Navigation object and navigate method to the action

STEP 5: CALL THE ACTIONS

Now do the opposite from Game fragment to Main fragment.
Open the game fragment and write following codes.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    btnBack.setOnClickListener {
        val action = GameFragmentDirections.actionMainFragment()
        Navigation.findNavController(it).navigate(action)
    }
}
```

STEP 6: ANDROID BACK BUTTON

The navigation controller manages the transition and navigation flow within your app with a single activity multi fragments concept.

Next step to set up the default “soft” android back button that is usually located on the top left corner of the app.

This back button is used to access the previous item on the back stack.

STEP 6: ANDROID BACK BUTTON

Open MainActivity.kt, create navController variable

```
class MainActivity : AppCompatActivity() {
    private lateinit var navController: NavController
```

The lateinit keyword stands for late initialization. Lateinit comes very handy when a non-null initializer cannot be supplied in the constructor, but the developer is certain that the variable will not be null when accessing it, thus avoiding null checks when referencing it later.

KotlinDevelopment.com

STEP 6: ANDROID BACK BUTTON

Open MainActivity.kt, create navController variable

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    navController =
(supportFragmentManager.findFragmentById(R.id.hostFragment) as
NavHostFragment).navController
    NavigationUI.setupActionBarWithNavController(this, navController)
}
```

These codes are used to initialize the Navigation Controller and associate it with NavHost object.

And then setup the action bar with soft button functionality

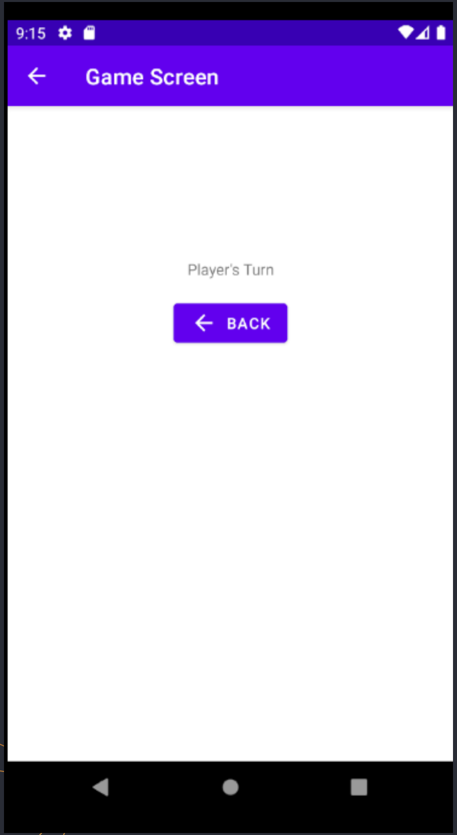
STEP 6: ANDROID BACK BUTTON

Next setup the action of this soft button

```
override fun onSupportNavigateUp(): Boolean {
    return navController.navigateUp()
}
```

You may replace codes with navigation drawer mechanism. Therefore instead go back to previous stack, app will show the menu drawer

RESULT



Notice that "soft" back button on the upper left of the screen. It will be displayed when needed.

If there is only one object on the back stack this button won't be shown.

STEP 7: CREATE ARGUMENTS

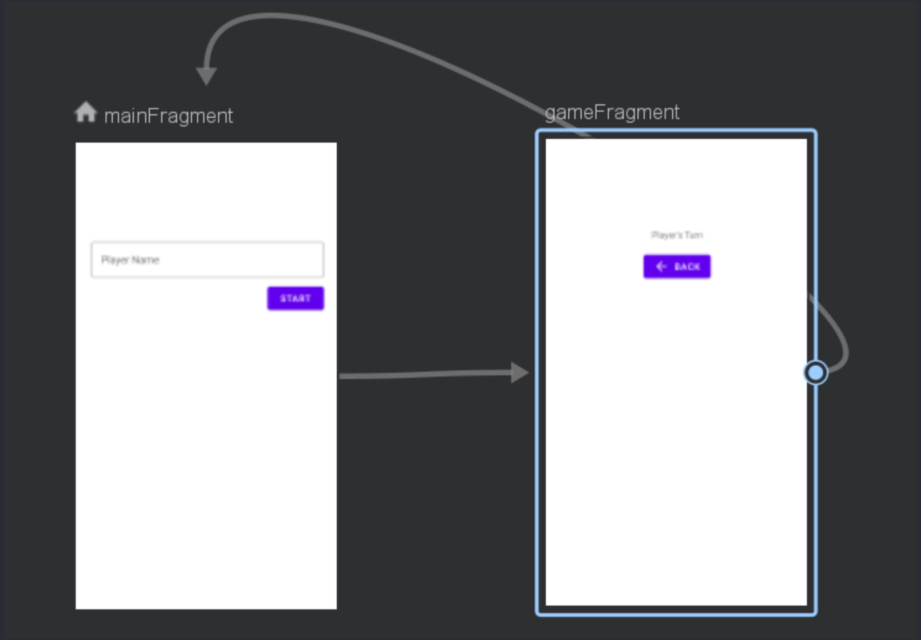
Sometimes we need to send an object/variables to destinations. Previously in Native course we can do this by using the intent extra (activity) or bundle injection (fragment).

In this new android Navigation jetpack, you need to define the argument inside the navigation graph.

Open the game_navigation to continue

STEP 7: CREATE ARGUMENTS

In the Navigation graph, the gameFragment expected to receive Player Name data from mainFragment, and displayed on the text view "Player's turn"

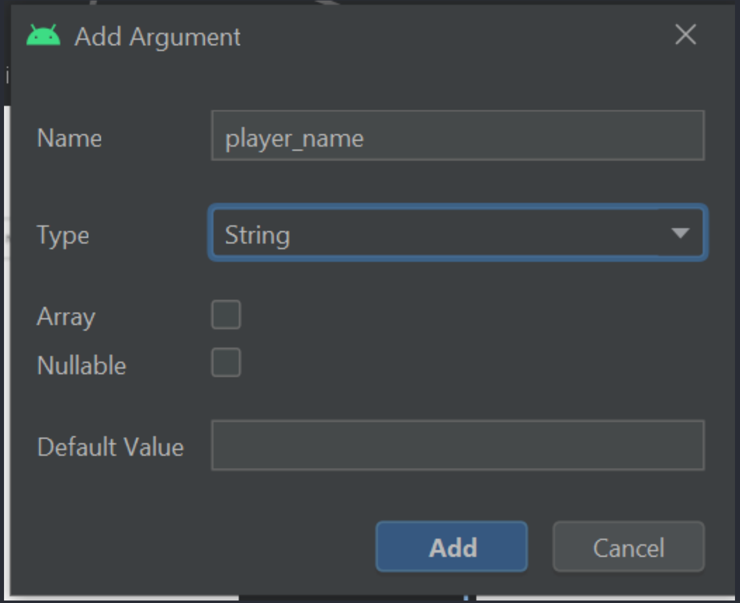


STEP 7: CREATE ARGUMENTS

Click on the game fragment, and click plus icon on the arguments property.

A dialog pop out:

- Set argument name as player_name
- Set type to String
- Press add
- Don't forget to rebuild the project (no need in latest android studio version)



Name: player_name
Type: String
Array: ☐
Nullable: ☐
Default Value:

STEP 7: CREATE ARGUMENTS

Open the MainFragment.kt, and add following codes on the btnStart listener

```
btnStart.setOnClickListener {
    val playerName = txtName.text.toString()
    val action = MainFragmentDirections.actionGameFragment(playerName)
    Navigation.findNavController(it).navigate(action)
}
```

Retrieve the player name data from EditText, and then use the action

If you have multiple arguments, you may use comma in action fragment constructor. Ie. actionGameFragment(playerName, playerScore)

STEP 7: CREATE ARGUMENTS

On the game frament side, we can receive the argument. Write following codes

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    if(arguments != null) {
        val playerName =
                GameFragmentArgs.fromBundle(requireArguments()).playerName
        txtTurn.text = "$playerName's Turn"
    }
```

GameFragmentArgs is another class created by the help of Safeargs

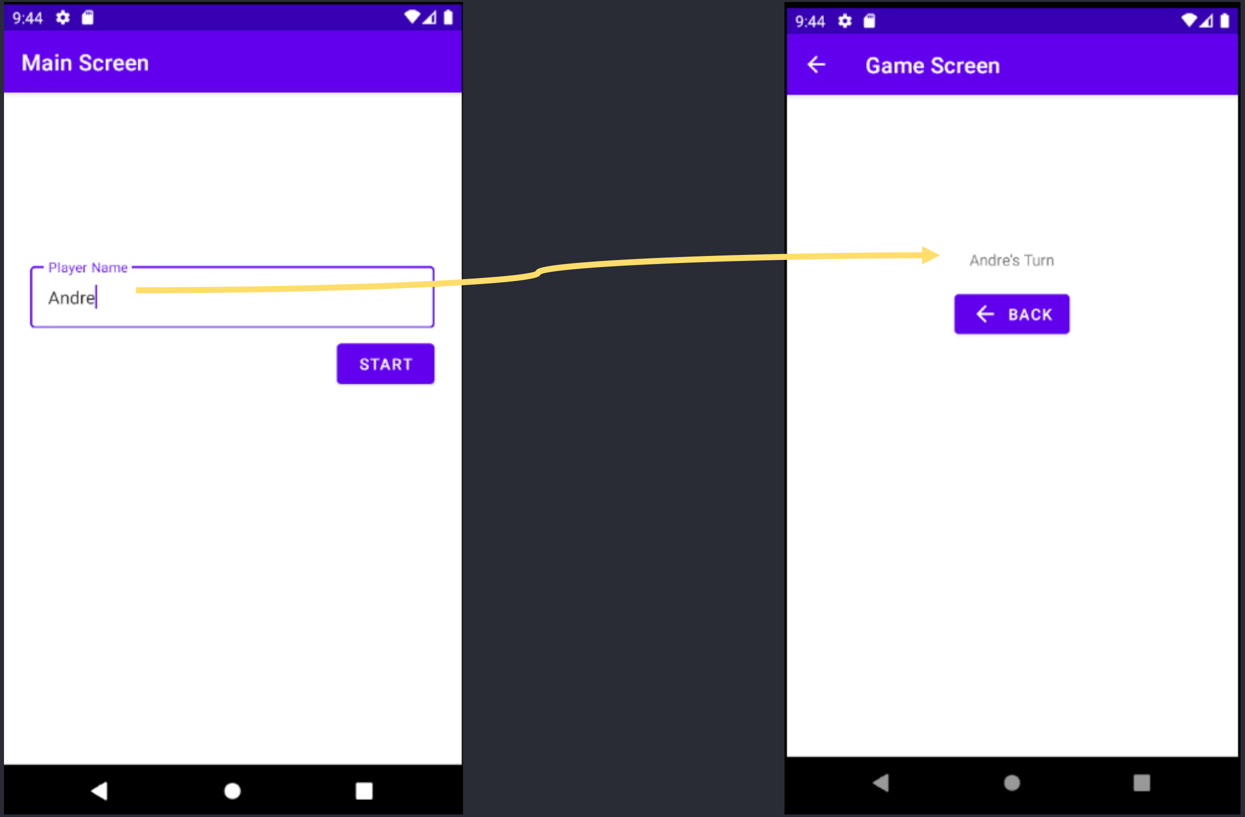
STEP 7: CREATE ARGUMENTS

Alternatively...

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    arguments?.let {
        val playerName =
                    GameFragmentArgs.fromBundle(requireArguments()).playerName
        txtTurn.text = "$playerName's Turn"
    }
```

You may use lambda **let** method.

RESULT


9:44
Main Screen
Player Name
Andre
START
9:44
Game Screen
Andre's Turn
BACK

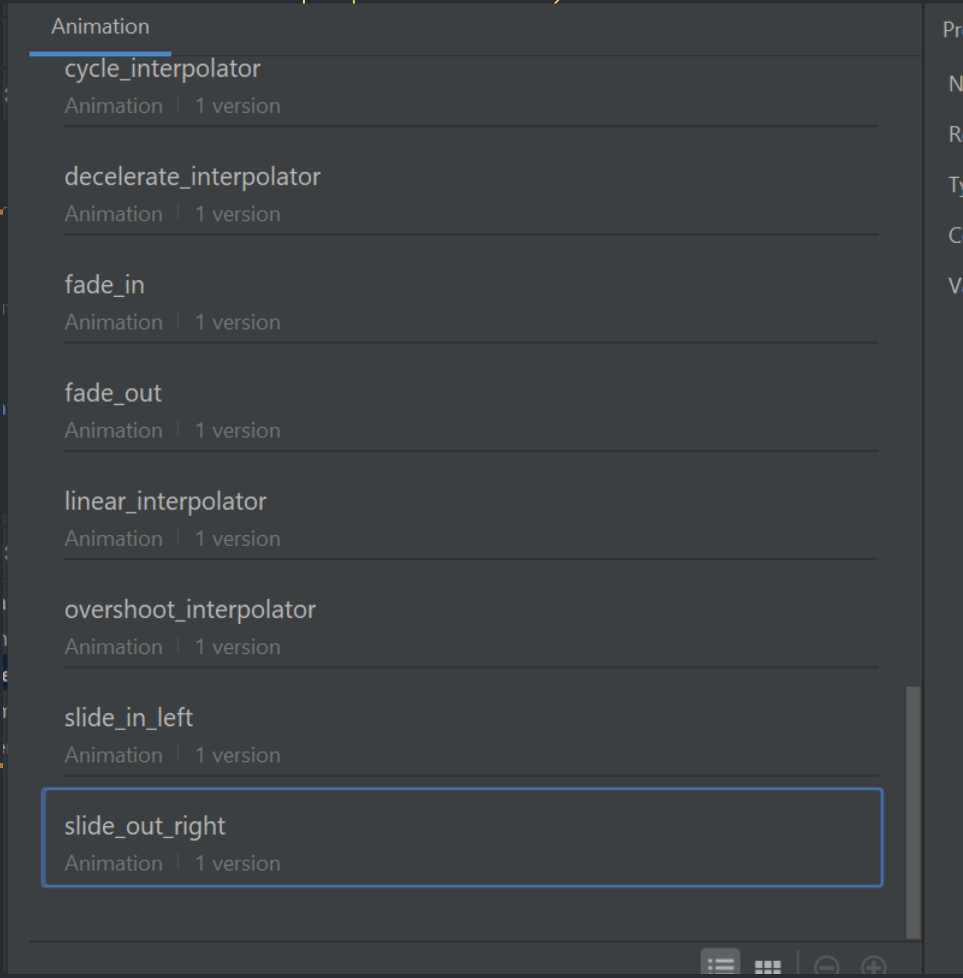
STEP 8: CHANGE TRANSITION ANIMATION

Android provide different set of transition animation effects. You can play with it in navigation graph.

1. Open the navigation graph
2. Click on the arrow action
3. Check on animation property
    a. enterAnim = animation when entering this fragment
    b. exitAnim = animation when exiting this fragment

ANIMATION OPTION

Try different animation. Find the best that you like



cycle_interpolator
Animation | 1 version

decelerate_interpolator
Animation | 1 version

fade_in
Animation | 1 version

fade_out
Animation | 1 version

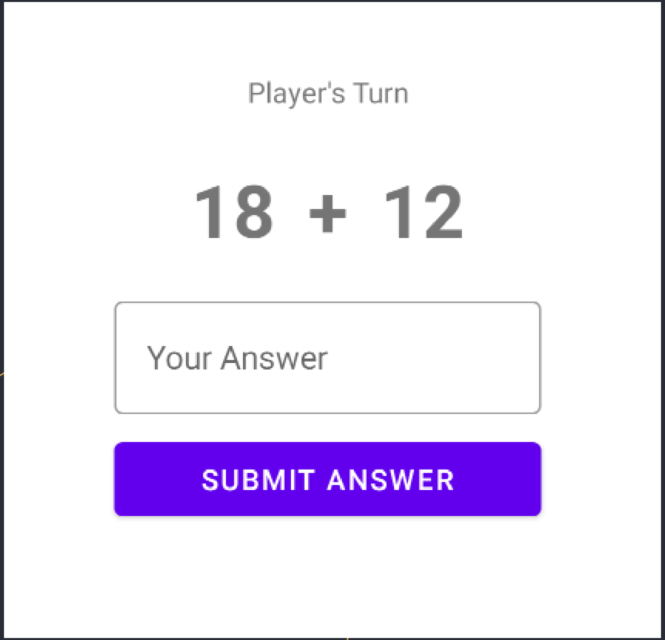
linear_interpolator
Animation | 1 version

overshoot_interpolator
Animation | 1 version

slide_in_left
Animation | 1 version

slide_out_right
Animation | 1 version

HOMEWORK



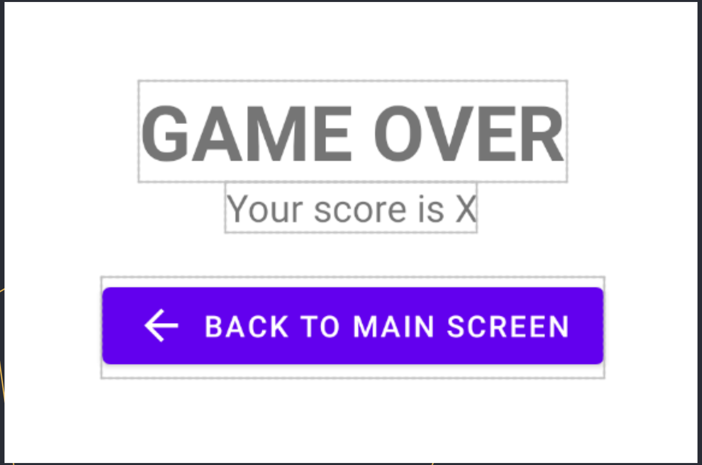
The Layout
Adjust the main fragment layout by using image on the left as reference.

The Logic
Game will randomize two number. Player's need to solve the addition equation.

The "Submit answer" button will check if the player's answer is correct or not.

If the answer is correct then the player gets 1 point. If the answer is not correct, then the game ends immediately and navigates to the result screen.

HOMEWORK (CONTINUE)



Create ResultFragment

The Layout
Use image on the left as references

The Logic
Display player score in “Your score” textview

Back to main screen button is used to navigate to main screen

DUE DATE

Submit your works next week. You need to write your github repo that contains this project to ULS

Good luck

THANKS!

DO YOU HAVE ANY QUESTION?

andre@staff.ubaya.ac.id