**Developer List:** Austin Chang (asc7kc), Nathan Blaine (ngb2zf)

**Device Name/Platform:** Unown / Android

**Project Title:** UVAnywhere

**Project Pitch:**

The purpose of this app is to provide a way for students to check the line/crowds at their favorite dining option/study space/gym. The app allows users to post pictures of certain areas (say, for instance, the line outside of the Einstein Bagel's at Rice) for other users to view and get a sense of what the crowd or line is like at that area at a specific point in time. Location services will be used to verify that users of the app are near a supported location, which will then give them an opportunity to post a picture of that location. Once the image is captured, it is uploaded to a cloud webservice, which will then update the user's feed for that location within one minute.

**Platform Justification:**

One logistical reasoning for choosing the Android platform was that one partner did not have full-access (aside from lab) to an Apple computer. This would have made development difficult because we would have always had to meet in a lab and only when we had access to them.

Another reason for picking Android was the hope that Google APIs for location would work easier, although as we found out permissions seemed to be the biggest hurdle when it came to using location API.

**Key Features:**

**View Locations** The main purpose of the application is to view the crowds/people presence at the various locations. This is presented to the user on the main screen of the app, and updates live by querying the cloud hosting API Cloudinary.

**Post Image** When a user is within a certain radius of a specified location and they click the "Post" button, they are prompted to take a picture, which is then uploaded to the cloud for others to see on the live feed. This keeps the image of the crowd up to date and is the other core feature of the app.

**Detail View** Clicking the image from the main feed brings up an enlarged "detail view" of the location with information on hours of operation/description and the most up to date photo enlarged.

**Settings Page** Allows the user to modify their username and change themes of the app.

**Testing:**

Testing involved walking to the supported locations (whose lat and long were hardcoded into the application) and seeing if we were able to post. This was the most obvious and tangible feature to test, as making sure only relevant information is being posted to the feed (i.e. within a certain radius of X place.) This allowed us to settle on a 60m radius based on the GoogleMaps address of a location, which we felt was fine enough to filter out posters out of the area but still broad enough to not filter out people in the area but out of range of the coordinates from GoogleMaps.

**Usage:**

- The radius is set to 60m around the lat/long of the supported locations. The reason this is so large is because the tablet is using WiFi to get the gps coordinates, so it can be a bit fudgy. The best place to demo it is Rice hall since the building is narrow enough where you are always using the routers in the building. Clemons is also working pretty consistently, though sometimes the location takes you out by Mem Gym. so see below to check where the location services think you are actually at.

- Use the GoogleMaps app on the tablet to see where its location is putting you. This was how we were checking if it was how we were handling locations or just getting bad locations because of WiFi.

**Lessons Learned:**

One of the most frustrating lessons learned was how to handle permissions for target SDKs > API 22. Android made a big change by switching to checking permissions at runtime rather than at install, so our work-a-round was to have a dummy activity be displayed after the user approves the permission, essentially gating our app behind a location services requirement. The change that this happened in V.23 of the app was well documented, but it was difficult to know that we had to rely on overriding the onRequestPermissionsResult() method was not very well published (and caused a lot of headache!).

Another learning point was that, while the external libraries from Android and Google can be extremely powerful and useful, it is sometimes easier to just write a "dumbed down" or more simple version of their utilities. For instance, we initially were trying to set our radius around a lat and long using Google's geofence library. GeoFence, while having all of the characteristics that we were looking for, also had a lot of other overhead that became very complex to work with very quickly. We solved this by simply making a Place class and loading the lat/long/radius into that, and using a pure Java method to calculate an accurate radius using math (something called the Haversine formula.)

**Wireframe:**

