**CCR – Mobile Application**

**(Computer Controlled Railroad)**



**Name:** Gwynn Fewell

**Professor:** Dr. Pankratz

**Class:** CSCI 460 – Senior Capstone Experience

**Date:** May 8, 2012

**Table of Contents:**

**General About Me**

* Introduction..……………………………………………………………………………………………………….……3
* Resume……………………………….………………………………………………………………………………......3
* Philosophy..……………………………………………………………………………………………….……………..4
* Career Plans …………………………………………………………………………………….………………………4

**CCR-Mobile Project**

* Project Definition……………………………………………………………………………………………………...5
* Project Requirements………………………………………………………………………………………………..5
* Brice Hilgemann’s Project ………………………………………………………………………………………..6
* Initial Strategy ………………………………………………………………………………………………………….7
* Client Interface …………………………………………………………………………………………………………8
  + Design ……………………………………………………………………………………………………………8
  + Main job ………………………………………………………………………………………………………..9
  + Structures ………………………………………………………………………………………………………9
* Communication – Protocol…………………………………………………………………………………………10
* Issues Resolved ………………………………………………………………………………………………………….11

**Documentation**

* Quick User’s Guide ………………………………………………….…………………………………………………..12
* Presenation Documentation …………………………….………………………………………………………….13
* Client Code ………………………………………………………………………………………………………………….20

**Introduction**

This is Gwynn Fewell’s Senior Capstone Project for the Computer Controlled Railroad Mobile Application.

This project has taken the client aspect of previous projects over to a mobile application. This project focuses on creating a client in which many devices can interact with the railroad through a previously programmed server.

**Computer Controlled Railroad Mobile Application**

Model trains have always caught the interest of many people of all ages. Now we are going to bring them into the mobile world. The goal is to build an Android application that allows the user to connect to a server and control the trains. This project will bring a little simplicity to operating the model railroad. It will be kid friendly and operable by all ages.

The following pages will give an introduction to the project and the author of this project. Documentation is included throughout this document as well as the project website at

<http://compsci02.snc.edu/cs460/2012/fewege>.

If there are any questions, feel free to contact me at [gwynn.fewell@snc.edu](mailto:gwynn.fewell@snc.edu).

**Philosophy**

I never intended to be a major in Computer Science but have found it to be a subject I actually enjoy to not only do but to also learn about. When thinking about how Computer Science has affected me and what I have learned so far I found the acronym PLAY to be very fitting.

**Perseverance:** Perseverance is a great word when it comes to computer programming and Computer Science. When programming every person will encounter problems. The problems may be in writing the most efficient system or finding the correct algorithm to use, but I have found the hardest problems come when you are closer to the finished product. When debugging the program you have just created, tiny errors that may just be a syntax error can take hours. Therefore, perseverance is needed to remember to work through the problems and stick with them because big or small, the end result is rewarding.

**Logic:** Logic plays a role in every aspect of Computer Science and I think that was my initial draw into the major. The program can be a huge puzzle that when all the pieces fit together just right, a beautiful finished product is just right. Logic is needed in debugging programs as well as developing useful algorithms that work efficiently with both time and space. Reason is used every day in Computer Science to argue the effectiveness of an algorithm or language and how they work differently from others.

**Ability:** The ability to do the work and use the help of others is needed in Computer Science as well. You have to have the ability to sit down and do the work and have the ability to ask others their thoughts on your approach to a problem. Communication is a large part of the Computer Science community because other people's code is available and useful when trying to solve a problem. If computer scientists didn't work together to help advance technology then the computers and devices that are available today would not exist. Therefore, as a computer scientist you need to have the ability to work with others and to interpret other people's work to assist your own work and always give credit where it is due.

**Yield:** Computer Science always yields a lesson in every project. Whether it is from another person's code or from playing and learning on your own, learning is always taking place. Technology is always rapidly evolving and there is something new to learn about it everyday. With this, you always have to be open to new ideas and learn how to adapt to the new platforms, languages, and projects that are brought before you.

I think that PLAY is the best acronym for the Computer Science field in general. As a computer scientist, you always have to be willing to play with something new and sometimes get things wrong in order to learn more about it. Also, actually playing with the user side of an operating system such as Android or Windows help you know how it runs better. I have found a lot of joy in Computer Science and therefore have always felt as though I was playing more than working when learning something new. I believe Computer Science would not be so attractive if I weren't able to PLAY.

**Career Plans**

After I graduate from St. Norbert College in May of 2012, I plan to move into the workforce. Although I do not have anything lined up and nothing particularly planned, I plan on putting myself out there and applying to all kinds of positions. I’m hoping to obtain an entry-level position in which I can further my knowledge and skills.

I plan on spending a lot more time with my daughter. I will continue to be in the Green Bay area and believe that St. Norbert has started me on a road to success with skills that I will use all of my life.

**CS460: Capstone Experience Project Definition**

**2012 DC Pankratz**

**COMPUTER CONTROLLED RAILROAD – MOBILE**

**Name:** Gwynn Fewell

Project: Use mobile devices to run trains on the CCR.

**General Description and Requirements:**

1. Build on the work of Brice Hilgemann, 2010
2. Each mobile app becomes a CAB that controls one train at a time
3. The app can switch between trains
4. Each CAB shows the state of each train.
5. Consider controlling trains using the accelerometer
6. Investigate racing conditions on CABS and track
7. Design the app so that the young and young at heart can play

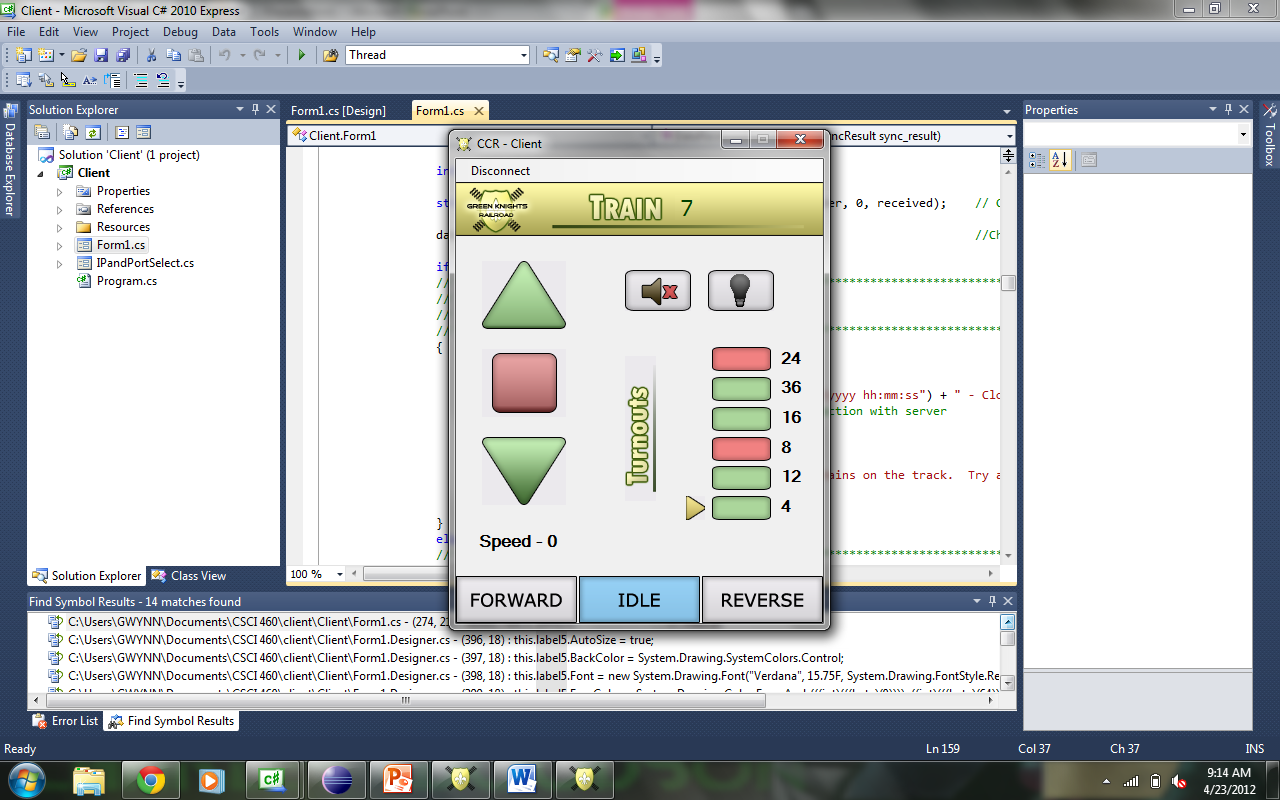
**Brice’s Project**

Description: Brice did a lot of work to create a visual server and client that allowed the user to view the trains by a simulator on the server. Brice also created a much easier interface for the client that allowed the user to change turnouts by a press of a button. His client and server can be seen below.

Server:



Client:



Location:

<http://compsci02.snc.edu/cs460/2010/hilgbs>

**Initial Strategy**

Initial Thoughts – When I first received this project I had no experience with coding Java or Android, never played with a CCR, and knew very little about sockets and how to use them. So my initial thought was to dive into Android and figure out the workspace I was going to be using.

Since the project was based off of the work of Brice Hilgemann, I knew I had to get into his code as well. I had to become familiar with how his server and client worked and how the trains worked with each of them. I was going to have to understand the similarities and differences of built in objects between C# and Java.

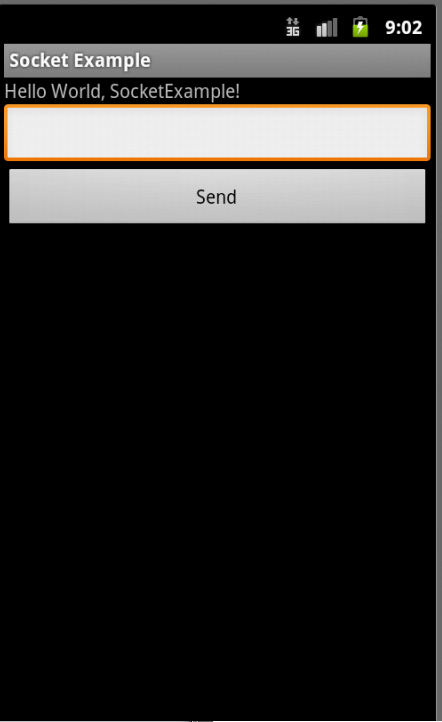
Initial Goal/ Concepts:

* Get familiar with the server and client created by Brice Hilgemann.
* Understand Java, Eclipse and Android.
* Utilize previous work
* Test and control connection issues and racing conditions

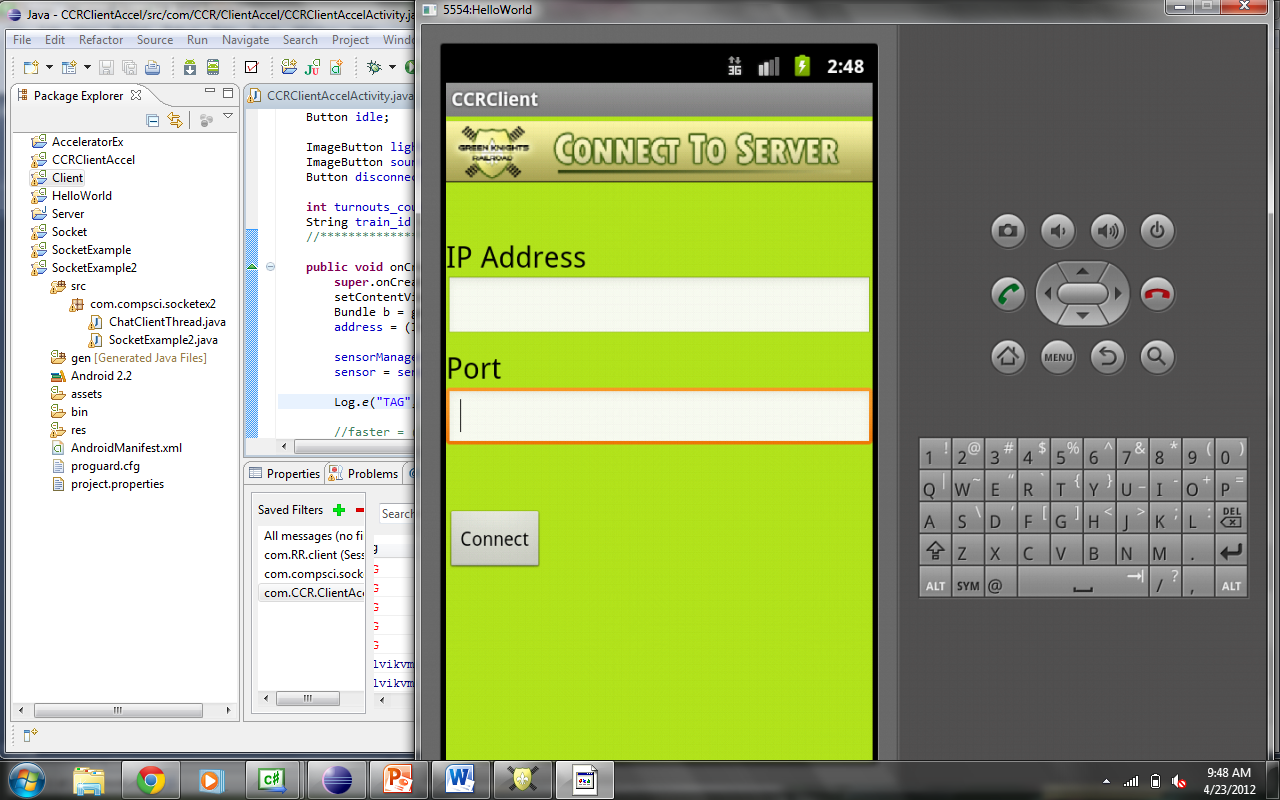
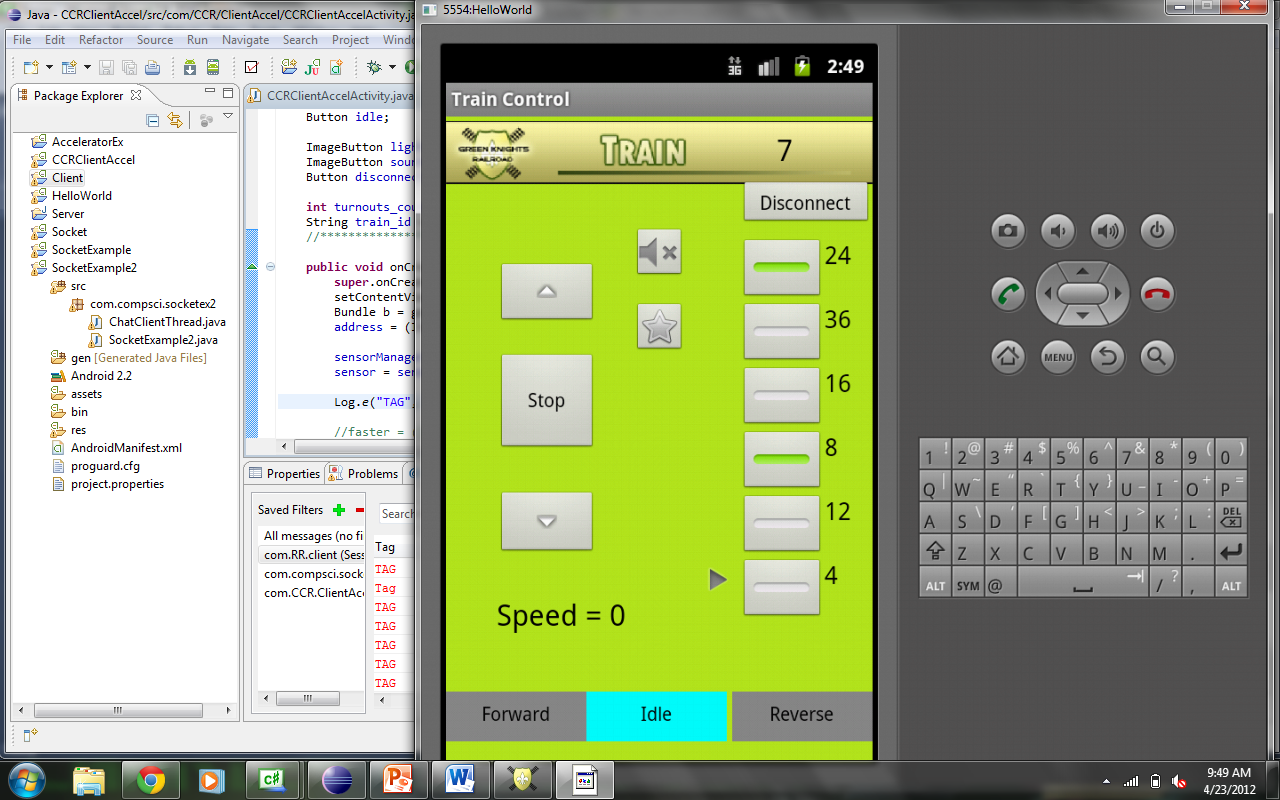
**Client Interface**

**Client Design:**

Phase 1:



Final:

**Main Job**

The main job of the client is to connect to the server and allow the user to easily control the trains. Through buttons, the user is able to send commands and toggle turnouts, lights, and sound. The user is able to control the train’s speed and direction of travel. The information regarding the train and its location is given by the train number and an arrow indicating the next turnout the train will reach in its path of travel.

**Structures in Client**

There are no structures in the client since all information is received from the server. To be able to test conflicts and approve requests, all processing of commands is done by the server and the interface of the client is updated based on the information passed back from the server.

**Communication – The Protocol**

The following is directly from Brice Hilgemann’s documentation to show the protocol used for the communication between the server and the client. The commands are sent across sockets where the server uses callbacks and this implementation of the client uses threads to handle the information from the server.

**Protocol to Server**

|  |  |
| --- | --- |
| **CMD FORMAT** | **DESCRIPTION** |
| [ 1 ] ; [ TRAIN ID ] | Increase speed or up arrow issued |
| [ 2 ] ; [ TRAIN ID ] | Decrease speed or down arrow issued |
| [ 3 ] ; [ TRAIN ID ] | Stop train has been issued |
| [ 4 ] ; [ TRAIN ID ] | Direction forward has been issued |
| [ 5 ] ; [ TRAIN ID ] | Direction backward has been issued |
| [ 6 ] ; [ TO # ] | Toggle turnout has been issued |
| [ 7 ] ; [ TRAIN ID ] ; [ DIRECTION ] ;  [ ORIENTATION ] ; [ COLOR ] ; [ TRACK ] | User wants to Add a train from the server – get basic information from form and call with protocol to add |
| [ 9 ] ; [ TRAIN ID ] | Toggle light has been issued |
| [ 10 ] ; [ TRAIN ID ] | Toggle horn has been issued |

**Protocol to Client**

|  |  |
| --- | --- |
| **CMD FORMAT** | **DESCRIPTION** |
| [ 0 ] | There are no trains available – disconnect |
| [ 1 ] ; [ TRAIN ID ] ; [ SPEED ] ; [ DIRECTION ] ; [ LIGHT ] ; [ HORN ] ; [ BLOCKED ] ;  [ TO\_NEAREST ] ; [ NBR TOS ] ; [ TO# ST ]... | General update of train information sent from the server to the client to update images and status indicators |
| [ 2 ] | Train has been deleted or client has been disconnected from server |
| [ 3 ] ; [ TRAIN ID ] ; [ NBR TOS] ; [ TO # ]... | Beginning train information for setting labels |

**Issues Resolved**

**Connection Issues**

One of the problems I had was connecting C# to Android. Android would kill the application when the server wouldn’t respond within a certain amount of time. To fix this, the read command in Java needed to be threaded separately from the main thread of the application so that Android would not kill the application. This worked effectively once the read command was threaded effectively.

**Firewall (SNC)**

Another problem I had was still a connection issue. Once I had my socket working off campus, I had troubles connecting when trying to connect to the actual trains on campus. This came down to an issue with the firewall. I contacted Rob Downard of IT, who then created a hole in the firewall for the specific IP address I was using with the physical address of the computer I was using for the server. This meant the trains couldn’t remain in the PAC building due to signal issues and had to be moved to Cofrin. Both the move and the firewall fixed the issue.

**Accelerometer implementation**

The other major problem I ran into was when implementing the accelerometer with the trains. I used a snippet of code that would pull the x, y, z coordinates of the device. When the sensor changed, then the coordinates would be retrieved again. However, this implementation proved not effective since I was sending a command each time the sensor changed and was filling up the queue with a command for the train to go in one direction and then wouldn’t allow the train to receive any other commands. Therefore, a better implementation was needed in order for the accelerometer to work effectively.

**User’s Guide**

Use the following instructions as a guide to control the trains on the track. If there are any issues with a device connecting to the server, be sure that the firewall has been fixed and a hole has been created to allow access.

Files needed: Client.apk, Server.exe with associated .txt files that hold default track information

**Setting up the Server** (from Brice’s project) :

1. Connect all of the track pieces together and plug in both of the cords to give the track power.
2. Switch both power adapters from the ‘Blackbox’ and ensure that all three green lights are lit on the track. Turnouts and other indicators should also be lit up.
3. Plug USB for direct connection into the computer/laptop that is going to be used as the server
4. Right-click on ‘My Computer’ on the desktop and select ‘Manage’ to access Computer Management. Here you can verify that a COM is associated with the USB
   1. Click ‘Device Manager’ and then expand the Communications tab to verify that a COM is set up with the USB. It should be listed as a Prolific USB device
5. Launch Server.exe from the associated folder where the .txt files for track defaults are located. It is crucial that these are in the same folder for setup purposes.
6. Locate the server IP and port listed at the top for client connection

The server can now add trains through the ‘Add Trains Dialog’, remove trains once added through the ‘Remove Trains Dialog’, and see all connections. If there are any questions about operations that the server can perform, please view my videos (Brice’s videos) on my site (Brice’s website) or review documentation.

**Setting up the Client:**

1. Install the .apk onto the device you would like to use.
   1. There are various ways to install, but the easiest way is to use the Google Gmail account associated with the device and email the .apk file as an attachment to that email. Once the email is received, the .apk file will have an install button. Then the device will install the application.
2. Open the client and retrieve the port and IP address that is listed on top of the computer that is running the server
3. Verify that a train has been added so that when “Connect” is clicked, you will have control of a train to send commands to (be sure a train is available as well)
4. Click connect
5. Control the trains
   1. Up and down arrows increase and decrease speed (negative speed = reverse)
   2. Turn out toggles light up to move toward top of track and move toward bottom of track otherwise
   3. Arrow next to turnout toggle buttons indicates next turnout in path of train
   4. Train number is displayed at the top
   5. Sound button turns on and off the horn
   6. Star button turns on and off the lights (if lights are working on trains)
   7. ‘Forward’, ‘Idle’, and ‘Reverse’ buttons at bottom indicate direction and allow user to change direction

The user can now control the trains using the above commands. A speed of “\*\*blocked\*\*” is displayed if a turnout is in improper position, another train is in the way, or the train is at the end of a track. Some commands will not work if the “\*\*BLOCKED\*\*” speed is displayed. This is not in the user’s interest to change.

**Presentation**

