TESTING STRATEGY AND PLAN

for

NAVATAR

Version 1.0 approved

Prepared by Matthew Berger, Liam Gomez, and Connor Parkinson

Advised by Professor Eelke of the UNR CSE Department

March 17, 2017

Contents

1 Abstract

Navatar is an indoor navigation system for visually impaired students. Unlike existing systems that rely on expensive equipment, Navatar only requires an Android smartphone. The phone's GPS, accelerometer, and compass are used to approximate a user's location and movements and the phone's speaker is used to provide auditory directions to rooms within a building. Buildings on University campuses often have the GIS maps and the design required for Navatar's accelerometer and compass based localization, which is why students are the intended audience for the project. With the application's open source status and modular design, crowdsourcing efforts can be used to integrate maps for buildings on any campus. The application is still under development and Team 6 is working on project management and scheduling for the implementation of a variety of new features alongside other open source contributors.

2 Project Updates and Changes

Team 6 has been working on cleaning up the existing code base to better facilitate the implementation of the planned features while also debugging issues encountered with the existing code. The team has recently added additional priority to fixing bugs with the existing maps and map creation functionality as their advisor, Eelke Folmer, informed them a university has expressed interest in the development of Navatar. Efforts are also being made to automate setup of the development environment to allow other open source contributors to easily assist with improving Navatar.

It became evident after the design and prototype stage that some of the team's ideas were better in theory than in implementation. One such example was using certain gestures as a form of interaction with the application. Gestures that are context-aware lead to the same application interactions eliciting separate and highly distinct behaviors that may lead to unexpected results for the user. The context of the application may be difficult to discern for a blind user. In this sense, gesture-detection functionality must be done carefully to ensure there are not conflicts between Navatar gestures and external accessibility gestures. Voice commands can also be used where gestures are found to not be feasible. We changed our design to also include the use of Google voice speech recognition as an input source should we deem it necessary. Simply speaking to the application through a microphone on a headset or set of ear-buds will allow for accurate control and potentially limitless interactions with the app which solves the issues associated with only using gestures.

After designing, prototyping, and testing the GPS features we found that reliably detecting a user's location indoors was difficult. It was determined that the real use-case for autolocation would be when a user enters a building, which allows for the best location accuracy using GPS. The subsequent use cases including the user leaving the building or navigating between rooms inside the building are made possible using an Android feature that allows for getting the last known location of the smartphone.The autolocation functionality is done using Geo-Fencing currently, and can be improved by using additional location data sources such as WiFi or cellular towers. As for navigation history, it is a nearly complete and functional feature barring the additional planned functionalities of allowing users to reverse the previously navigated route and collecting data on which routes are completable. Additional focus for improving the feature is allowing users to navigate the selections with easily spoken voice commands. Overall there is much to work on and many features to develop with the overarching task of refactoring the codebase to improve the quality of the existing code.

3 Acceptance Criteria

3.1 Route History - Liam Gomez

This component is responsible for logging and storing a user's previously taken routes. Storing the previous taken routes allows for a route repeat feature to be implemented, where a user has the option to view their route history and start a new navigation from that history. Storing route history also allows for a user to a reverse a given navigation after reaching their destination.

3.1.1 User Stories and Required Tasks

1. As a user, after opening Navatar, I should be given an option that takes me to the route history page so I can view my route history.
   - Create a button on homescreen of application that takes user to route history page. Only display button if there is route history to display.
   - Create route history page that pulls history from the route history json file.
2. As a user, on the route history page, I should be able to select a route from my history so I can repeat that navigation.
   - Store route history to json file in local application storage upon completing a given route successfully.
   - Create a new activity responsible for starting navigation from a previously taken route.

3. As a user, after successfully completing a navigation, I should be prompted if I would like to reverse that route so I can navigate back to my origin.
    - Create prompt / button that is displayed on successfully completing a route asking if user would like to reverse.
    - Create a new activity responsible for starting route reversal navigations.
    - Prompt / button should trigger the route reversal activity.
4. As a user, on the route history page, I would like to avoid having failed or incomplete navigations saved to my history so I can avoid repeating them.
    - Ensure route history is not saved to failed to incomplete navigations in NavigationActivity.java
5. As a researcher, I should have access to the route history logs in a parsable format for analytical purposes.
    - Investigate the feasibility of pushing route history logs to a cloud based storage solution.
    - Ensure route history logs are stored in an easily locatable directory of the local device storage.

3.2 Geofencing - Connor Parkinson

This component is responsible for the automatic selection of the campus and building based on the user's current location. This component will use a combination of the smartphones GPS, cellular, and Wi-Fi to provide locational data. This locational data will then be compared with the known geographical boundaries of available campuses and buildings.

3.2.1 User Stories and Required Tasks

1. As a user, I want Geofencing so that I don't have to select the campus I am on.
    - Investigate method for generating and storing campus boundaries that is accurate and easy for other open source contributors and map creators.
    - Implement automatic search for user location on campuses with defined geographic boundaries.
    - When the user is found to be on a campus, have the campus automatically be selected.
2. As a user, I want Geofencing functionalities to be active when I open the app.
    - Have the app automatically begin polling location when it is opened.
    - Investigate best use of sensors and radios to find an accurate location within a reasonable amount of time.
    - Have the app use the last known location if it is recent and accurate enough.
3. As a user, I want to revert to existing campus and or building selection if Geofencing fails (is not accurate enough or times out).
    - When Geofencing attempts fail (did not get accurate enough location before timeout) inform the user via voice or other method so they know to use existing campus and building selection functionality.
4. As a user, I want Geofencing so that I don't have to select the building I want to navigate within.
    - Investigate method for generating and storing building boundaries that is accurate and easy for other open source contributors and map creators.
    - Implement automatic search for user location in or near a building with defined geographic boundaries.
    - When the user is found to be in or near a building, have the building automatically be selected.

5. As a user, I want Geofencing to select at least the campus even when accuracy is not high enough to select the building.

    - Investigate accuracy increase over time trend when polling the user location.
    - Have accuracy required for automatically selecting a campus be significantly lower than what is required for selecting a building.
    - When only the campus has been auto selected, continue to poll user location in attempt to get a more accurate reading for building selection.

3.3 Gesture-Based Controls - Matthew Berger

This component is responsible for providing an additional input method to the user that is outside the existing Google Talkback integration. For a user with visual impairments navigating content heavy menus and input fields can be quite tedious. To improve this, a gesture based input system will be implemented and integrated with the existing user interface. Gesture based input will allow the user to make gestures on their devices screen that control the applications behavior.

3.3.1 User Stories and Required Tasks

1. As a user, I want the option to be able to control the application without having to speak in noisy environments.
   - Have the app detect and respond to gestures such as swipes and long-presses.
   - Have the app use audio feedback and haptic feedback to respond to the user mechanically.
2. As a user, I want gestures to be consistent, intuitive, and reliable.
   - Have the same gestures perform the same actions throughout the app.
   - Have similar gesture functionality as other applications, for intuition.
3. As a user, I want to be able to long press to add a landmark.
   - Investigate landmark component and integrate it.
   - Implement the observer pattern. Receieve from and broadcast events to other components for integration.
4. As a user, I want gestures to be available for the entire screen, not a subsection.
   - Force the app to use the whole screen for gesture detection.
5. As a user, I want to be able to gesture to confirm or reject application prompts.
   - Implement a swipe right for accepting a prompt.
   - Implement a swipe left for rejecting a prompt.

4 Testing Workflow

5 Testing Strategy

6 Team Contributions

6.1 Matthew Berger

- Created LaTeX document
- Setup source and development environment controls
- Completed project assignments and schedule
- Created gesture subsystem user stories
- Completed five of the ten risk management strategies

6.2 Connor Parkinson

- Completed the abstract
- Outlined the project design and requirement updates
- Established the project monitoring plan
- Completed five of the ten risk management strategies

6.3 Liam Gomez

- Outlined the project deliverables
- Organized user stories on the team's ZenBoard
- Compiled the references.

6.4 All Team Members

- Peer review, editing, and formalization of this document and the knowledge contained herein.