Time of Travel Calculator

Cpet\_499 Final project report

trey lohman and GARETTleckrone

2017

Contents

[Figures 1](#_Toc481444700)

[Executive Summary 2](#_Toc481444701)

[Problem Statement 2](#_Toc481444702)

[Proposed Solution 2](#_Toc481444703)

[Mobile App Hardware and Software Requirements 3](#_Toc481444704)

[Mobile App Software Design (use information listed from guideline 1 to 6) 3](#_Toc481444705)

[Top Level Functions for the App 3](#_Toc481444706)

[Design of the User Interface 4](#_Toc481444707)

[The State Diagram of the App 4](#_Toc481444708)

[Class Definitions (UML Class Diagram) 5](#_Toc481444709)

[Mobile App Software Testing and Demonstration 6](#_Toc481444710)

[Summary and Conclusion 10](#_Toc481444711)

# Figures

[Figure 1: User Interface 4](#_Toc481444331)

[Figure 2: State Diagram 5](#_Toc481444332)

[Figure 3: Class Definitions 6](#_Toc481444333)

[Figure 4: App Demo Idol 7](#_Toc481444334)

[Figure 5: App Demo Edit Text Screen 8](#_Toc481444335)

[Figure 6: App Demo Radio Button Screen 9](#_Toc481444336)

[Figure 7: App Demo Compute Screen 10](#_Toc481444337)

Executive Summary

Over the course of the semester there has been assigned multiple styles of applications through Android Studio software. Each app that has been instructed to do, as either a lab or extra class practice, has its own unique taste of functions and design set up. Some of the applications were a bit more function filled and intuitive then others, which gave for a very wide variety of reference applications. For the final project, the knowledge gained from practice with the labs and other assignments is to be used to put together a custom application. Given that there are multiple application that involve some type of calculation would make the choice of a calculator type app easy to implement. The application that was chosen is called the Time of Travel Calculator. The app idea came from trying to determine, without paper and calculator, how much time would be saved or lost, if one were to 5 or 10 miles above or below the speed limit. The idea was initially for highways or rural roads, or roads without stops, but in reality majority are not strictly driving on these types of roadways. With this consideration there was to be added a driving environment section, which would allow for the use of radio buttons along with a radio group to harness these inputs. The app is set up to enter in the millage of your destination, the speed limit on the road of travel, the amount of desired speed above or below the speed limit, and the driving environment. Once entered and calculated the amount average amount of time lost or saved can be determined. Although, this is considerably a simple calculator app, the utilization of a different style application would increase the entertainment of the app and bump up the creativity. This would allow for the animation part of the app to be implemented.

Problem Statement

It is easy to determine the time of arrival when traveling a constant 60 MPH (1 mi/min), but it can be difficult to determine how much time one will save or lose given the change in speed and or the driving environment.

# Proposed Solution

To create an application that can compute the time of travel along with the time saved or lost based on the driving environment, the speed limit, and the amount of speed above or below the speed limit. With the given distance of travel and the speed limit entered, the time of travel can then be calculated. With the speed above or below the limit +/- option, the amount of time saved or lost in minutes will then be displayed. Driving environment, which implements the average number of stops per mile and how long those stops last given the environment (suburb, rural, highway, city), can then create a more accurate time of travel. For entertainment purposes, there was and animation of a vehicle which wheels will rotate each time a new calculation is entered.

# Mobile App Hardware and Software Requirements

Hardware Requirements:

The application shall run on Android devices running 4.2 or later.

Software Requirements:

The application shall support a minimum SDK version of 17.

The application shall target SDK version 25.

# Mobile App Software Design (use information listed from guideline 1 to 6)

## Top Level Functions for the App

For this application there will be a single activity that the user will interact with by entering numbers into the EditText boxes, using increment and decrement buttons, selecting radio buttons, and computing the results using a Compute button.

There will be two EditText views that will be used to enter the distance of the trip and the speed limit.

An increment and decrement button will change the desired amount of speeding. The value of the desired amount of speeding will be displayed between the increment and decrement buttons in a TextView.

The environment will be selected using four buttons in a radio group. Each radio button will change the environment and will determine the algorithm for the total time of stops along the trip.

Lastly, once all of the values have been entered and selected the user will press the compute button, which will display the total time of the trip, the amount of time saved or lost by speeding, and an entertaining animation of wheels turning

# Design of the User Interface

The figure below shows the design of the user interface for the application. All of the values and computations are done within the single activity. After defining the functions of the application, the various views were added to the design window and repositioned for the cleanest appearance. Instead of using RelativeLayout, it was found that AbsoluteLayout allows the UI to be customized more easily and with better results.

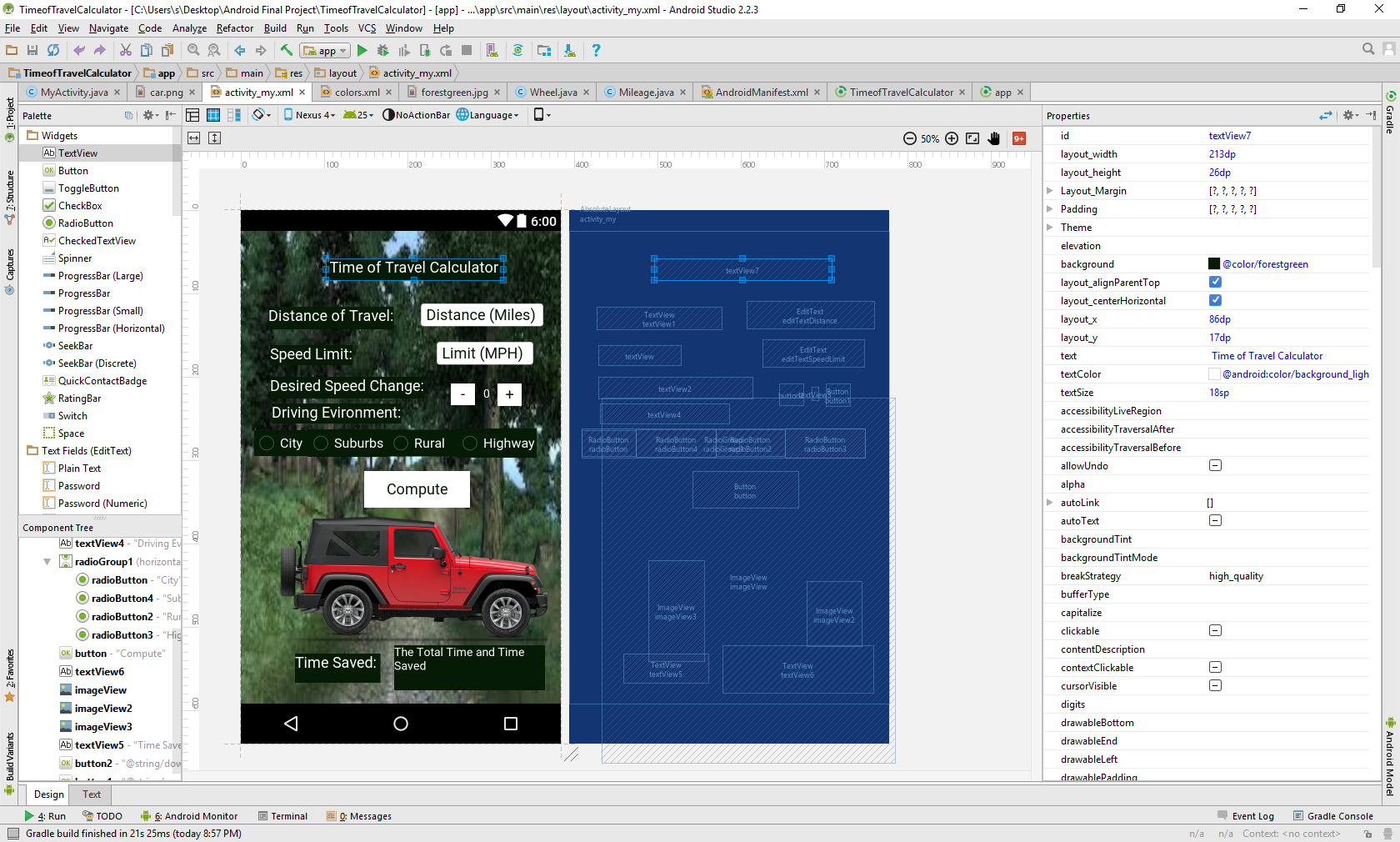


Figure : User Interface

## The State Diagram of the App

Below is the state diagram for the application showing the stages of its lifecycle.



Figure 2: State Diagram

## Class Definitions (UML Class Diagram)

Below is the UML class diagram for the application showing the three java classes for the app. MyActivity is the main class for the application, and will contain and utilize a Wheel object and a Mileage object. The fields and methods are listed in the UML diagram including the input arguments for the methods and the return types.



Figure 3: Class Definitions

# Mobile App Software Testing and Demonstration

The figure below shows the android application being simulated and shows the android application in the Idle state waiting for values to be entered.

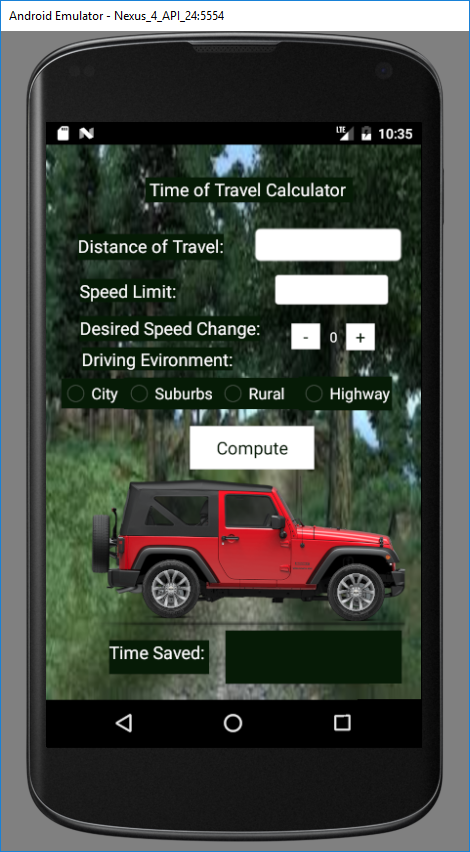


Figure 4: App Demo Idol

The figure below shows how the user enters data into the EditText boxes for the distance of travel and the speed limit. When the user presses the EditText view, the number keyboard will appear and allow the user to enter the values. For this instance, the user has entered that they will be traveling 100 miles at a rate of 60 miles per hour.

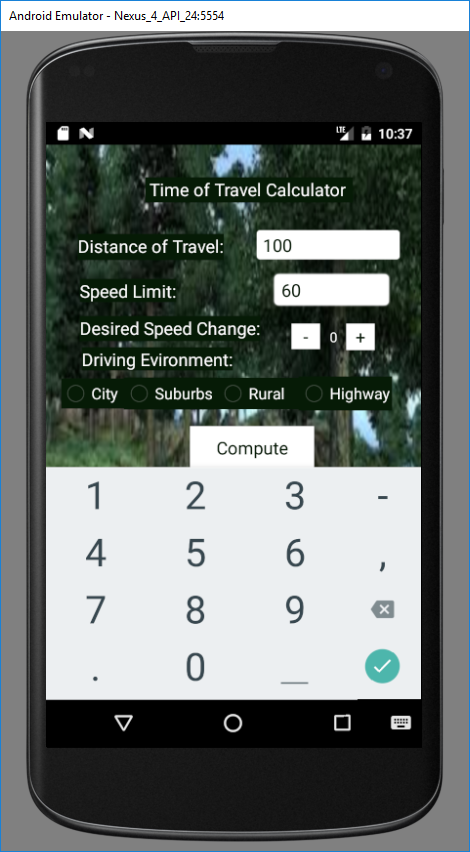


Figure 5: App Demo Edit Text Screen

The figure below shows that the desired speed change can be changed by using the increment and decrement buttons and the driving environment can be selected using the radio buttons. For this instance, the user has indicated that they would like to travel at 10 miles per hour over the speed limit and intend to drive on the highway.

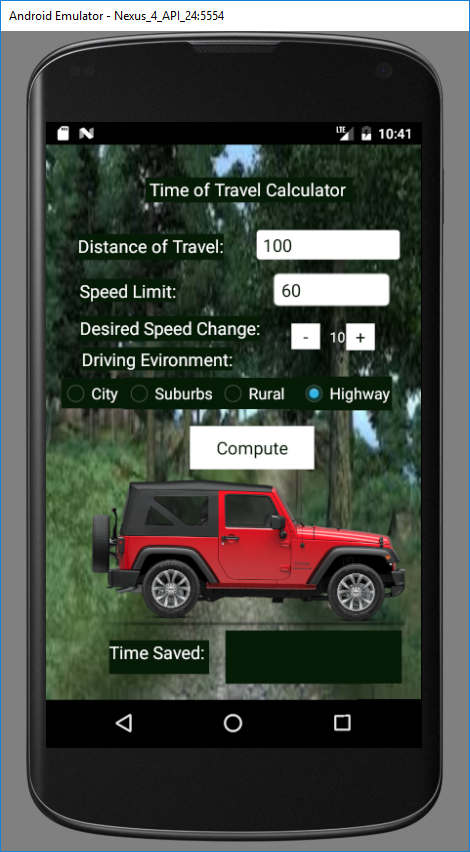


Figure 6: App Demo Radio Button Screen

The figure below shows the results of entering values into all of the fields and then pressing the compute button. The results are shown at the bottom in a text field and indicates the total time that the trip will take, given the amount of speeding selected, and also the amount of time that was saved. Also, when the compute button is pressed the tires on the Jeep begin turning for 25 rotations. In this instance, the user will be traveling 100 miles at 70 miles per hour on the highway, which means that they will save 14 minutes and their total trip will take a little more than 85 minutes.

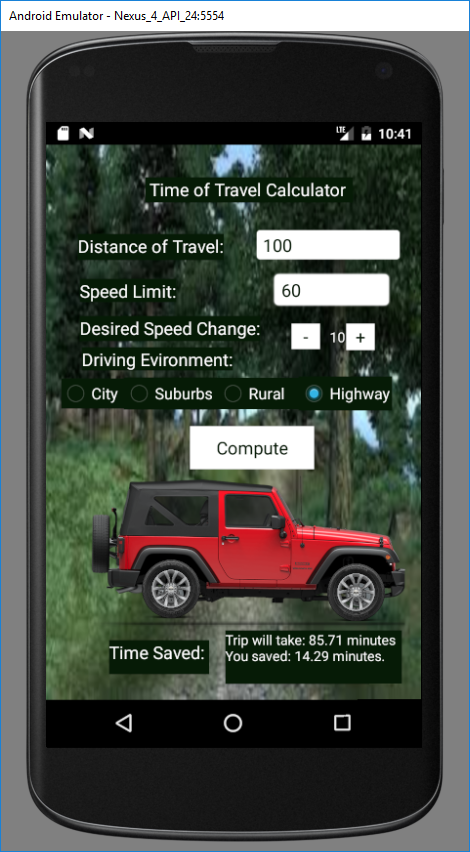


Figure 7: App Demo Compute Screen

# Summary and Conclusion

The app ended up being a success, although there were some complications through many different parts of the project. The design part of the app had many of its own problems that accrued. The major GUI inconvenience would have to of been the positioning of the components. Although there was a way around the positioning with each mothed, it became evident that some methods would be a bit easier to work with then others. First, there was the use of the relative layout, which was based on a snapping effect which would align and snap each component to one another. This made it difficult to making those precise placements for the animation with the car and wheels, which were three separate JPEG files. What would come to be a helpful alternative method is the absolute layout, which would allow for the aligning and snapping portion to allow for free movement of components. The snapping portions wouldn’t be completely eliminated, but instead would make it inaccurate with the exact placement choice. The best way around this would be to go into each of the component’s properties and choose the exact x and y layout which is in absolute layout exclusively. Other problems that accrued in the GUI portion of the project was the rendering problem, which had to do with how the images were saved in the drawable folder. To start with the images were saved as png files which caused the first rendering problem and was easily fixed by saving them as JPEG files instead. As far as the coding aspect one of the most notable problems faced was with the radio buttons in the radio group. The button listeners were not responding and the computed results would not change despite changing the environment. Once found that it was needed to create a separate button listener that existed outside of the compute method in order to get the values stored for the driving environment to change when the selected radio button changed. Other than that the only major problem that was encountered was sending the project back in forth through a zip file the only way around this was to send the entire project as a text file.