GPS Drawing 1:
Android
Mobile and Ubiquitous Games
ICS 163
Donald J. Patterson
Issue #1

- You must install the Play SDK to get access to location
- The details are in Lecture 08’s notes
- You must import that project as code
- Then you must add the project as an *Android* library which is different than the normal Java library in the build path
• Issue #2

• For the Google Play SDK to work there must be a Google account associated with the phone
Those are both reasonable designs, but more than we are shooting for for this assignment.

Remember, you aren’t turning in your client.

The visualization will be graded and evaluated on the server side.

You can will be able to see your drawing in near real-time on the server.
GPS Drawing

• Should it be like this?
GPS Drawing

- Should it be like this?
• Issue #3

• Read the assignment it will answer many questions for you.

• Consider what the UI should be like after reading the assignment
So something more like this is in order:

- One view, Android native widgets, basic callbacks
GPS Drawing

- So something more like this is in order:
  - One view, Android native widgets, basic callbacks
  - A place to type in group id
  - A place to type in drawing id
  - A location feedback area
  - A place to toggle the pen
  - A place to select color
  - A feedback area showing the buffer fullness
  - A button to upload the data
GPS Drawing

- Threading
  - Android is a multi-threaded platform
  - There is only one thread that is allowed to update the U/I
    - You have to make sure that your code is being run by the U/I thread to change the U/I, or Exception
  - You have to make sure that long-running processes don’t take the U/I thread’s time, or Android will shut down.
• Threading
  • Three ways to manage threading:
    • runOnUIThread() construct
    • AsyncTask construct
    • low-level message passing
  • (more to come)
AsyncTask construct

For running a task that takes a long time and still being able to update the UI thread

Android runs each section on the right thread

4 hooks, each optional

- `onPreExecute()`, runs on the UI thread
- `doInBackground()`, runs on a non-UI thread
- `onProgressUpdate()`, runs on UI thread
- `onPostExecute()`, runs on UI thread

Example and usage: click below

GPS Drawing

- Actually Making the U/I
- The U/I is defined in XML
- Can be manipulated graphically
- Supports internationalization out of the box
- Supports multi-resolution displays out of the box
- Supports multi-sized displays out of the box
- Is complex as a result
• Fragments
  • Enable one code base to work on many different sized devices
  • Each fragment is like a small app with it’s own lifecycle
  • Big displays will run multiple fragments at once
  • Small displays will run fewer at once
GPS Drawing

- Fragments
  - They are complicated
  - For our project we are just going to have Activity as a shell for one Fragment
  - We are going to turn off device rotation
  - We will treat the Fragment as a single U/I
Buttons

How do we get control when someone clicks a button?

- Statically via the XML U/I description
- Dynamically via code

Here is a project you can download with examples:

The static button is assigned a callback function in the XML U/I.
• The dynamic button is assigned a callback function in the code.

• First the container is created in `onCreate`

• Then the fragment is built and put in the container

• Then the fragment is built in `onCreateView`, the button is located dynamically, and an anonymous function is assigned to the button that is used as the callback function
```java
/**
 * A placeholder fragment containing a simple view.
 */

public static class PlaceholderFragment extends Fragment {

    public PlaceholderFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_miles_is_awesome, container, false);

        Button updateButton = (Button) rootView.findViewById(R.id.button2);
        updateButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                dynamicClick(v);
            }
        });
    
    return rootView;
    }

    private void dynamicClick(View v) {
    
    }
}
```
So, that should be enough to make a UI

- With buttons
- That respond to being clicked

Next

- Location
  - Updating UI
- Uploading Data