ChatGPT

Day 1: Setting Up NinJoy Android Launcher & Games (macOS, Multi-Module Project)

Android Studio Installation on macOS

1. **Download & Install Android Studio:** Visit the official Android Studio download page and get the latest DMG installer. Open the DMG file and **drag the Android Studio app into your Applications folder**, then launch Android Studio [1]. If prompted about previous settings, choose "Do not import settings" (for a fresh install) and proceed. Android Studio's Setup Wizard will run – select **Standard installation** and allow it to download the required Android SDK components [2].

2. **JDK Configuration:** Android Studio on macOS comes bundled with its own Java JDK, so you typically **do not need to install Java separately** [3]. The JDK is embedded within Android Studio's app bundle (accessible at `/Applications/Android Studio.app/Contents/jre/...`). The Setup Wizard will configure this automatically. After installation, you can verify JDK is working by opening **Android Studio > Preferences > Build, Execution, Deployment > Build Tools > Gradle** and confirming that **Use embedded JDK** is checked (default).

3. **SDK and Tools:** In Android Studio, open the **SDK Manager** (Tools > SDK Manager) and ensure the necessary SDK platforms and tools are installed. For this project, you can target a recent Android API (for example API 33 or 34). The default SDK from the installer should be fine, but verify that **Android SDK Platform-Tools** (which includes `adb`) is installed. (The wizard usually installs this by default.)

4. **AVD (Optional):** An emulator isn't strictly needed since you'll test on a Galaxy S5, but you can skip creating an Android Virtual Device for now to save time. Our focus is on the physical device deployment.

Enabling Developer Options on Galaxy S5 and ADB Connection

1. **Enable Developer Mode:** On the Galaxy S5, go to **Settings > About phone > Software information**. Find the **Build Number** and tap it **7 times** in a row. After a few taps you'll see a countdown message, and on the seventh tap it will confirm **"You are now a developer!"** [4] [5]. This unlocks the Developer Options menu in Settings.

2. **Enable USB Debugging:** Go back to **Settings**, scroll to **Developer Options** (it may appear under **Settings > System** on some devices). Enter Developer Options and **turn on the "USB Debugging" toggle** [6]. Confirm any prompts about allowing USB debugging. This setting allows your Mac's Android Studio/ADB to recognize the phone for installation and debugging.

3. **Connect Phone via USB:** Using a USB cable, connect the Galaxy S5 to your Mac. On the phone, you might see a popup **"Allow USB debugging?"** – check "Always allow from this computer" if prompted and tap **OK**. This grants your Mac's ADB connection permission to communicate with the device.

4. **Verify ADB Connection:** Open Terminal on macOS (you can use Android Studio's built-in terminal or the macOS Terminal app). Type the command: `adb devices`. You should see an output listing the device serial number with "device" status – for example:

```
List of devices attached
1234abcd    device
```

If you see "unauthorized", check the phone for the authorization prompt. If you see an empty list, ensure USB debugging is enabled and try `adb devices` again [7]. (On macOS, no separate USB driver is needed for ADB. **On Windows**, however, you must install the Samsung USB driver for the Galaxy S5 to be recognized [8].)

1. **Troubleshooting Tips:** If the device isn't showing, try another USB cable/port, make sure the phone is unlocked, and that the USB mode is MTP or File Transfer (not "Charge only"). Once `adb devices` lists your phone, you're ready to deploy apps to it.

Creating the NinJoy Launcher Project (Multi-Module Setup)

1. Start a New Project: In Android Studio, click **File > New > New Project**. Choose the **"Empty Activity"** template for a Phone/Tablet app. Name the application **"NinJoyLauncher"** (this will be the custom launcher app). For the package name, use something like `com.ninjoy.launcher` (you can adjust if needed). Set the minimum SDK to an appropriate level (e.g. API 21 or higher, depending on your device's Android version). Choose your preferred language (Kotlin or Java). Finish the wizard to create the project. Android Studio will create a new project with a default **app module** (often named "app") containing a `MainActivity` (or `LauncherActivity`) and a basic layout.

2. Configure Launcher Activity in Manifest: To make this app function as a home screen/launcher, you need to declare its main Activity as a launcher activity in the AndroidManifest. Open the app module's **AndroidManifest.xml** (under **app/src/main**). Inside the `<application>` tag, find or add the `<activity>` entry for the main launcher activity (Android Studio's template might name it `MainActivity`; you can rename it to **LauncherActivity** for clarity). Add an intent filter with **MAIN** action and **HOME** category, like so:

```
<activity android:name=".LauncherActivity"
          android:exported="true"
          android:label="NinJoy Launcher"
          android:screenOrientation="landscape">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.HOME" />
        <category android:name="android.intent.category.DEFAULT" />
```

```
    </intent-filter>
</activity>
```

This tells Android that our LauncherActivity is a Home Screen replacement. The `android:exported="true"` is required on Android 12+ since this activity has an intent filter. We also lock the orientation to landscape for this activity (using `android:screenOrientation="landscape"`) so that the launcher stays in landscape mode [9]. The critical part is adding the `HOME` category in the intent filter – this is what designates the app as a launcher/home app [10]. (Do **not** include the `LAUNCHER` category here – HOME replaces the usual LAUNCHER category for a default launcher app.)

3. Implement a Basic Launcher UI: Now, design a simple UI for the launcher. Open **res/layout/activity_launcher.xml** (or the layout tied to your LauncherActivity). For Day 1, keep it very simple. For example, you can use a vertical LinearLayout or ConstraintLayout containing a large **TextView** and a few **Button** views:

- A TextView at the top (maybe centered) that says **"NinJoy Home"** – this serves as a title.
- Three Buttons below it labeled **"Game 1"**, **"Game 2"**, and **"Game 3"**. These will eventually launch the respective games. For now, they can be dummy buttons (no onClick logic yet).

In XML, this could be structured as:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center">

    <TextView
        android:text="NinJoy Home"
        android:textSize="24sp"
        android:padding="16dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

    <Button
        android:id="@+id/btnGame1"
        android:text="Game 1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <!-- Similarly add btnGame2 and btnGame3 -->
    <Button
        android:id="@+id/btnGame2"
        android:text="Game 2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
```

```
        android:id="@+id/btnGame3"
        android:text="Game 3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

This will show a title and three vertically stacked buttons. (Feel free to use ConstraintLayout if comfortable, but LinearLayout is straightforward here.) Ensure the Activity's code uses this layout (the template should already call `setContentView(R.layout.activity_launcher)` in `onCreate`).

4. Add Game Modules (Multi-Module Structure): Instead of separate projects for each game, we'll use one Android Studio project with multiple **app modules** – one for the launcher and one for each game. This way, all code lives in one Git repository (easier to manage).

- **Create Module for Game1:** In Android Studio, right-click the project in the Project view (the top-level folder), choose **New > Module...**. In the module wizard, select **"Android App"** (sometimes shown as **"Phone & Tablet Module"**). For the module name, use **NinJoyGame1**. Choose a unique package name (e.g. `com.ninjoy.game1`). Select an Empty Activity template for this module as well (so it generates a basic Activity and layout). Finish the wizard to create the Game1 module. Android Studio will add it to your project (you should see a `game1` folder alongside `app` in the Project view). It will also update the **settings.gradle** to include `":game1"` module.

- **Configure Game1 Activity:** The Game1 module will have its own AndroidManifest.xml and a launcher activity (e.g. `Game1Activity`) with an intent filter for LAUNCHER (by default, new app modules include a launcher activity so the app appears in the app drawer). Open Game1's AndroidManifest (under game1/src/main). Change the label if you want (e.g. "Game 1"), and ensure it has `android:screenOrientation="landscape"` set for the Game1Activity (so that the game runs in landscape) [9]. In the Game1 layout XML (e.g. activity_game1.xml), put a simple placeholder UI – for instance, a TextView that says **"Game 1 – Coming Soon"** centered on the screen. This is just to confirm the app launches correctly.

- **Repeat for Game2 and Game3:** Add two more modules the same way (New > Module > Android App). Name them accordingly (e.g., **NinJoyGame2**, **NinJoyGame3**) with package IDs `com.ninjoy.game2`, `com.ninjoy.game3`. Each will have its own Activity (Game2Activity, Game3Activity). Set their orientation to landscape in the manifest and update their layout with a "Coming Soon" message. Now your project has four modules: one launcher and three game modules. Each module is an independent app with its own manifest, resources, and code. (Android Studio should have added all new modules to the project's settings.gradle automatically, registering them for the build [11] [12].)

Project Structure Note: In the Android view, you might see all modules (launcher and games) listed. The **launcher module (NinJoyLauncher)** is our custom home app, and the **game modules** are separate apps. They share no code yet (and for now, they don't reference each other). This modular setup will allow us to keep everything in one repo (ninjoy-android). You can build/run each module independently.

GitHub Repository Setup and Initial Commit

Now that the Android Studio project (with multiple modules) is set up, we'll initialize a Git repository and push it to GitHub for version control.

1. **Create GitHub Repo:** Log in to GitHub and create a new repository (e.g. named `ninjoy-android`). You can create it empty (no need for README or .gitignore via GitHub, since we'll add our own). Copy the GitHub repo URL (HTTPS or SSH).

2. **Initialize Git Locally:** Open a Terminal and navigate to your project's root directory (the directory containing the settings.gradle and app/build.gradle files). Run the following commands to initialize git, add files, and commit:

```
git init
git add .
git commit -m "Initial project setup: launcher and game modules"
```

Before committing, ensure you have a **.gitignore** file to avoid committing unnecessary files. Android Studio may include a basic .gitignore; if not, create one. At minimum, ignore build outputs and IDE files. For example, your .gitignore should include entries like:

```
*.apk
*.iml
.gradle/
/build/
/local.properties
/.idea/
.DS_Store
```

(These ignore the build directories, local config, and IDE metadata [13].)

1. **Push to GitHub:** Link the local repo to GitHub and push the code:

```
git remote add origin <your-github-repo-url.git>
git branch -M main   # (optional) rename default branch to main
git push -u origin main
```

This will upload the project to GitHub. You can verify on GitHub that your files (app, game1, game2, game3 modules, etc.) are present. Going forward, both Person A and Person B can pull this repo to collaborate.

Note: If you prefer, you could use Android Studio's built-in VCS → "Import into Version Control" → "Share Project on GitHub" feature, which automates the above steps. Under the hood it performs similar `git init`, commit, and push operations. Either method is fine, but the command-line approach is explicit [14].

Deploying and Testing on the Galaxy S5

With the environment ready and code pushed, let's test the launcher and games on the actual device.

1. **Build and Install NinJoyLauncher:** In Android Studio, select the **NinJoyLauncher app module** as the run configuration (usually there's a drop-down to choose which app to run). Connect the Galaxy S5 (if not already). Click the **Run** button (►). Android Studio will build the launcher APK and install it via ADB on the phone. Watch the Run console for any errors; if "Install succeeded", it's on the device.

2. **Switch Default Home to NinJoy:** After installing, press the **Home button** on the Galaxy S5. Since you now have a second launcher installed, Android will prompt you to choose which app to complete the action with – likely giving you a choice between the default TouchWiz/Samsung launcher and **NinJoy Launcher**. Select **NinJoy Launcher** from the list. If you get an option to set it **"Always"** or **"Just once"**, choose "Always" to make it the default home app. (This dialog is provided by Android OS whenever multiple apps declare the HOME category [15], so users can choose their default launcher.) After selecting NinJoy as default, you should see your custom launcher UI come up immediately.

3. **Verify Launcher UI:** The phone's screen should now display the content of your NinJoyLauncher app – e.g. the "NinJoy Home" text and the three placeholder buttons. This confirms that the launcher activity is running as the home screen. Try rotating the phone; it should stay in landscape (since we locked orientation).

4. **Install Game Apps:** Next, install the three game apps on the device. You can do this one by one from Android Studio:

5. In the run configuration drop-down, switch to the **Game1** module (Android Studio should list "NinJoyGame1" as an available app to run). Run it on the device. It will install and automatically launch Game1Activity on the phone, showing the "Game 1 – Coming Soon" placeholder screen.
6. Do the same for **Game2** and **Game3** modules. Each should launch and display its placeholder text. (Alternatively, you could build APKs and use `adb install`, but using the IDE is simpler.)

7. After installation, you may also see icons for Game1, Game2, Game3 in the phone's app drawer (since their manifest still has the default LAUNCHER category). That's fine for now. You can manually open each app from the app drawer to verify they work.

8. **Test the Launcher Behavior:** Press the Home button while in any of the games – it should immediately return you to the NinJoyLauncher home screen (since you set it as the default launcher). This is the desired behavior: our launcher is now acting as the central home screen. The placeholder buttons on the launcher don't have functionality yet, so tapping them won't do anything on Day 1 (we will later wire these to launch the game apps). The main goal was to ensure the launcher replaces the default home.

9. **USB Debugging Prompt:** While testing, keep an eye on the device for any "Allow USB debugging" prompts if you restart it or reconnect – always allow them so you don't lose the ADB connection mid-test.

Day 1 Checkpoint

By the end of Day 1, **Person B** has a fully set up Android development environment on macOS and has successfully built and deployed the initial versions of the NinJoy launcher and game apps:

- **Android Studio** is installed and configured with the SDK and embedded JDK on macOS.
- The **Galaxy S5** is in Developer Mode with USB Debugging enabled, and ADB recognizes the device [7].
- The **NinJoyLauncher app** is created and set as the default Home app on the device (verified by the custom home screen UI appearing). The launcher's orientation is locked to landscape for consistency.
- Three **game app modules** (NinJoyGame1, 2, 3) are created with placeholder activities. All are installed on the S5 and display "Coming Soon" screens when launched.
- A **GitHub repository** (e.g. *ninjoy-android*) is initialized, and the Android Studio project (with multiple modules) is committed and pushed, complete with an appropriate .gitignore for Android projects [13]. This allows collaboration and version tracking going forward.

Both the hardware and software sides are ready. Person A can now focus on microcontroller programming, and Person B can iterate on the Android apps. The foundation is laid: when the S5 powers up or Home is pressed, our **NinJoyLauncher** takes over [15], and we have stub apps for the games, setting the stage for further development in the coming days. Congratulations on completing Day 1's objectives!

Sources: Steps and configurations are based on Android Developer documentation and community guides for setting up Android Studio on macOS [1] [3], enabling developer options [6], and creating Android launcher apps [10]. Guidance on .gitignore and GitHub setup follows recommended practices for Android projects [13] [14].

---

[1] [2] Install Android Studio | Android Developers
https://developer.android.com/studio/install

[3] macos - Using JDK that is bundled inside Android Studio as JAVA_HOME on Mac - Stack Overflow
https://stackoverflow.com/questions/43211282/using-jdk-that-is-bundled-inside-android-studio-as-java-home-on-mac

[4] How do I turn on the Developer Options menu on my Samsung Galaxy device? | Samsung UK
https://www.samsung.com/uk/support/mobile-devices/how-do-i-turn-on-the-developer-options-menu-on-my-samsung-galaxy-device/

[5] [6] Configure on-device developer options | Android Studio | Android Developers
https://developer.android.com/studio/debug/dev-options

[7] android - Set up adb on Mac OS X - Stack Overflow
https://stackoverflow.com/questions/17901692/set-up-adb-on-mac-os-x

[8] Samsung Android USB Driver | Samsung Developer
https://developer.samsung.com/android-usb-driver

[9] Screen Orientations in Android with Examples - GeeksforGeeks
https://www.geeksforgeeks.org/android/screen-orientations-in-android-with-examples/

10 Custom Android Launcher — Why and How do I build one? | by Viktor Mukha | Paradox Cat Tech Hub | Medium
https://medium.com/paradox-cat-tech-hub/custom-android-launcher-why-and-how-do-i-build-one-6a1b3af89d43

11 12 Building Multiple Applications in a Multi-Module Android Project | by Prakash Ranjan | Medium
https://medium.com/@prakash_ranjan/building-multiple-applications-in-a-multi-module-android-project-4601b22560ae

13 git - What should be in my .gitignore for an Android Studio project? - Stack Overflow
https://stackoverflow.com/questions/16736856/what-should-be-in-my-gitignore-for-an-android-studio-project

14 git - Push existing project into Github - Stack Overflow
https://stackoverflow.com/questions/17291995/push-existing-project-into-github

15 android - How do you prompt the user to set the default home app - Stack Overflow
https://stackoverflow.com/questions/13525370/how-do-you-prompt-the-user-to-set-the-default-home-app