

Client Guide - Navigational Tool

How to add the Fry Building

We have tried to design the application to be as modular as possible - no application code will need modifying to add the Fry Building, and only minor additions would be needed to add further buildings.

In general, **Bold** text in this guide indicates something on-screen such as a button or menu item, and `monospace` text means a file name, path or contents.

Contents

[Getting Started](#)

[Project Setup](#)

[Project Overview](#)

[Running the Application](#)

[Buildings](#)

[Building File Structure Overview](#)

[Graph Files](#)

[.building](#)

[.locations](#)

[.paths](#)

[.links](#)

[Maps](#)

[Resolution](#)

[360° photos \(optional\)](#)

[Recap](#)

[Adding the Fry Building](#)

[The Builder](#)

[Example](#)

[Notes & Limitations](#)

[Debugging](#)

[Publishing](#)

[Appendices](#)

[A. Setting up the Android Emulator](#)

[B. Custom Builder Scale](#)

[C. Further Buildings](#)

Getting Started

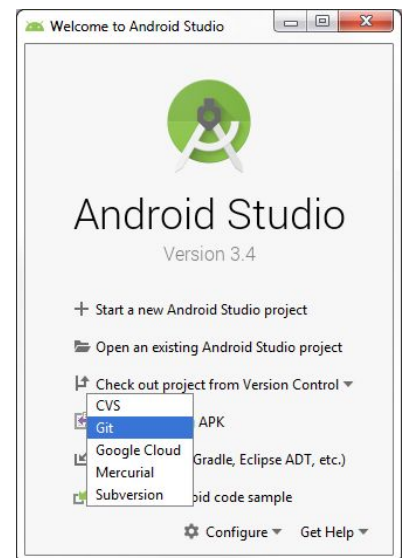
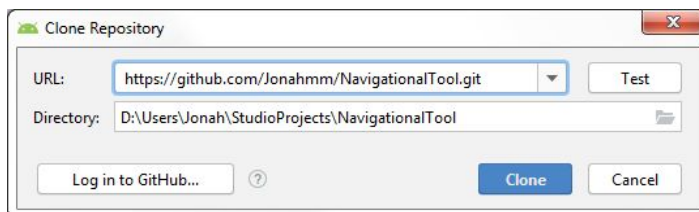
To edit the app you'll need:

- The latest version of [Android Studio](#)
 - If you haven't used an IDE before (or don't know what one is), you should read at least [this page](#) of the Android Studio documentation, and it's worth exploring more of the pages listed on the sidebar, especially in the **Meet Android Studio** section.
- An Android Device to test changes on
 - running Android 8.0 or newer
 - In developer mode with USB debugging enabled: [See here](#) for more information on setting up your device
- OR an emulated device - see [Appendix A](#)

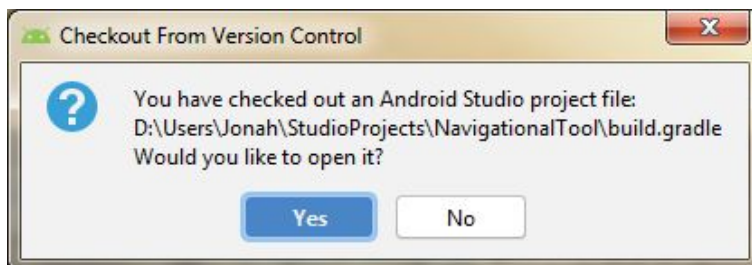
Project Setup

The project is hosted on [GitHub](#), and can be downloaded from within Android Studio.

- From the start screen shown on the right, choose **Check out project from Version Control**, then click **Git**.
 - This screen is usually shown when Android Studio loads. If yours instead loads a recent project, choose **File>New>Project from Version Control>Git**.
- On the subsequent screen asking for the URL, enter <https://github.com/Jonahmm/NavigationalTool.git> as illustrated below and click **Clone**. You *don't* need to log in to GitHub.

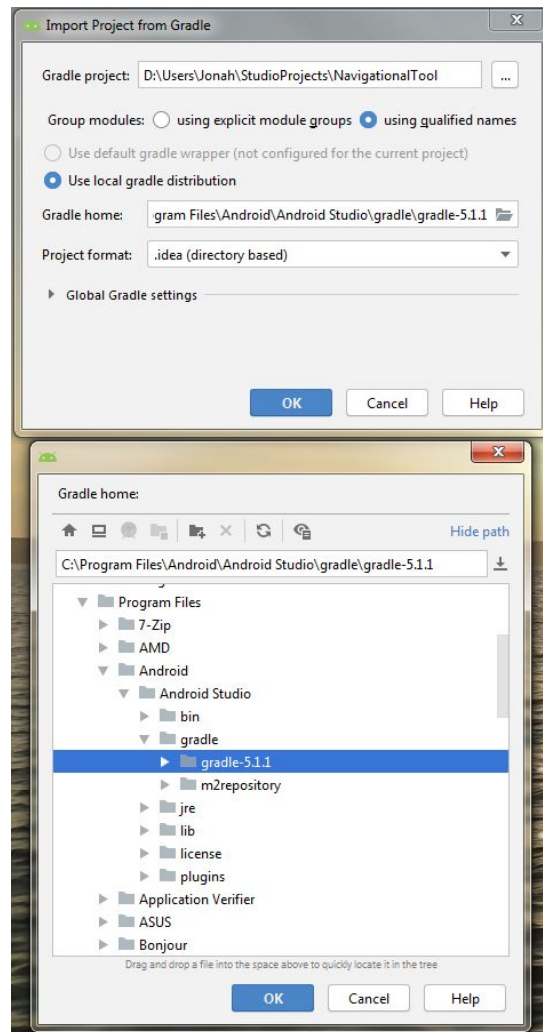


- Select **Yes** if asked if you would like to open the project file

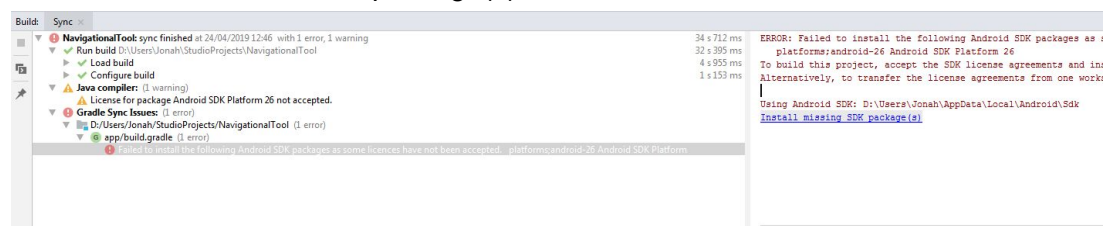


- Click **OK** on the next screen.

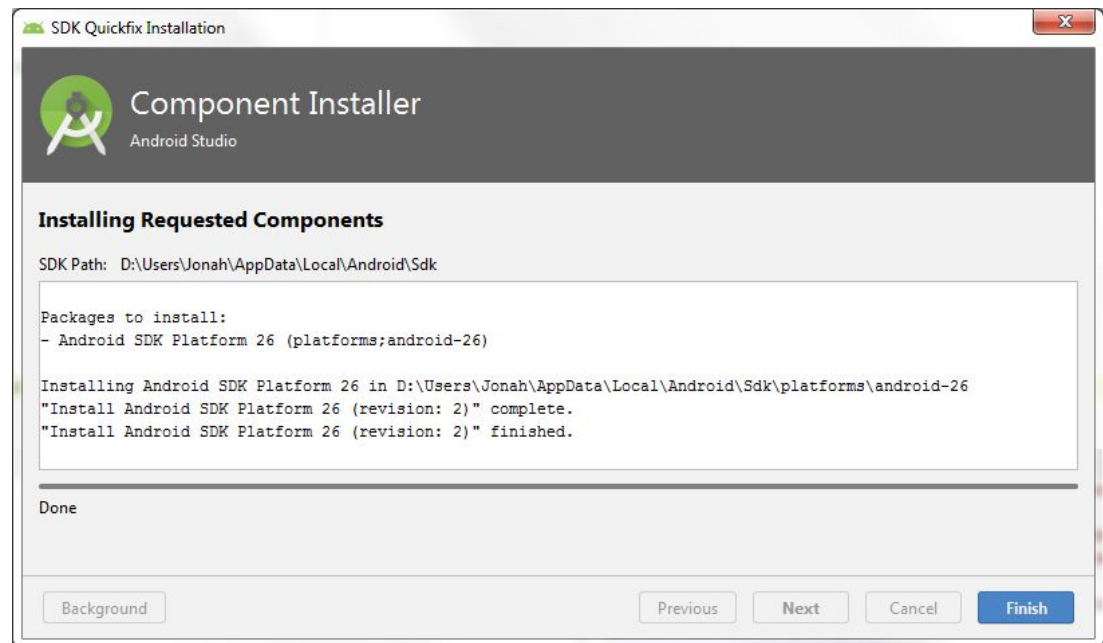
- If you get a popup that says “Gradle location not specified”, select the **gradle/gradle-X.Y.Z** folder found where you installed Android Studio. An example is shown below for Windows systems.



- The project should now be loaded. Android Studio will automatically ‘Sync’ (Install dependencies) and attempt to build the app.
 - If you get the below error about SDK Licences, select **Install Missing SDK package(s)** on the right side of the screen, accept the agreement(s), and wait for Android Studio to install the package(s).

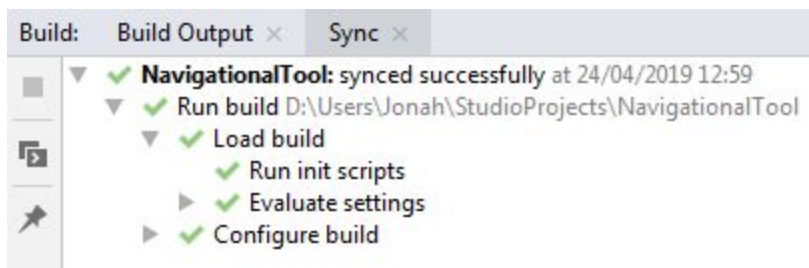


- Afterwards, you should see something like the below screen.



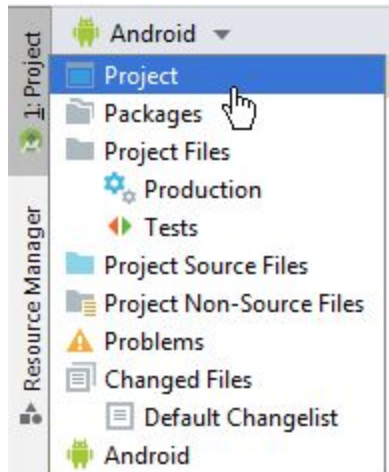
Click **Finish**, then click [Try Again](#) on the bar at the top of the editor. If it's not present, choose **File>Sync Project with Gradle Files**.

- All being well, you should see all processes succeed as below.



Project Overview

If it's not already open, open the **Project** pane in Android Studio, and select the **Project** view as shown below.



An overview of important files/folders:

`app` : This is the root directory for everything that eventually becomes the application. All source code, images, etc is in here.

`app/main/assets` : In this folder are the files used to construct and display buildings - see [Building Structure](#)

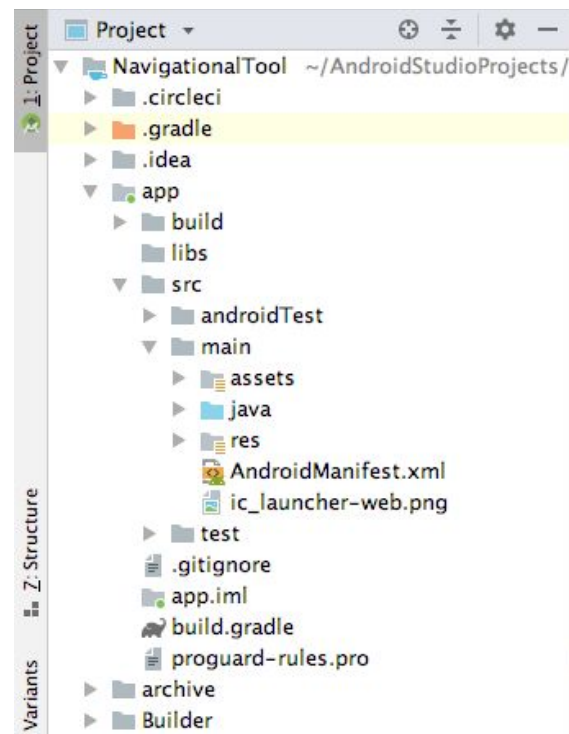
`app/main/java` : The Java source code for the application.

`app/main/res` : XML files for interface layouts, text values, drawables and more.

`app/main/AndroidManifest.xml` : Defines important properties of the application.

`app/build.gradle` : This defines the tasks and their prerequisites needed to create the app.

`Builder` : See [Builder](#)




The only folders you'll need to work with to add the Fry Building are the first and last as listed above, but it's worth knowing what the rest are for should you want to edit them in future.

Running the Application

To run the app:

- If you're using a physical Android device, connect it now. You should see a notification saying "USB debugging connected".
- Make sure that 'app' is selected in the drop down box in the top right of Android Studio



- Click the  button.
- Select your device from the list.
 - If you're going to use an emulated device and haven't set it up yet, click **Create New Virtual Device** then see [Appendix A](#) (skip step 1)
- If you're only going to be using one device, you might want to check **Use same selection for future launches** so you can bypass this menu in future.

Now wait for the launch to complete - you can see what's being done in the status bar at the bottom of Android Studio, with more details in the **Run** pane in the bottom left.

After a while, the app will pop up on the phone's screen (or, if something went wrong, the details will be in the **Run** pane). If the phone is locked while installing, it won't wake up automatically.

If you haven't used the app yet, now is a good time to familiarise yourself with it - get to know how to use the map, navigation system, search, 360° viewer etc as it will be helpful to relate what you do to how the app works if you're going to edit the buildings.

Buildings

Building File Structure Overview

The buildings' data and maps are stored in the `app/src/main/assets` subfolder. Each building has its own subfolder; currently only `maths` and `physics` exist.

As the physics building is included in the application, we'll use its content as an example.

Graph Files

Throughout these files, a line starting with a `#` symbol is a 'comment' and is ignored by the program. These are useful when making sense of the files. All files **MUST** end with an empty line, and no field may contain a comma, apart from the building name.

`.building`

This file contains the building name, followed by a list of floor identifiers (unique single characters) and their names, each pairing separated a comma. Each entry is on its own line, and one of these must start with a `*` symbol, which denotes the default floor. Take a look at `physics.building` for a full example.

There **must** be a `.paths` and a `.locations` file for each floor identified in the `.building` file. The file name must be the floor identifier - in the `physics` folder, you should see `0.locations`, `0.paths`, `1.locations` etc.

`.locations`

Every line in this file represents a **Location**. Take a look at this line from `physics/2.locations` for an example.

```
72,2.20,Computer Room,2,832,3712
```

Let's break this down;

- `72` is the location **ID**. This must be a non-negative integer and there must be no other location with the same ID on this floor.
 - The app's loader goes through floors one at a time, and will assign IDs to locations based on the highest ID previously allocated. To maximise efficiency, IDs in a file should ideally range between 0 and ((No. of locations in file) - 1), though this is not a requirement.
- `2.20` is the location **code**. This can contain numbers, symbols (no commas!), and UPPERCASE letters only.

- When the loader first encounters a code, it will mark this location as a 'principal' location, and subsequently loaded locations with the same code are added as 'children' of this, inheriting the name.
 - It can help to think of codes as identifying 'spaces', where locations identify points.
 - Codes that begin with 'L' or end with 'S' are treated as lifts and staircases respectively by our algorithm used to generate directions, so if possible avoid using these pre/suffixes for any codes other than these.
Similarly, codes for these locations should have the correct pre/suffix to be considered properly.
- `Computer Room` is the location's **name**. It can contain any character except a comma.
 - If the location's code has already been seen by the loader, the name will be ignored.
 - Take a look at any of the physics `.locations` files and you'll see a collection of locations at the end of the file with their name set to '#'. If you inspect the file, you'll see that each of these locations' codes has appeared previously, either in the same file or in that of another floor (whose identifier must appear before the current one in the `.building` file). See [here](#) for why we use the '#' names.
- `2` is the identifier of the floor that the location is on. It always matches the file name.
- `832,3712` are the (x,y respectively) coordinates of the location on the map. These must be integers.
 - X values should range from 0 to 4319
 - Y values should range from 0 to 7679
 - The app may function if a location is outside one or both of these ranges, but the location will be off-screen and may cause crashes when it is selected.

`.paths`

Each line in this file represents a **Path** between two **Locations**. The entry below is from `physics/0.paths` and represents the doorway between the rear foyer of the building and the central corridor, where students are not permitted except in the case of disabled students who need to cross the building.

```
# G.BF-G.CC
17,39,DISABLED_STUDENT STAFF DISABLED_STAFF
```

The first line is a comment added by us to make the file easier to explore, as codes are more reader-friendly than IDs. The second line is the actual entry;

- `38` and `29` are the IDs of the two locations linked by this path.
- The rest of the line is a space-separated list of keywords denoting who can use this path. In our example the `STUDENT` keyword is the only one not present, as all other users are allowed and able.
 - Staircases should have no `DISABLED...` entries

- Staff-only paths should have no ...STUDENT entries
- In the physics building, we have designated lifts as for disabled use only, to discourage their use by able-bodied people and avoid causing overuse.

.links

This file shares a name with the .building file and is of a similar format to a .paths file, with one notable exception. It is the final file to be loaded.

```
b:41,0:129,STUDENT STAFF
```

The above is taken from physics.links and represents part of the front staircase, between the basement and ground floors. Notice the different format in the first two fields: `b:41` means “the location on floor ‘b’ whose ID is 41”.

These four file types completely define a building’s structure.

Maps

For every floor defined in the .building file, there should be an associated map. Maps reside in the same folder as the data files and must be **.png** format. For a floor with identifier `i`, the associated map file must be named `mapi.png`.

Resolution

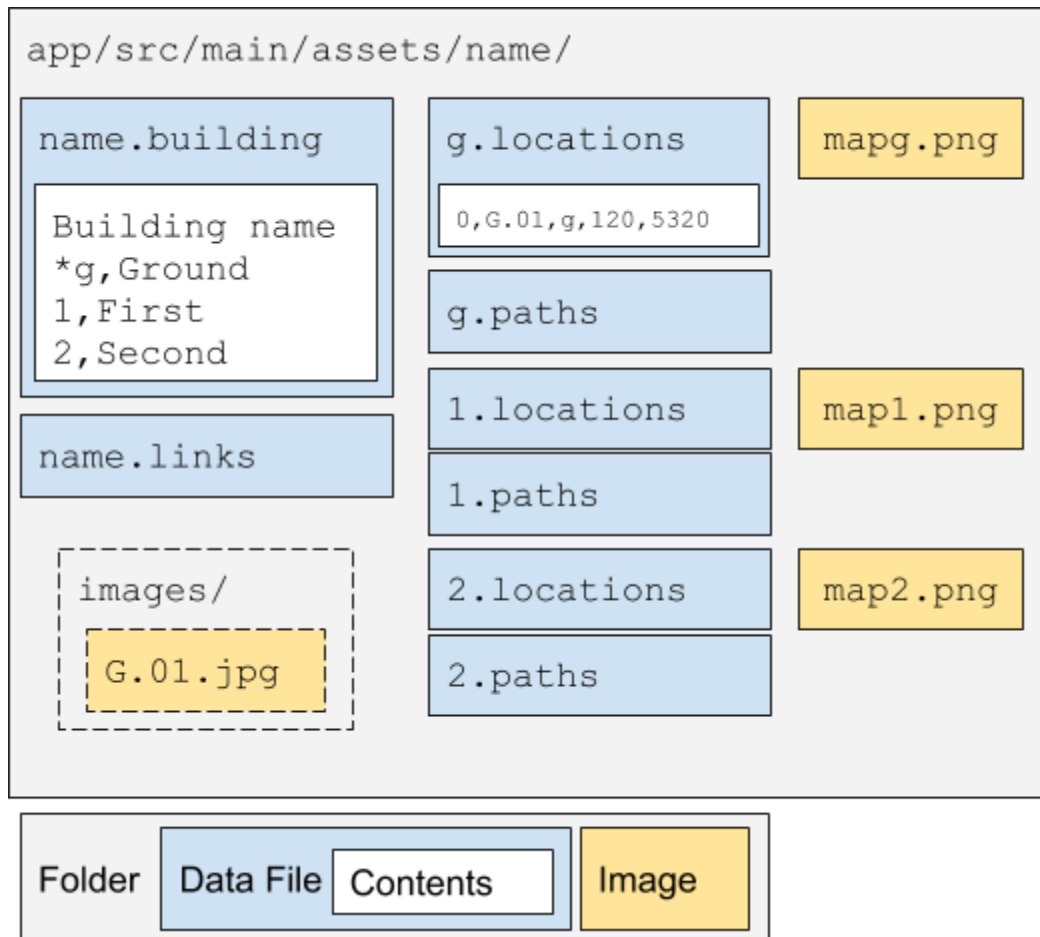
A map must have 9:16 aspect ratio, though we place no constraints on the actual resolution. We found during development that using full size 4320x7680 images resulted in bad performance due to the amount of processing needed for such a large image. We settled on a resolution of 2160x3840 for the final version, a ‘sweet spot’ between high resolution and performance. Using maps of too low resolution may result in pixelation when zoomed in.

360° photos (optional)

These are stored in the `images` subfolder and should be in **.jpg** format, named with the code of the room they show. For example, a 360° photo inside the Powell lecture theatre should be named `G.42.jpg`

Recap

A building's folder should have a structure similar to the example below.



Adding the Fry Building

In the `app/src/main/assets/maths/` folder, we've defined a placeholder building, named 'Fry Building' with only one floor, Example (Identifier `e`), whose map is just a capture from Google Earth. Also provided are `e.locations` and `e.paths`, which contain only roads, as well as an empty `maths.links` file.

The general process for creating the building structure is:

- For each floor `x`
 - Declare the floor and name it in `maths.building`
 - Create or obtain a 9:16 map and place in `app/src/main/assets/maths` folder
 - Populate an `x.locations` file with the locations
 - Populate an `x.paths` file with paths between the above
 - Add any cross-floor paths in `maths.links`

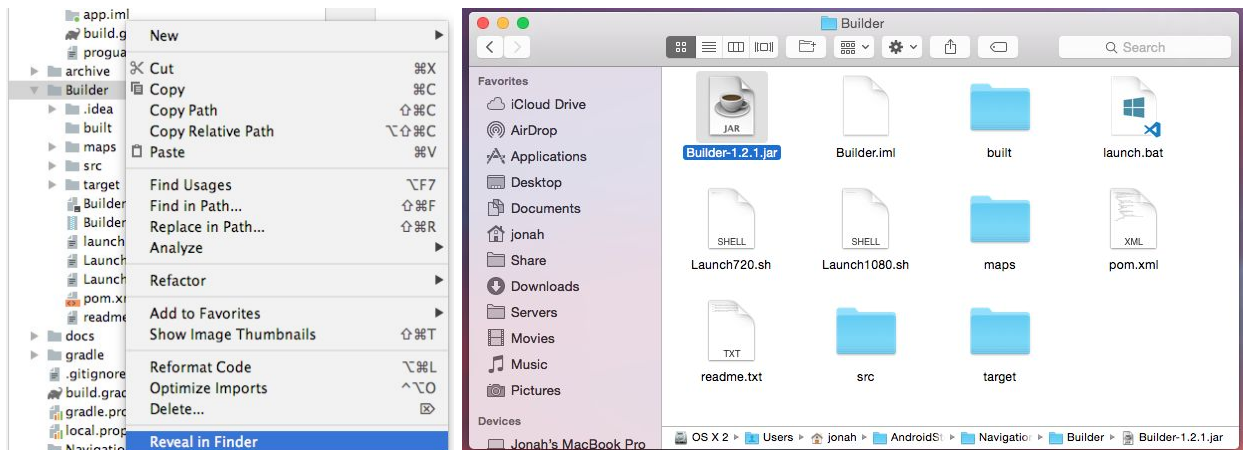
The Builder


Because creating the data files manually can be very tedious, we developed a tool, separate to the app, to allow easy creation of `.locations` and `.paths` files. It can be found in the `Builder` folder.

The builder requires [Java 8 or above](#) to run. Opening the folder in File Explorer (Windows) or Finder (Mac) (by right-clicking it in Android Studio and selecting **Show in Explorer** or **Reveal in Finder**) is recommended to allow easy file manipulation.

Example

The best way to explain how to use the builder is with an example; first I load the folder by selecting **Reveal in Finder** from the right-click menu in Android Studio (below-left). On Windows, this option is named **Show in Explorer**.

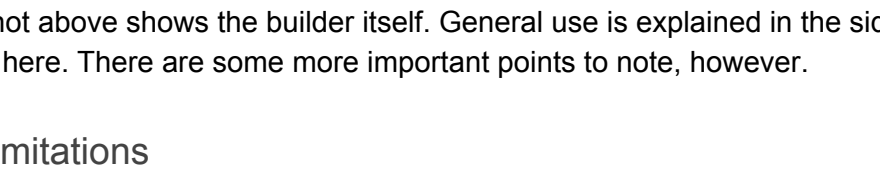




Graph Builder: Init

Enter floor string (e.g. b, 0, 1, m etc): 0

Start Builder



Notes & Limitations

are there to ensure that if the user taps a space, it will be selected. Otherwise, if the space only had location entries around its edges, where paths come in, the centre of the space might be outside the selection radius for any of the locations.

- **When using the builder, it is important to mark child locations by giving them the name '#'. For each code, there should only be one location whose name is not '#'. Without this, there is no guarantee that parent locations are exported before their children, and will be loaded in the correct order.**
 - The builder will warn against this when exporting.
- The builder works on files in the `Builder/built/` directory. It does not work directly on the files in `app/src/main/assets/...` to avoid data corruption.
 - When you're finished creating a floor, you can copy its `.locations` and `.paths` files to the building's folder in `.../assets`. Make sure that the floor has an entry in the `.building` file and that there is a valid map also present in the folder.
- Note that for this example, I've copied the `0.locations` and `0.paths` files from `app/.../physics` into `Builder/built`, and the builder has recognised this and loaded the locations and paths from the ground floor of the Physics Building.
 - If I hadn't copied these, the builder would have just loaded an empty map, ready for the addition of new locations.
- As you can see, the map is quite cramped with 158 locations on it, so 'Hide mode' as described in the sidebar is very useful when editing. If the builder is much smaller than your screen size, see [Appendix B](#).
- The builder can only work on one building at a time - it can help to think of the `built` folder as representing a building.
- The builder will always attempt to load a floor when it starts, but will **not** automatically save anything, so be sure to click the **Export** button before quitting if you want to save changes.
- As mentioned in the above section, the builder requires a map to run. It is recommended to use full size 4320x7680 maps when using the builder. This is because the app uses these scales as described [here](#). Even if your actual maps are smaller than this, it's a lot less effort to scale the maps up once (any image editor can do this) than it is to scale the locations every time you load and save in the builder.

Debugging

If you find that the app crashes upon loading the building, enable logging by uncommenting (removing the `//` from) line 57 in `app/src/main/java/.../Building.java`. This will provide a live output what locations and paths are being added, up to the point at which it crashes, which should help you identify the lines causing issues.

Publishing

For ways of releasing the app, including via a website or on the Play Store, see <https://developer.android.com/studio/publish>.

Appendices


A. Setting up the Android Emulator

This assumes you've already installed and opened Android Studio, and loaded the project. See [Project Setup](#) if you haven't done this.

Unfortunately, the emulator doesn't support computers that have non-Intel processors and are running Windows versions prior to the Windows 10 April 2018 update.

To install an emulator:

1. Open **AVD Manager** from the **Tools** menu and click **Create Virtual Device**.
2. Select a device and click **Next** - you'll want one with a high resolution display, **Pixel 2** is recommended.
3. Select a system image and click **Next** - you'll see the Android version numbers in the **Target** column. **Oreo** (8.0) is recommended for testing as it's the oldest version supported by the app.
 - You may have to download and install the image - click **Download** and Android Studio will guide you through the process. The download is likely to be quite large so a fast internet connection will be helpful.
4. Click **Finish**

If you want to, you can now start the emulator with the  button in AVD Manager.

B. Custom Builder Scale

By default, the builder loads images at 1/8th of their original resolution. This means that when working on a full-size 4320x7680 image as recommended, the map occupies a 540x960 area on screen. This works well on displays with resolutions up to 1920x1080, but many modern computers have significantly higher resolutions.

Read `readme.txt` in the `Builder` folder to see some examples of how to launch the builder with various scales. If on Windows, try `launch.bat` for a 1/6 scale (720x1280), or on Mac/Linux try running `launch720.sh` or `launch1080.sh` for scales of 1/6 or 1/4 respectively.

If you want to use a different scale, you'll have to start the builder from the command line. Start a Command Prompt / Terminal window and navigate to the Builder folder. (The easiest way to do this is usually to type "`cd`" (without quotes), then drag the Builder folder into the window from File Explorer/Finder and press enter.

To launch, run one of the commands below.

Windows only:

```
start launch.bat x  
Or  
launch.bat x
```

All operating systems:

```
java -jar Builder-1.2.2.jar x
```

Where x is a positive integer denoting the denominator used when scaling the map, i.e. running `launch.bat 5` will launch the builder at 1/5 scale.

C. Further Buildings

The process to add a third building is slightly more involved than editing the Fry placeholder, but is not much more complicated. *It does, however, involve editing some of the Java source code*, so can be daunting if you haven't worked with code before.

The first step is to create and add the building in the `app/src/main/assets/` folder. It must have a name and at least one floor (including one marked as default), as well as `.locations` and `.paths` files and a map for each floor, and a `.links` file.

Once this is done, there are two files that need editing:

First, open `app/src/main/menu/activity_display_drawer_drawer.xml` in Android Studio. It should bring up an editor. Drag **Menu Item** from the **Palette** into the **Component Tree**, making sure that it appears in the `building_picker` group.

In the **Attributes** pane, give the item a suitable ID. For this guide, we'll imagine the addition of the Life Sciences building, so I'll use `building_lifesci`. Set the **Text** field to the name of the building

`app/src/main/java/uk/ac/bris/cs/spe/navigationaltool/MapActivity.java` in Android Studio. To help navigate around the file, open the **Structure** pane from the **View>Tool Windows** menu. In this pane, scroll down until you see the three methods below.

```
m intTold(int): int  
m idToInt(int): int  
m loadFromIndex(int): String
```

Click any of these and the editor will scroll to the definitions, shown top left on the next page.. You'll see that they all have the same basic structure; in fact, they all perform slightly different operations on the same data.


```

private int intToId(int in) {
    switch (in) {
        case 0: return R.id.building_physics;
        case 1: return R.id.building_maths;

        default: return R.id.building_physics;
    }
}

private int idToInt(int id) {
    switch (id) {
        case R.id.building_physics: return 0;
        case R.id.building_maths : return 1;

        default: return 0;
    }
}

private String loadFromIndex(int in) {
    switch (in) {
        case 0: return "physics/physics";
        case 1: return "maths/math";

        default: return "physics/physics";
    }
}

```

The purpose of these functions is to provide some sort of mapping between the menu items and their respective buildings. However, we can't directly map the menu item IDs to the building paths, as the app needs to remember which building to load, and the values of the IDs may change between builds.

Assuming that my Life Sciences building data is stored in `.../assets/lifesci`, and that the `.building` and `.links` files are named `lifesci`, see the lower left example for how I would integrate the building into the app; the value 2 now corresponds to both the ID of the building selector (`idToInt` and `intToId`) and the path to the `.building` file (`loadFromIndex`).

```

private int intToId(int in) {
    switch (in) {
        case 0: return R.id.building_physics;
        case 1: return R.id.building_maths;
        case 2: return R.id.building_lifesci;
        default: return R.id.building_physics;
    }
}

private int idToInt(int id) {
    switch (id) {
        case R.id.building_physics: return 0;
        case R.id.building_maths : return 1;
        case R.id.building_lifesci: return 2;
        default: return 0;
    }
}

private String loadFromIndex(int in) {
    switch (in) {
        case 0: return "physics/physics";
        case 1: return "maths/math";
        case 2: return "lifesci/lifesci";
        default: return "physics/physics";
    }
}

```

D. Further Resources

If you are familiar with Android and Java, and want to further modify the app's functionality, see the technical documentation at <https://jonahmm.github.io/NavigationalTool/>.