Slide 1

- Hello my name is Quinn Roemer and I am going to show you guys what I had the responsibility of doing in the code that we created.
- I can split my responsibilities into three main categories.
- The first of which is, putting in input validation for the menu.
- Second, designing the framework for the code.
- Third, designing a procedure that could be called to generate random numbers.
- In the next couple slides I will explain each in a little more detail.

Slide 2

- Like I said in the previous slide I was responsible for implementing a form of input validation in the beginning menu that you will see in our code.
- Although Luke did most of the programming for the menu I added a couple needed features that made his code even better!
- These input validation features resemble IF statements in higher level programming languages. The main idea behind my code is that if a certain event happens a flag is set so that the program will know that it is okay to continue.
- The first feature that I implemented was the ability for the code to detect whether or not an item had already been bought. This prevented the user from buying multiple items in the same menu. Something we did not want since our code was not set up properly to deal with multiple wagons.
- The second feature that I added was the ability for the program to detect whether you had enough funds for the item that you wished to purchase. Needless to say, this was necessary because we did not want the user to purchase the best item in each category. Our design decision was to allow the user to buy two of the more expensive items and one of the cheaper items.
- Lastly, I added code that prevented the user from quitting the menu without purchasing one item from each category. The reason that I saw the need for this code to be implemented is because through purchasing the items the variables that hold the value of the oxen, wagon health, and food are not set. This prevents the player from leaving the menu with no food or oxen immediately losing the game.

Slide 3

- In designing the framework for our code, I needed to do several things.
- First and foremost, I needed to code a place where all of the other people in my group could place their scenarios one of which would be called each turn.

- This was done by implementing a rather large jump/loop structure in the code that is able to jump to the correct scenario.
- If you want to see code in that actually does this you will be provided with a link to download our code and run it for yourselves at the end of our presentation.
- After Adding that feature I needed to add the lose cases. You lose the game whenever one of your supply counters hits or goes below zero. This causes a certain message to trigger making you ultimately lose the game.
- Next, I needed to add the win game end. This code simply get jumped to when the turn counter reaches zero or below.

Slide 4

- The last responsibility that I had was designing code that allowed us to produce random numbers.
- In a high-level language such as C++ you need to call a couple functions to create random numbers. Once to grab a seed for the rand function. And the rand function itself to give you the random number.
- However, it is not the simple in assembly.
- After a lot of playing around and trying different approaches I eventually settled on a method which gets the job done pretty easily.
- Essentially what I am doing in the procedure that I implemented is grabbing garbage data from the registers at the beginning of the code. This garbage data is then used to create the seed for my random procedure. I then plug in into the formula that you can see at the top of the screen and implement it as a procedure.
- All of the numbers that you displayed on this slide are sample outputs of my random number generator. There are 50 numbers in total with a range of zero through ten.