AUTOMATED SCORING SYSTEM FOR TICKET TO RIDE

By

Andrew Douglas

Matthew McCracken

Final Report for ECE 445, Senior Design, Fall 2017

TA: Kexin Hui

13 December 2017

Project No. 43

Abstract

Discussed in this report are the inner workings of our automated version of the popular board game Ticket to Ride. All of the game pieces been removed and replaced by LEDs and players have to do none of the score calculations. Our product significantly reduces playtime and increases enjoyment as a result. The end product is a working proof of concept that can be applied to a full-scale version of the game in question.

Contents

1. Introduction

Ticket to ride is a popular board game for good reason: it's fun to play. The only thing that bogs down an otherwise enjoyable experience is the scoring process. There are 36 cities in total and paths in between adjacent cities. This allows for 100 unique paths (with a total of 308 train spaces) to be claimed while playing Ticket to Ride. Each path has a differing number of cars required, ranging from 1 to 6. The more cars laid down at once, the greater the point reward: 4 points for 3 cars, 7 points for 4 cars, 10 points for 5 cars, etc. As the scoring doesn't linearly translate to the number of cars on the board, scoring is confusing and mistake ridden [1]. Figure 1.1 shows the game board and how the paths are laid out.

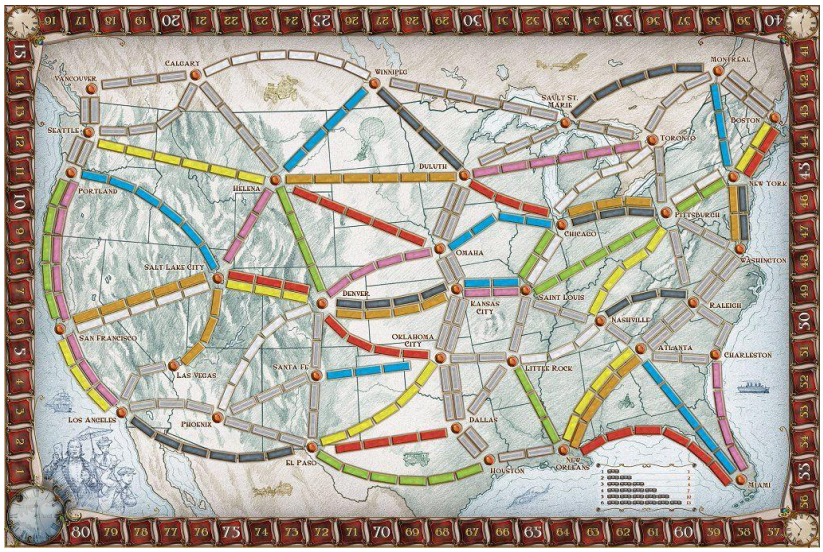

Figure 1.1: The game board for Ticket to Ride showing the different cities and paths to be claimed.

To automate the scoring process for ticket to ride, some way to keep track of pieces on the board was required. Using some sort of sensor on each square on the board was examined, but using 308 sensors in total plus other components proved to be costly. Further automation of the game by eliminating pieces was a cost-effective measure that makes the game even more convenient for the player. This is accomplished by lighting LEDs instead of placing train pieces to claim a route.

The final game works as such: when a player wishes to claim a path, they push the two corresponding city buttons, the path lights up in their color, and the score is automatically updated and displayed on a LCD display. The design itself went through many iterations, ultimately resulting in a proof of concept that accomplishes the tasks presented, represented by Figure 1.2:

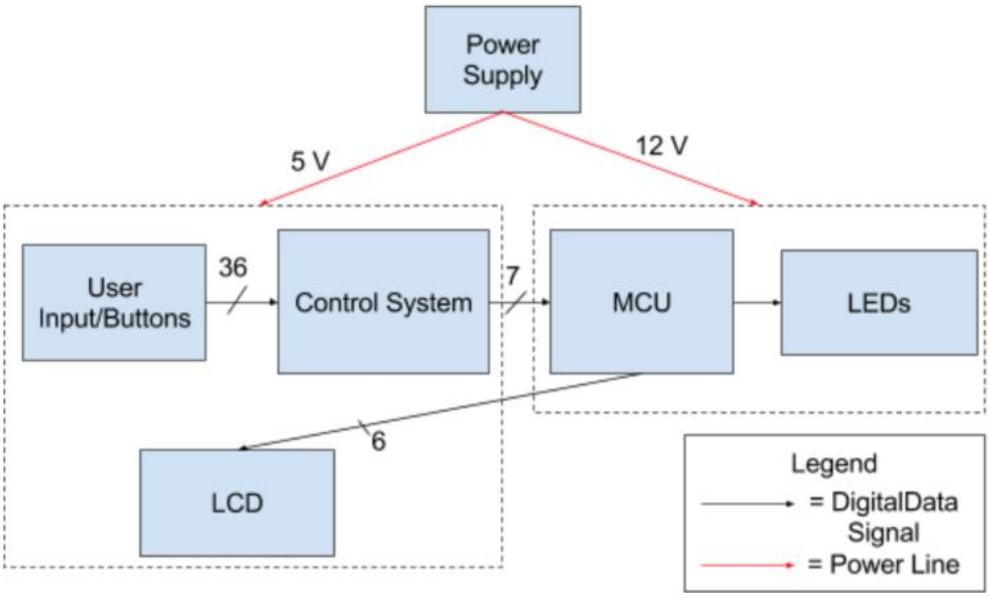

Figure 1.2: Overall block diagram for the design.

This project can be divided into six blocks: Power, User input, Control, MCU, LED, and LCD. The power supply provides power to the whole design. The user input module consists of buttons that allow the user to provide input to play the game. The control module compresses the amount of data sent to the MCU module. The MCU module processes player moves, updates score, lights corresponding LEDs, and controls the LCD. Finally, the LCD module displays all player information such as score and cars left.

Goals:

- Reduce playtime
- Automate scoring process
- Eliminate game pieces

High level requirements:

- Light up train spaces
- Calculate and update score every turn
- Automatically end game

2. Design

2.1 Power

The power module is one of the components that underwent significant changes over the course of the semester. The two options that had been initially considered were to use batteries or to use a power adapter that plugs into the wall. Batteries have the benefit that the game remains portable, but they need to be replaced and don't store that much energy. Using wall-power solves the problem of replacing batteries and can also supply much more power than any reasonable number of batteries. The downside to wall-power is that needing to be near an outlet limits the portability of the game and having a cord coming out of the project would not be as aesthetically pleasing as having a contained compartment for batteries. Initially, the decision was to use battery power for its superior portability, however, we hadn't considered how long the batteries would last with the power draw of our project.

The component of our project that consumes the most power, by far, is the LED module. We measured each strip of 3 RGB LEDs to consume about 60 mA of current when producing white light. Note that producing white light draws the most current because the red, green, and blue LEDs that make up the RGB LED must all be on at full power. When producing purely red light, we measured a strip of 3 RGB LEDs to only draw about 20 mA of current. This makes sense as only the red LED needs to be on. Based on the number of board spaces that need to be illuminated, a total of 135 strips of 3 RGB LEDs. If all of these LEDs were to be producing white light at the same time, there would be a total current draw of 135 *60=8100mA or 8.1A. Understanding the huge amount of current that would need to be provided, also considering that the LEDs need to be run at 12 V, using batteries was deemed a poor solution for the power demands of this project.

Having eliminated batteries as an option, a wall plug power supply needed to be selected that could handle the current draw of the LEDs in addition to the other components in the project. The maximum current draw of 135 strips of 3 RGB LEDs was calculated to be 8.1A, however this is significantly more current than will be used in practice. All 135 strips are needed to illuminate the entire board, but there is no way that all of the spaces can be claimed under the rules of the game. Each player possesses 45 train cars that they can use to claim routes and there is a maximum of 5 players in a game. Therefore, if all of the players were able to use all of their pieces, a total of 45*5=225 spaces would be illuminated. Given that there are a total of 308 spaces on the board, only about 73% of the spaces could be on at one time. This reduces the maximum current draw to 0.73*8.1=5.9A. The upper bound provided by this calculation assumes that all of the LEDs are producing white light. In reality, each player will have a different color, meaning that the not all the LEDs will be producing white light. As measured previously, producing colored light draws less current when compared to producing white light. Therefore, it is safe to assume that the upper bound on the current draw of the LED module is 6A.

Based on the requirements of the LEDs, a power supply is needed that can deliver at least 6 A at 12 V. The other modules in the project need to be accounted for in the selection of the power supply. Our microcontroller was measured to draw 25 mA when running code and the LCD consumed about 3

mA. The user input module draws a negligible amount of current when buttons are being pressed and consumes no current when none of the buttons are pressed. Like the user input module, the control module used a negligible amount of current. In the end, a wall plug power supply was selected that could deliver 8A at 12V. The whole design should not consume much more than 6 A at maximum, so this supply is able to handle the maximum current draw of the design with a fair margin of error.

2.2 User Interface

The User Interface module is a fairly simple module. In the initial design, there was to be a PCB to check for valid inputs: a series of AND gates and XOR gates would ensure that one and only one valid input was being pressed at one time: Ready = $(C_j \,\&\, C_k) \oplus (C_p \,\&\, C_q) \ldots \oplus (C_x \,\&\, C_y)$. A series of flip-flops and an AND gate would ensure that the output 'ready' signal was only high for one clock cycle. This design was ultimately thrown out based on the sheer cost of components and soldering time required to implement the design. A more cost effective and faster solution was to check the city inputs in software instead of a dedicated piece of hardware.

2.3 Control

In order to only have to use one MCU for our purposes, a way to compress the 36-bit input of the city buttons was required. The main hardware component of this design, the control module, deals with compression of the city inputs. The city buttons act as a 36-bit binary input to the system, and the output is two integer values: the number of low signals encountered before a high signal. The desired process is outlined in Figure 2.1 and the schematic for the PCB is shown in figure 2.2.

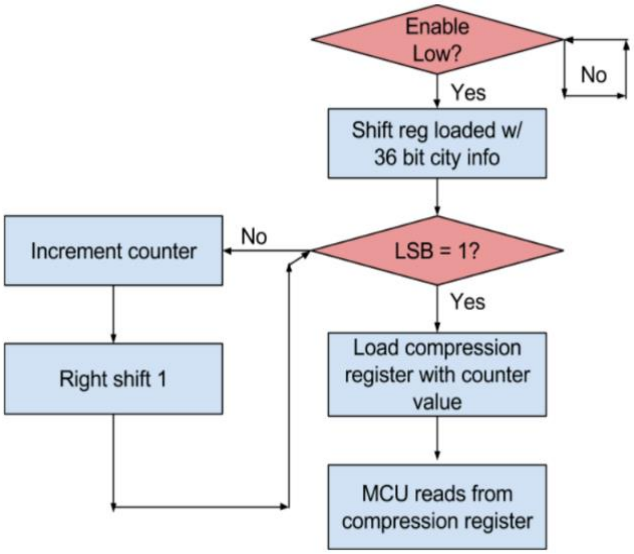

Figure 2.1: Flowchart of compression operation

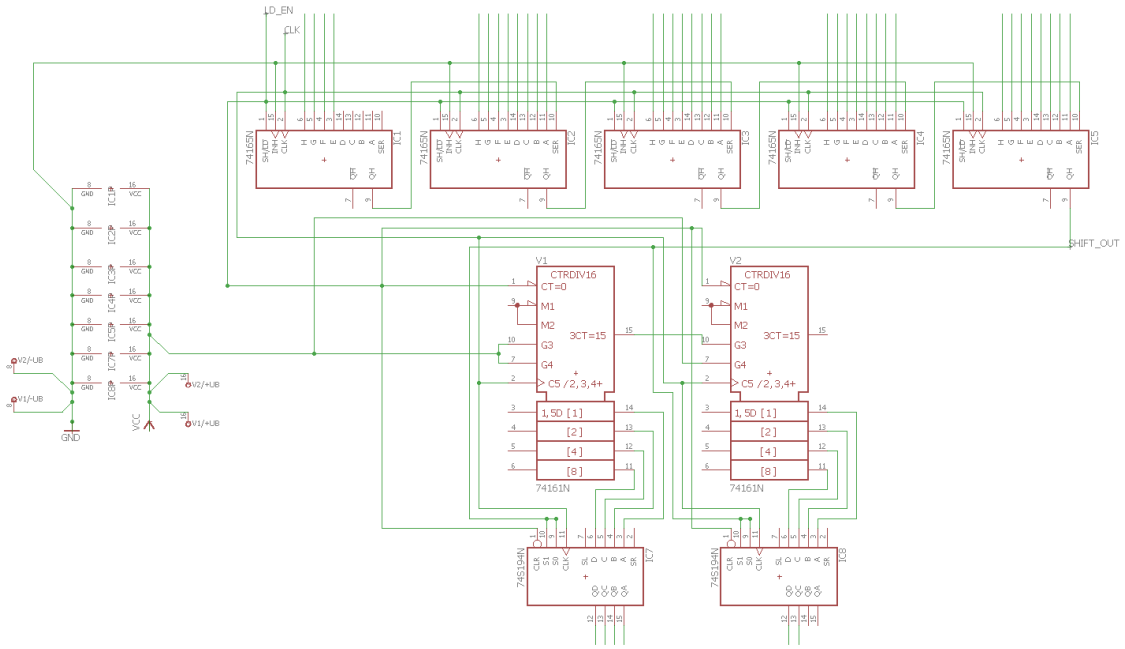

Figure 2.2: Schematic of Control Module

The button data was loaded into 8-bit parallel load shift registers (74165) [2]. Two 4-bit counters (74161) [3] are on the same clock as the parallel load shift registers. The shift-out of the parallel load shift registers acts as the load for the two parallel in - parallel out shift register (74194) [4] which then holds the final output: the number of 0s before a 1 from our original city-button inputs.

2.4 MCU

The MCU module is responsible for maintaining the state of the game and calculating the player score. In addition, the MCU must communicate with the control module to evaluate a player's move as well as communicate with the LED and LCD modules to give visual feedback to the players. An Arduino Mega 2560 [5] was used as our MCU because we had one on hand. The block diagram representing the general flow of the game is represented below in figure 2.3:

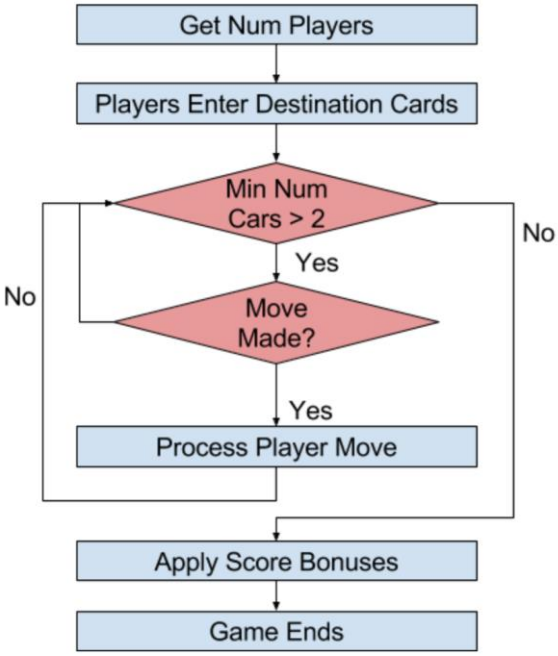

Figure 2.3: Software Block Diagram

Each element in this block diagram is a major step in updating the game state. At the beginning of the game, the number of players is entered and each player takes turns entering their destination cards. The game continues until one player has less than 3 train cars remaining. While the game is in progress, the MCU polls the control module to see if a player has made a move. When a valid move is detected, the game state is updated and then the game continues. Once a player has 2 or less train cars remaining, the longest path and destination card score bonuses are applied and the winner is declared.

The entire board is shaped like a graph, so it was logical to use a graph data structure to represent the game board. Each of the cities make up a node in the graph and the paths between them are weighted edges. An adjacency list was used for the graph, rather than an adjacency matrix, to optimize for memory at the expense of runtime. The memory available on the microcontroller is very limited and the graph of the game board is sparse. Using an adjacency matrix would have wasted a lot of valuable memory just to gain a small performance boost. The program only needs to run fast enough for the player to deem it responsive. For this reason, we can afford some delay in the software.

The longest path bonus can be assigned by using Dijkstra's algorithm. The weights of the edges can all be made negative and then running Dijkstra's algorithm for each player will give the length of the longest continuous path each player has. Determining if a player has completed a destination card is also relatively simple. Starting at one of the cities specified on the destination card, a breadth first search reveals whether that player has connected the two cities in question.

2.5 LED

The LED module had to eliminate the need for physical pieces to play the game. By illuminating different color LEDs on each of the board positions, players can still visually identify who is in control of the various routes on the board. In the early stages of the project, individual RGB LEDs would be placed under each of the spaces on the board. This approach would require soldering 308 individual LEDs to PCBs that would be placed underneath the game board. A huge amount of time would have been necessary to solder these individual LEDs in place, not to mention the large cost of creating all of the PCBs to go under the game board. A cheaper and more time-effective solution was to use RGB LED strips. The strips selected were WS2811 12V RGB LEDs. In addition to reducing the amount of time that we would spend soldering, using LED strips simplified interfacing with the MCU. Rather than having to use a one GPIO pin to control a single group of LEDs, the LED strips have a 1 bit serial data line that can be used to address LEDs on any strips that are connected in series.

2.6 LCD

The primary function of this module is to give detailed feedback to the players. To achieve this purpose, two different ideas were considered: output relevant information to an LCD display or create a mobile app that could communicate with the MCU and display the game state information. The mobile app has the benefit that the larger screen size could be used to display more information. In addition, all the players can easily read the information if it is displayed on their own mobile device. Comparatively, the LCD can display much less information and would be difficult to read for somebody who isn't sitting directly in front of the display. While the LCD is clearly inferior in terms of functionality, it has the benefit of being cheap and easy to implement. A mobile app would have required a significant investment of time to develop the app and to add a communication module (Bluetooth, Wi-Fi, etc.) to the project.

The specific LCD used in this project was a 2 row by 16-character display that used Hitachi HD44780 LCD controller chips. Controlling the LCD was made easy because the LiquidCrystal library for Arduino works specifically with the Hitachi HD44780 chips. Since the display only has two different rows for text, conveying information posed a challenge. Each line of the display is used to show current score and number of train cars remaining for a single player. For example, a line such as "P1 15 39" is displayed to show that player one has 15 points and 39 train cars remaining.

3. Design Verification

3.1 Power

The verification process for the power module was straightforward. The multimeter was used to check the output voltage of the supply. We measured an output voltage of 12.1 V, which was within our voltage tolerance. To verify that the power supply could handle the current draw of the project, a total of 225 LEDs were connected to the power supply. Recall that this is the maximum number of LEDs that could be powered on during the game. All the LEDs successfully turned on and the power supply did not grow hot to the touch. One benefit of the selected power supply is that it had several 2.1 mm plugs as output. This is the exact plug that is used by the Arduino. To see if this plug could be used directly with the Arduino, it needed to be verified that the plug was center-positive. Placing the positive lead of the multimeter inside the plug and the negative lead of the multimeter on the outside of the plug, a reading of positive 12.1 V was obtained. This confirmed that the plug was center-positive and thus could be used directly with the Arduino.

3.2 User Interface

Verification for the user interface included a multimeter and an LED block. First, an LED block was hooked up to each button circuit and the button was pressed. If the LED turned on, a multimeter was then used to ensure we were getting 5V and ground from each button. The current was then measured across the pull-down resistor to ensure that the resistor value was high enough to reduce current draw. A 1kΩ resistor was chosen so the buttons would only draw 5mA when a button was on.

3.3 Control

Verification of the control module included grounding all inputs except for one and tying the outputs to a block of LEDs. The clock and enable were controlled by the Arduino. When the enable was set low for one clock cycle, the clock would run for another 36 clock cycles and the output would be recorded on the LED block. This was done for each individual output.

One difficulty in this situation is using the LED block as a verification tool. When we integrated the control module with the MCU module, the same results were not obtained as were with the LED block. Eventually it was discovered that while the outputs of the control module were correct, the voltage levels of the outputs (~2.2V when measured with a multimeter) were not high enough to be recognized as a logical high by the MCU. This was fixed by analog reading the output of the control module and manually setting a threshold to be considered logically high.

3.4 MCU

The MCU module was verified by playing through the game. On each turn, it was visually verified that once two city buttons were pressed, the corresponding path between the two cities was illuminated. When a path between two cities was claimed, it was also verified that the correct player's score was updated and displayed to the LCD. For example, if player one claimed the path between Helena and Denver, the route between Helena and Denver should light up player one's color, their score should be increased by 7, and their number of train cars should be decremented by 4. When any of the players had less than 3 cars remaining, the game ended as expected. The longest path and destination card bonuses were verified to be working correctly by looking that the change in each player's score at the end of the game. The information displayed on the LCD was always identical to what would have been calculated by hand in the traditional version of the game.

3.5 LED

Verifying the LEDs is a visual process. The LEDs were connected to the power supply and first tested for current draw using a multimeter, which was measured at around 20mA per strip per individual color (60mA when all three LEDs were white). The LEDs were then connected to one of the data pins on the MCU. A test program was written to address different LEDs and turn them on and off or change the color to ensure they were in proper working condition before they were soldered in place. After soldering one more strip, the growing strip of connected LEDs were again tested to ensure connections were soldered well in case modifications were needed along the way. Figure 3.1 shows the LEDs in working condition in the final design.

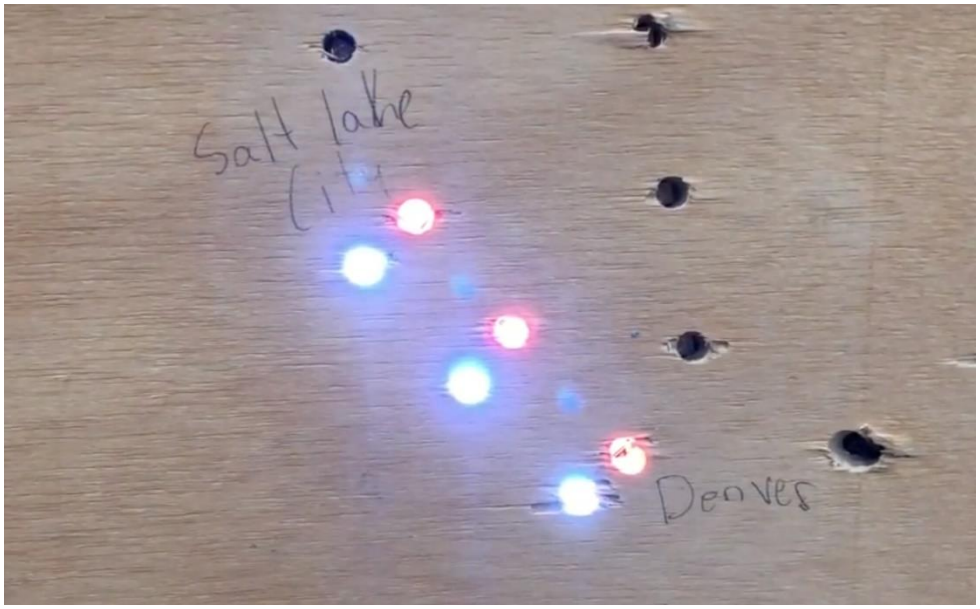

Figure 3.1: Verification of LED Module when claiming a double path

3.6 LCD

The verification for the LCD module is almost entirely visual. Once the LCD was connected to the MCU, we could display any messages desired to the screen. When playing through turns of the game, the game state information displayed was exactly what was expected. As mentioned in the power module, we measured the current draw of the LCD module to be approximately 3 mA. Note that this current draw is only for displaying characters to the screen. A backlight could be enabled to increase visibility in low-light conditions, but that would increase the current draw of the LCD. See Figure 3.2 for an example of how game state information is displayed on the LCD.

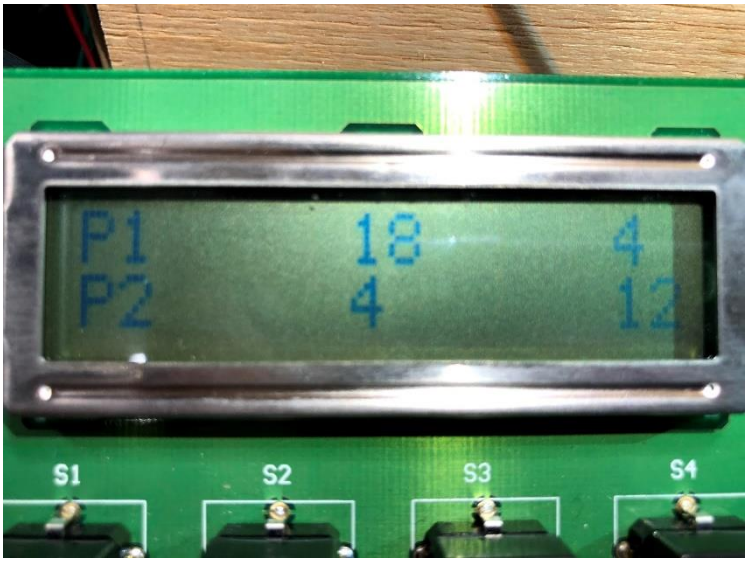

Figure 3.2: Game State Displayed on LCD

4. Costs

4.1 Parts

Table 4.1 Parts Costs

Part	Manufacturer	Quantity	Retail Cost ($)	Actual Cost ($)
Power Supply	Jhua	1	19.99	19.99
Buttons	CO-RODE	40	2.72	2.72
4-bit Shift Register (74194)	Texas Instruments	2	2.69	2.69
8-bit Shift Register (74165)	Texas Instruments	5	3.61	3.61
Control PCB	PCB Way	1	1.00	1.00
Arduino Mega 2560	Arduino	1	38.50	38.50
4-bit Counter (74161)	Texas Instruments	2	3.90	3.90
WS2811 LED Strips	IKSACE	3	44.67	44.67
LCD Display	Hitachi	1	8.95	0
1 kΩ Resistor	Vishay	40	1.47	0
Plywood 4x8 ft.	Home Depot	1	23.38	0
Total			**150.88**	**117.08**

4.2 Labor

Table 4.2 Labor Costs

Name	Total Hours	Hourly Rate ($/Hr)	Labor Total (x2.5)
Drew	**122**	**40**	**12,200.00**
Matt	**122**	**40**	**12,200.00**
Total	**244**		**24,400.00**

Table 4.1 gives a combined retail parts cost of $150.88 and table 4.2 gives a total labor cost of $24,400.00. Adding these costs together gives a total project cost of $24,550.88.

5. Conclusion

5.1 Accomplishments

Throughout the course of the semester, a successful proof of concept was developed that accomplished all the tasks outlined (See Figure 5.1). It plays a full, scaled-down version of the game complete with city buttons, LEDs on each included path, and score automation. Claiming paths, inputting destination cards, calculating bonuses, and displaying the data all work as expected. All of the elements of the design had to be working correctly in order for the scaled-down version of the game to work. It follows, then, that a full-scale version of the game should be possible using all of our components.

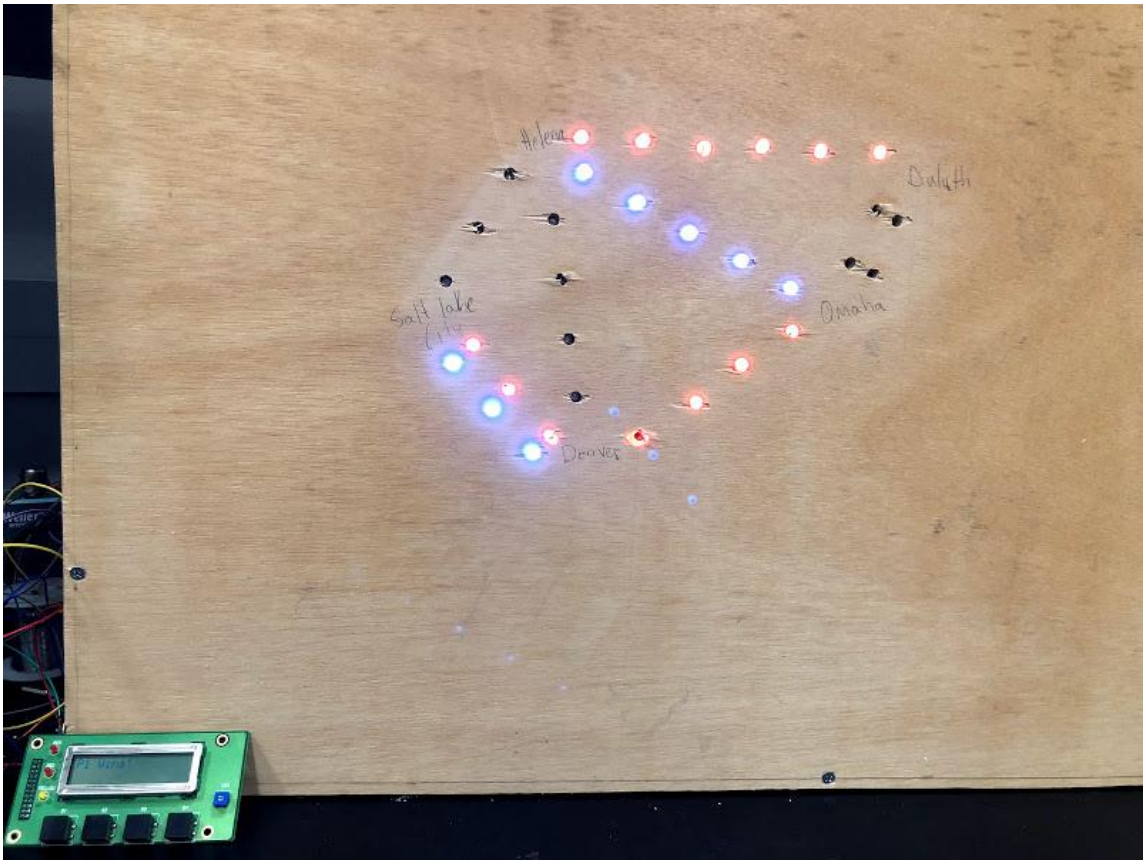

Figure 5.1: Final proof of concept.

5.2 Uncertainties

It's uncertain how some aspects of this project will scale up. One of the huge difficulties in integrating all the working components was soldering - soldering all the LEDs took an immense amount of work that could have been used to make a high quality scaled down version, like the final outcome. The huge time investment (approximately 22 man-hours) that was spent soldering the LEDs together didn't even result in having fully functional LEDs over all spaces in the game. Therefore, estimating the total amount of work that would be required to connect all the LEDs becomes very difficult. One alternative to assembling all the LED strips by hand would be to order a PCB the size of the game board and simply solder the LEDs in the appropriate place on the PCB. However, a PCB of this size would likely be prohibitively expensive. Another option would be to have individual PCBs placed under each route. These PCBs could be assembled with the LEDs already installed. Even if the LEDs are already installed in the PCB, there would still be the work of connecting these PCBs together. Other than solving the problem of installing that many LEDs, the rest of the components of the design should face no difficulty in scaling up to the full-size version of the game.

5.3 Ethical considerations

One ethical consideration would be getting a license to produce and sell this concept as a full-blown product. Ticket to Ride is of course owned by a company, so to be able to manufacture and sell a product a license must be obtained. At the moment, this product does not seem to be commercially viable. Between licensing fees and the high cost of materials and labor, the price of an automated version of Ticket to Ride would be too high for most consumers. As we do not intend on selling our system, we will avoid any ethical concerns associated with profiting off the work of others.

In the design of our project we strove adhere to the IEEE code of ethics [6]. Of special importance to was point 7 in the code of ethics: "to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others". Completing an automated scoring system for Ticket to Ride was a learning process throughout - and constructive criticism was always sought to improve our design as much as possible.

5.4 Future work

The first task in the future work would be to scale up the game. In reality, scaling up can be done with careful LED soldering, planning, and execution, as well as a lot of dedicated time to finish the manual labor. Completing the full-scale version of the game will also make apparent any components that take a long time to integrate into the greater system. This would allow us to better understand the bottlenecks that could exist if this product were to be produced again.

References

[1] "Ticket to Ride Rules" *Days of Wonder Boardgames,* https://cdn0.daysofwonder.com/tickettoride/en/img/tt_rules_2015_en.pdf

[2] "SNx4HC165 8-Bit Parallel-Load Shift Registers." *SNx4HC165 8-Bit Parallel-Load Shift Registers*, www.ti.com/lit/ds/symlink/sn74hc165.pdf.

[3] "Synchronous 4-Bit Counters" *SN74161 4-Bit Synchronous Counters, http://www.ti.com/lit/ds/sdls060/sdls060.pdf*

[4] "74HC194 4-Bit Bidirectional Universal Shift Register." *74HC194 4-Bit Bidirectional Universal Shift Register*, media.digikey.com/pdf/Data%20Sheets/NXP%20PDFs/74HC194_Rev_3.pdf.

[5] "Amtel ATmegaV-2560" *8-Bit Amtel Microcontroller,* http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

[6] "IEEE Code of Ethics." *IEEE*, www.ieee.org/about/corporate/governance/p7-8.html.

Appendix A Requirement and Verification Table

Table A System Requirements and Verifications

Module	Requirement	Verification	Verification status (Y or N)
• Power	• Output 12V tolerance of ± 0.5V from our power source. • Output at least 6A to the LEDs.	• Use a multimeter to measure output voltage. • Use a multimeter to measure current output.	Y
• User Input	• Output signals should be provided at 5V or ground with a tolerance of ± 0.5V.	• Use a multimeter to determine input and output voltage of the module.	Y
• Control	• Compress 36-bit data from the User-Input module into unique 12-bit data to be sent to the MCU.	• Use oscilloscope to compare input-output voltages from logic gates and ensure the correct compression is output.	Y
• MCU	• Decode 12-bit digital signals from the control module to calculate current scores. • Successfully calculate score, including bonuses from destination cards and longest path.	• Every combination of cities will be tested to ensure that all scorekeeping is correct. • We will display values associated with the completion of routes and the longest path in real time so we can verify our results from the MCU.	Y

	• Output game-state information to the LCD screen, such as current player, current scores, and number of cars left. • Output correct data signal to LEDs using the data pin	• Every possible input will be executed as to verify the correct output from the MCU. This is a highly visual requirement so the verification will also be visual. • A multimeter will be used to measure the color signals and ensure that the correct color is being output at all times. This is a requirement that can be verified visually as well as quantitatively.	
• LED	• Display the paths taken in the correct colors.	• Every combination of button inputs will be tested to ensure that all LEDs light up accordingly.	Y
• LCD	• Output correct messages from MCU	• This requirement, can be verified visually, as we will be able to see if the output is correct.	Y