IOWA STATE UNIVERSITY

AEROSPACE ENGINEERING DEPARTMENT

COMPUTATIONAL TECHNIQUES FOR AEROSPACE DESIGN

AERE 361

SPRING 2021

FINAL PROJECT REPORT

TEAM 10

Team Member Names :
Cvikota, Samuel
Fleming, Victoria
Johnson, Nicholas
McPhail, Clayton
Melville, Jessica
Walberg, Zachary

Contents

1 ABSTRACT

This project consists of the creation of a Blackjack card game and card counter on an Adafruit CLUE and Neokey board. Card counting is shown to improve mental math, and the mental stimulation from playing games and counting cards is engaging and can help prevent depression, Alzheimer's, and dementia. It also gives the player an advantage over the house, which can lead to earning more money from winning games. Our group developed code for the game logic of Blackjack, designed a display for cards, totals, and the card counter, and programmed various button and light functions to create an interactive, playable game on the boards. The board randomizes each hand to simulate a true Blackjack game, and allows the user to check the card counter, which carries over between hands. Overall, the developed device gives users practice at playing Blackjack and counting cards, leading to increased mental stimulation and an advantage at casinos.

2 INTRODUCTION

Counting cards in Blackjack is commonly frowned upon at casinos, to the point it can get a person banned. However, studies have shown that counting cards aids with quick mental calculations and memory, and is an application of statistics and probability [4]. That is why we created a small handheld Blackjack game to practice mental math and basic strategy in Blackjack. Our team of Nicholas Johnson, Victoria Fleming, Clayton McPhail, Jessica Melville, Samuel Cvikota, and Zachary Walberg imagined, developed and tested a prototype of a handheld game for all ages. Zachary, Nicholas, and Clayton primarily worked on the software development while Jessica, Victoria, and Samuel worked on the hardware and integration development.

Technologies such as cell phones and laptops are capable of having card counters, but it would be fairly obvious if these were being used while playing Blackjack. Therefore, it is beneficial for players who wish to count cards to learn how to do so exclusively in their heads. This project will provide an aid in the process of learning to count cards in the form of a handheld Blackjack game. Players can practice basic strategy while playing a computer. If desired, players can count cards and check their counts by using a built-in counter displayed upon request. With enough practice, players can learn how to mentally count cards, improving their odds of winning Blackjack.

3 FEATURES

Our device has three main features. The first of these is the display on the Adafruit CLUE board. The display shows the dealer and player cards with numbers and suit letters, and displays face cards as the corresponding letter. The displays also includes the player and dealer totals, as well as labels for the buttons. Finally, the current card count is included.

The next feature is the buttons on the Adafruit CLUE board. The right button displays the card counter, and it can be pressed whenever the user can make a choice in the game to hit or split. The left button clears the card counter by resetting the card display, allowing the user to glance at the counter and remove it before moving on in the game. Initially, these buttons were going to be used to display the card counter and end the game, however, we had difficulty implementing a way to exit the game due to the infinite loop that is run in Arduino. We decided to use the other button to clear the card counter, as then there is no delay needed between displaying and clearing the counter, which was needed when using only one button.

Finally, the Adafruit Neokey board was used to provide additional buttons. The top two buttons on the board allow the user to hit or split. The keys light up white when pushed, which goes away when the game ends or another user input is requested, showing which button was pressed by the user. The keys also light up red if the player loses the game, and green if the player wins. Initially, there was supposed to be a split button on this board as well. However, we were unable to incorporate split logic into the game, as we decided to focus on fixing issues with the Ace logic and polishing the game integration with the two boards. We also decided to use the lights on the Neokey board rather than the light on the CLUE board to signal the player winning or losing, as it is much more obvious to see the key lights than the light on the back of the CLUE board when playing the game.

4 PROBLEM STATEMENT

Like most card based games, Blackjack involves a certain amount of mental math and probability. Being able to assess the hands on the table and make quick decisions can increase the odds of winning over the dealer. This ability to do quick mental math is used everywhere in our lives to solve problems or make estimations. Like any skill, mental math must be practiced, and what better way to practice than by incorporating the practice into a game. Studies have shown that practicing a skill by using a game is more effective because “fun motivates students and helps them pay attention and stay focused on the subject” [7]. By using a game as a way to teach, people see the material as less of a “chore” to learn. Additionally, people are

able to see and apply what they are learning in real time.

The act of practicing mental math can also sharpen ones mental capacity. By consistently performing simple math operations, both mental capacity and health can be improved. The act of repeating these actions can help fight depression and anxiety as seen in a study done at Duke University. 186 students were asked to perform mental math problems while their brain was under observation. While performing these operations, areas of the brain that deal with depression and anxiety were stimulated. Increased activity in this part of the brain has been seen to lowered depression and anxiety levels in students [5]. In addition to fighting depression, mental stimulation can also decrease the likelihood of developing Alzheimers or Dementia. The most common cause of mental decline is boredom, routine, and lack of challenging activities to do, and continuously performing these mental calculations can help fight that [6].

5 PROBLEM SOLUTION

5.1 Software

In order to solve our problem, we had to break it into manageable steps. Our first step was how to keep track of all of the cards in the program. Our solution to this issue was to create an array to hold 1 deck of cards. This array had multiple columns; *Card Number, Card, Card Value, Number of Times Drawn.* The card number was simply a number 1 through 52 that corresponded to a row in the matrix that could then be called to select a card. The card column held the face of the card numbered 1-13 with a 1 corresponding to an ace and a 13 corresponding to a king. The card value held the point value for the card according to the rules of blackjack. The last column held the number of times that particular card had been drawn–this was to be used in order to make sure no card was drawn more times than the number of decks we were playing with. We stored the initial values in a CSV file and used an `fscanf` statement to read them into a matrix before the game started. We chose to use the `fscanf` statement over a more complicated csv parser because we knew the input that would be coming into the game and could therefore format 1 line of code easier than copying or creating a parser.

Once that was input we had to figure out how to deal the cards. We knew we wanted a random number generator [3] and through some googling we were able to find and code a random number generator shown in listing 1. This was how we picked which cards to draw and returned the values so that they could be pulled from the matrix. This was the most integral part of the solution as it was required for any kind of functionality to be possible.

```
upper = 52;
lower = 1;

srand(time(NULL));

for(i = 0; i < count; i++){
    draw = (rand() % (upper - lower + 1)) + lower;
    //printf(" %d",draw);
}
//printf("\n");

return draw;
```

Listing 1: Random Number Generator

The Next challenge was to track the cards in both the players and dealers hand. This was relatively easily accomplished by splitting the the dealt cards and assigning them to a unique array for either the dealer or player. Once the cards were in their respective hands the point values of those cards were added together for each. These were statically allocated arrays. As we moved forward we realized that we needed a good way to allow us to call these cards easily and without creating too many more variables. At one point we were using too much memory and getting segmentation faults at the start of the program. This was fixed by changing most of the `integer` variable types to `short` variable types. To deal with the passing of variables we decided to make the card array and the player array–among a few other variables–into global variables declared outside of the main function. The cards were then printed using our print function that is shown in listing 4. Once we had dealt with the tracking and passing of cards, we needed to institute the game-play logic that would allow the user to play the game correctly.

The start of this was the players turn. In order to handle this we used an option that would allow the user to input whether they wanted to hit or stand–this was done with an option to hit one of two buttons when the code was moved to the CLUE board. If the user chose to hit the draw function was called and the new card was put into the players hand. The card was called from the array and added to the player's total. The total was then checked using a series of if statements to determine how the program was to proceed. The series of `if` statements is shown in listing 2. If the player's total is not greater than or equal to 21 it gives the player the choice to hit, stand, or check the card count. All of this was done inside a `do while` loop which had an exit condition triggered if the player chose to stand, busted, or drew cards equal to the point value of 21.

```
if(player_total > 21){
                red();
```

```
                printf("Player Busts!\nDEALER WINS\n");
                reset();
                sleep(2);
                player_done = true;
                dealer_done = true;
                break;
            }
            else if(player_total == 21){
                printf("Player at 21!\n\tDEALER PLAYS OUT\n");
                sleep(3);
                break;
            }
        }
        else if(input == 0){
            printf("\e[1;1H\e[2J");
            printf("\tDEALER PLAYS OUT\n");
            sleep(1);
            player_done = true;
        }
        else if(input ==2){
            printf("Current High-Lo value: %d\n",current_count);
        }
        else{
            printf("Not valid input! Try again!\n");
        }
```

Listing 2: Player Choice Logic

The biggest challenge we faced in this project was dealing with what happens when an Ace is in play. In our code an Ace was automatically played as an 11, so we had to add logic to reduce the value of the card from 11 to 1 if the total was over 21. We did this with a simple for loop that checked the players hand for an ace if the total was 21. If the condition was met the total would be reduced by 10 and it would run through the `if` statements with the new total. Once the player stood or hit 21 it was then the dealers turn to play.

The logic for the dealer playing was very similar to the player logic in structure, however it had a very different set of conditions. At most blackjack tables the dealer is required to hit if their total is less than 17 and must stand if their total is greater than 17. The code would check the dealer's total to ensure it was under 21. If it was it would then go through a series of `if` and `else if` statements–shown in listing 3–to determine whether the dealer was required to hit or stand, while simultaneously checking its total against the player's total to determine the winning condition. After each card was drawn as the dealer played the print function would be called and the screen updated.

```
if(dealer_total <= 21){
        if(dealer_total == 21){ W
            red();
            printf("Dealer HIT 21!\n\tDEALER WINS\n");
            reset();
            dealer_done = true;
        }
        else if(dealer_total >= 17 && dealer_total >= player_total){
            red();
            printf("Dealer's total greater than player's!\n\tDealer
    WINS\n");
            reset();
            dealer_done = true;
        }
        else if(dealer_total >= 17 && dealer_total < player_total){
            green();
            printf("Player's total greater than dealer's!\n\tPlayer
    WINS\n");
            reset();
            dealer_done = true;
        }
        else if(dealer_total==17){
            for(i=0;i< dealer_cards;i++){
                if(dealer_hand[i]==11){
                    dealer_total -= 10;
                }
            }
        }
      else if(dealer_total < 17){
          new_card = card_draw(1) - 1;
        dealer_hand[dealer_cards] = cards[new_card][1];
        dealer_total += cards[dealer_hand[dealer_cards]][2];
            for(i = 0; i < dealer_cards; i++){
                if(cards[dealer_hand[i]][2] == 11 && dealer_total > 21)
    {
                    dealer_total -= 10;
                }
            }
            sleep(1);
        dealer_cards++;
        print_cards(player_total,player_cards,dealer_total,dealer_cards
    ,player_done,current_count);
      }
```

Listing 3: Dealer Choice Logic

The print function was mentioned multiple times previously. It was a function outside main that we would call when we wanted to display all of the current game-

play information on the terminal or screen. Separate for loops were used to print the dealer and player hands. It would loop through and check the card number to determine the face of the card and if it was a face card it would call a second function to print the characters for the face cards and aces. After printing either the face character or the number on the card it would print a suit based on the 1-52 value of the card. The print function is shown in listing 4.

```
printf("Dealer: ");
    for(i = 0; i < dealer_cards; i++){
        card = cards[dealer_hand[i]][1];
        if(dealer_cards == 2 && i == 0){
            printf("? ");
        }
        else{
            if(card == 1 || card > 10){
                high_val_print(card);
                suit_print(cards[dealer_hand[i]][0]);
            }
            else{
                printf("%d ",cards[dealer_hand[i]][1]);
                suit_print(cards[dealer_hand[i]][0]);
            }
        }
    }
    if(dealer_cards == 2 && player_done == false){
        printf("\nDealer Total: %d\n\n",cards[dealer_hand[1]][2]);
    }
    else{
        printf("\nDealer Total: %d\n\n",dealer_total);
    }
    printf("Player: ");
    for(i = 0; i < player_cards; i++){
        card = cards[player_hand[i]][1];
        if(card == 1 || card > 10){
            high_val_print(card);
            suit_print(cards[player_hand[i]][0]);
        }
        else{
            printf("%d ",card);
            suit_print(cards[player_hand[i]][0]);
        }

    }
    printf("\nPlayer Total: %d\n",player_total);
```

Listing 4: Printing Function

The `high_val_print` function was what printed the higher value cards. It used

a switch case statement based on the Card that was drawn with 1 being an ace and 13 being a king. It simply printed the correct character instead of printing the card number from the array. Directly after the number was printed a function to print the suit of the card was called. This function also simply printed a character corresponding to the suit of the card as shown in table 5.1.

Card	**Suit**
1-13	Hearts
14-26	Clubs
27-39	Diamonds
39-52	Spades

Table 1: Card Suits

Our last issue to overcome was the card counter. This one did turn out to be a simple inclusion. The called function would evaluate the card based on our chosen hi-lo system and assign it the correct value of -1, 0, or 1. The function was simply called and given a card, it then returned the assigned card count value for that card and it was summed into the count.

In order to make the game easier to read from the display, we used colored text in the terminal and on the board to display the outcome of the game; using red for a loss, green for a win and yellow for a push. These color change functions [2] were found through the help of google and extremely useful.

5.2 Hardware

When creating the hardware code, examples from the Adafruit Arcada [1] were used. When creating the display, we used an initial layout from our proposal and made modifications as we saw fit. We used the functions from the C code to return the suit and face card characters, and added in the necessary Arcada print statements for displaying the cards, totals, card count, dividers, and labels. The display layout can be seen in Figure 1.

The card counter on the display can be seen in Figure 2.

We first tested the display and button code on the CLUE board by using hard-coded cards, totals, and the card count. This allowed us to make sure the functions were displaying the values passed in correctly. Once the C code was integrated, we were able to check the cards, totals, and count with each other to make sure the correct values were being displayed.

When working with the Neokey board, examples from the Adafruit Seesaw library were used Citeseesaw. We decided to use the lights on this board rather than the

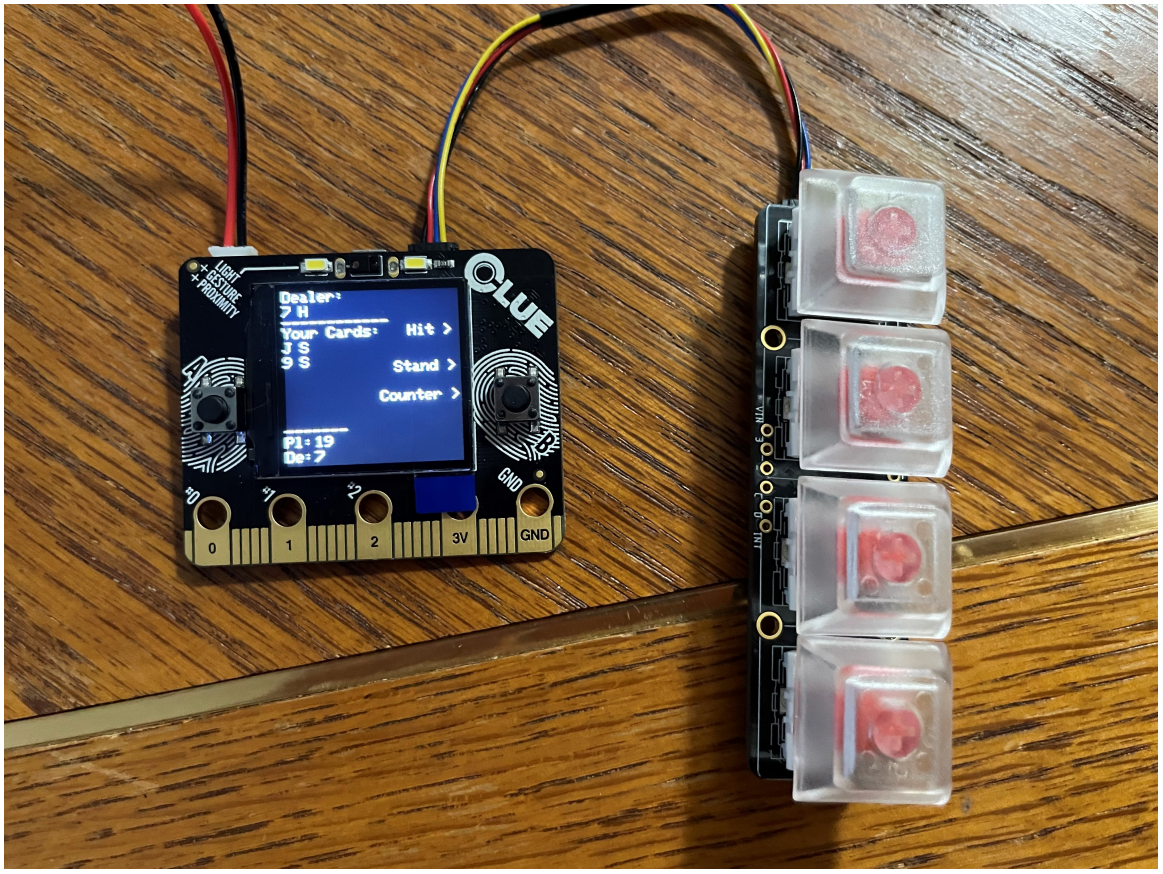

Figure 1: Display layout on CLUE board

one on the CLUE board due to increased visibility. We also used the top two buttons for the hit and stand actions, and those keys light up when pressed. The hit and stand buttons can be seen in Figures 3 and 4.

The win/lose scenarios can be seen in Figures 5 and 6.

We first tested the display and button code on the Neokey board and for the game-ending conditions by hard coding the win/lose status and by having the buttons on the Neokey board trigger actions like displaying and clearing the card count. This allowed us to check that the lights and buttons were functioning when pressed, and that the win/lose condition on the display matched the lights. Once the C code was integrated, we were able to see how the game played out when each button was pressed and use the totals to check the win/lose conditions with the lights and display.

We think that the solution with the CLUE and Neokey boards is a good solution.

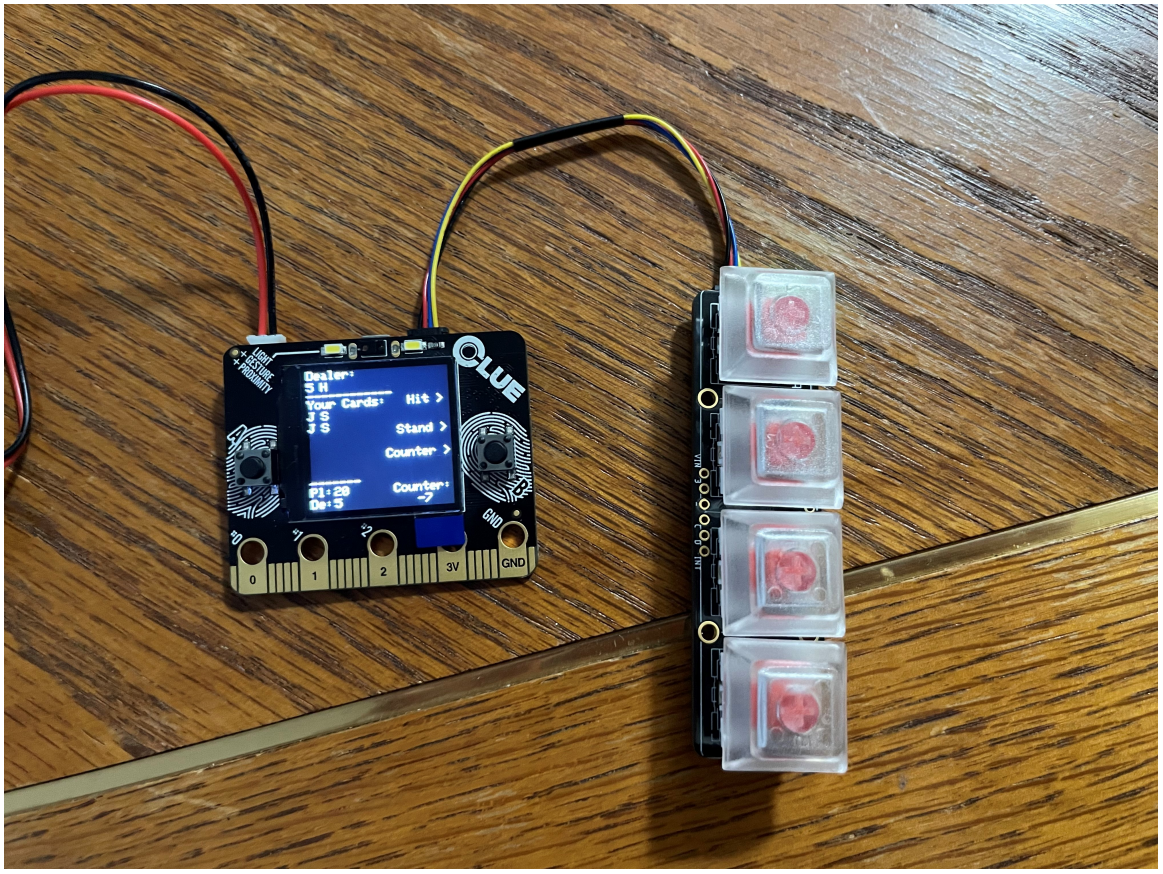

Figure 2: Card counter on CLUE board display

The cards, totals, labels, and counter on the display are clear, as well as the win/lose conditions. The Neokeys are easy to use and it is obvious which button was pressed by the user. The only downside of using the CLUE board that way we did is that the text on the display cannot be appended, so the whole screen must be reset to clear old cards or the card counter.

6 STATUS

6.1 Software

The project on the side of software development was a complete success. The software is at a point where the game can be played and it can be recognized as Blackjack. Although the basic functions and rules are programmed correctly, it was not possible to get the more advanced functions of the game included within the time-

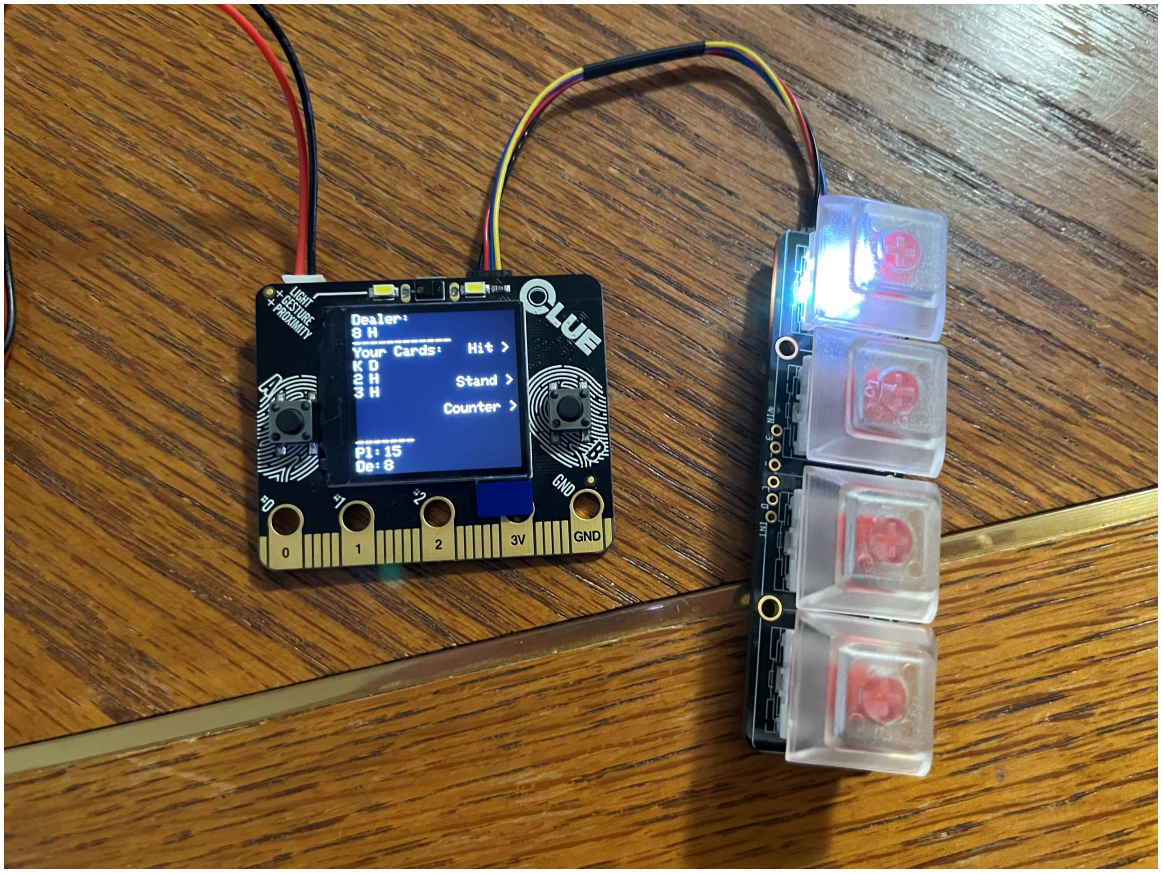

Figure 3: Hit key with lights

frame allotted for the project. It was one thing to get the game debugged everytime we added a feature, but it was another task to get it integrated successfully with the hardware, which took more time than adding the feature in the first place at times. Due to this abundant amount of time it took to add more functions and rules, we had to sacrifice more advanced features in order to have a working game by the end of the semester.

6.2 Hardware

The integration of the NeoKey Buttons and the software into the Adafruit CLUE board was successful and yielded a fully functioning Blackjack game. We achieved a game that played against the dealer with buttons for inputs such as revealing and clearing the card counter, the hit button, and the stand button. In addition, visuals were added to better inform the user if they had won or lost a hand. This was achieved by NeoKey lighting up all red on a loss and all green on a win in addition

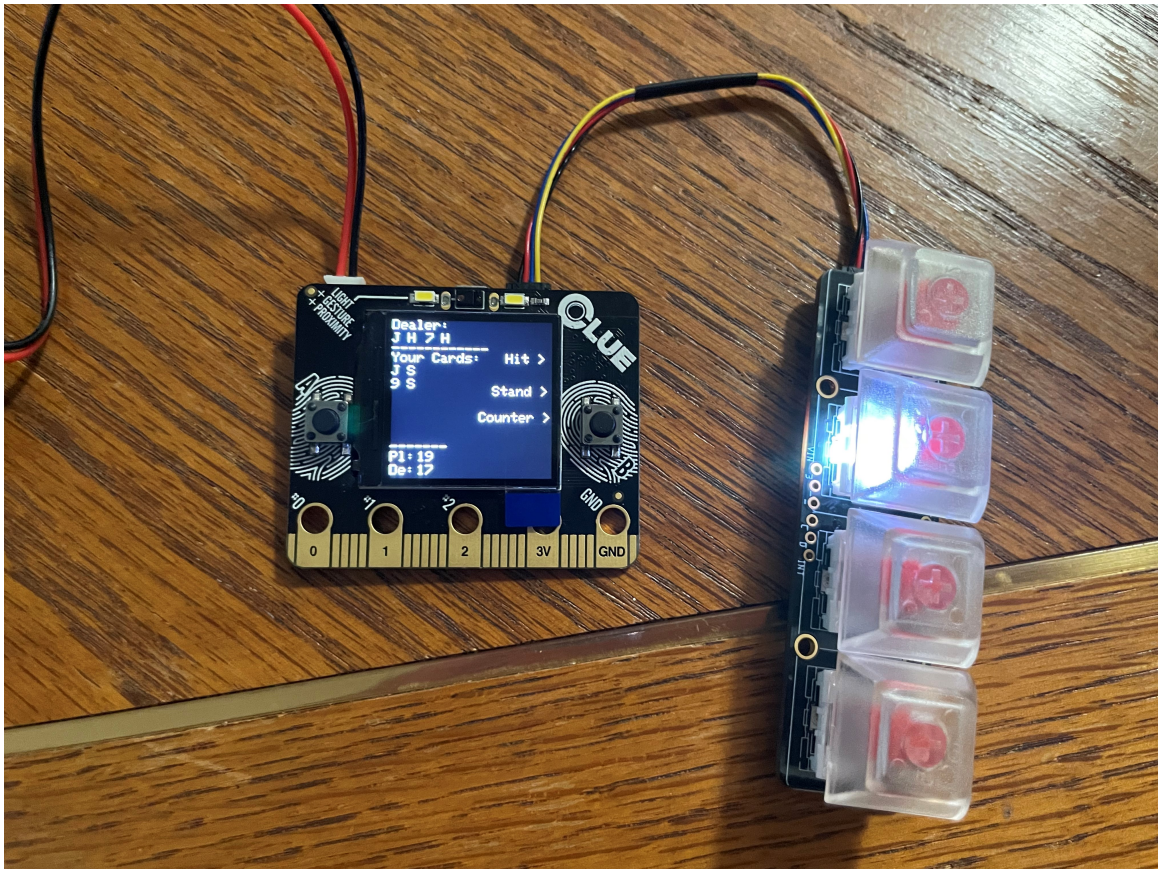

Figure 4: Stand key with lights

to a win or lose message displayed on the board.

Three things that could be worked on would be adding a way to completely exit the game, split logic, and having a random set of card hands. Exiting the game was rather challenging because Arduino runs the main code in an infinite loop. While an attempt was made at writing logic to exit the game, it was unsuccessful, and we chose to direct our efforts towards other issues. The game will also deal a specific set of hands throughout the game and if the CLUE board is reset without a recompile, the same hands will be played over again. We attempted to seed the random number generator to try and change this, however, we did not get it working. We also did not add in the split logic to the hardware because it was not incorporated into the software. Overall, we were able to create a game that a player could play against the computer with a set win or lose scenario.

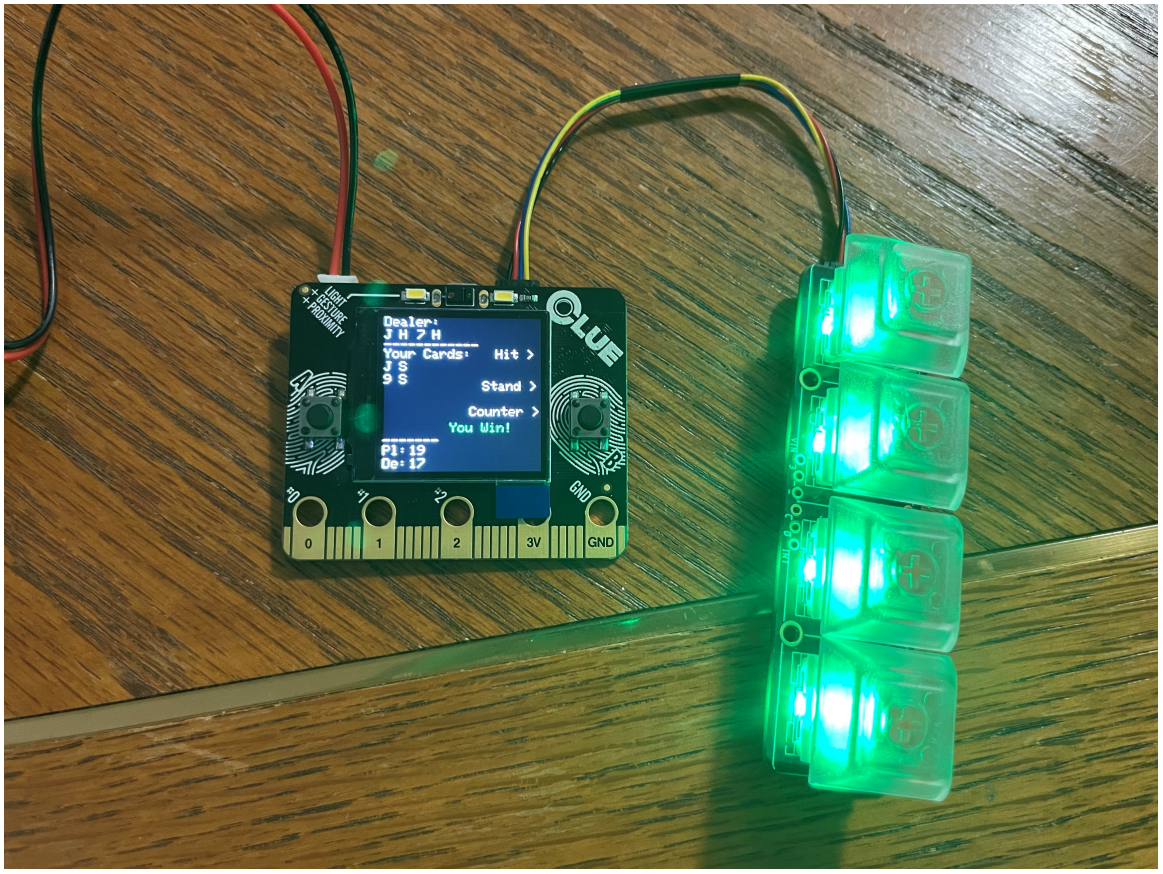

Figure 5: Win display and lights

6.3 Lessons Learned

One of the skills our group improved during this project was communication. There was not a lot of time for the project, so we had to divide and conquer between the hardware and software. This required being able to communicate about our code with the other team so that any necessary adjustments or bugs could be fixed. We also had to learn how to find sources for the Adafruit boards, as there are not a lot of resources on how to use the libraries we selected besides the examples. Therefore, we had to be able to pick out the necessary information from the code examples and the header file information to call the classes and functions we needed.

Expanding on the time management, although we did not focus a lot of time on the project over break and before, the last few weeks up until the presentation we were able to effectively set up meetings and split up work to get specific parts of the presentation and code done. Where we could have improved on this was better utilizing the time before break to get most of the initial code done to allow the

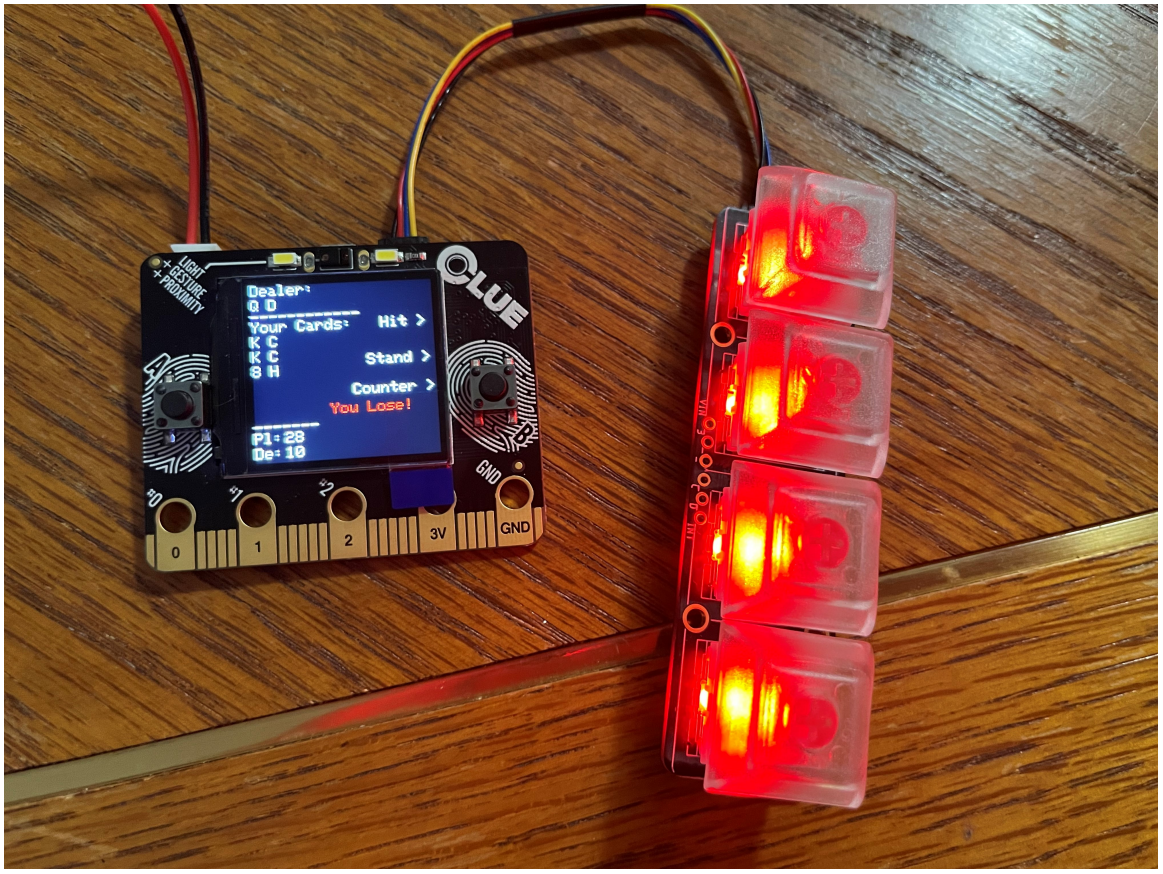

Figure 6: Lose display and lights

hardware side more time to convert from C to the arduino and iron out any of the bugs that came with it. Doing this might have allowed us to integrate aspects of the game that were left out of the final version such as splitting and more decks, which would have increased the overall effectiveness of the card counting and simulating a realistic Blackjack game.

7 RESULTS

As our device does not collect data, our only results are reactions from others. Overall, we were complemented by our classmates and instructors on our implementation of the Blackjack game and card counter.

Our Github repository can be found here: `https://github.com/Zwalberg/AerE-361-Final-Project`. The code for the project is located in the src folder. The C files contain function declarations, while all of the function bodies and hard-

ware code can be found in Project Hardware Code.ino.

8 FUTURE WORK

If we were to continue working on this project, we would implement the split logic into the Blackjack game. The split rule applies at the start of the game; If a player's starting two cards have the same value, they can decide to split their hand and play on each card as a new hand. We believe this is definitely possible to implement if given more time. The first main issue for this rule's implementation was that it was hard to build a more advanced game function into an already more advanced game-code structure without breaking what we already had. Another big issue was that this rule changed the display shown to the user, which would have taken a lot of time to build into and debug on the hardware.

Next, we would work on getting the game to have an exit button, ideally on the Neokey board. This would cause the game to stop looping after the current hand ends. We believe this could be done using an if statement to check if the button has been pressed, and if so, the game code is not run in the Arduino loop. We would also work on trying to generate different sets of hands, which we believe may be possible by seeding the random number generator. While we did try this, we didn't have a lot of time to work with it, and believe more time may solve the issue.

Another addition to the project would be to have a more structured device. As of right now, the clue board and Neokey board are held separately. To increase the player's experience, it would be beneficial to add a 3D printed case to model a modern day handheld game device. The clear reason for why this was not included up to this point is because it is more important to have a working game rather than improving aesthetics, but we believe it would be feasible given more time.

9 CONCLUSION

The goal of this project was to create a functioning game of Blackjack with a card counter with the intent to teach new players not only how to just play the game but to teach them how to play in a way that increases their odds of winning. Although some high-level functions had to be left out of the current version of the game, we believe that the project was a success in the aspect of what was accomplished within the given timeframe.

A lot was accomplished during this project, but even more was learned. There were a lot of unknowns going into the project on both the software and hardware side.

The group was able to play to each other's strength and learn new things in order to accomplish a common goal.

References

[1] *Adafruit Arcada Repository*. URL: https://github.com/adafruit/Adafruit_Arcada.

[2] "Adding Color to Your Output From C". In: ().

[3] "C library function - srand()". In: (). URL: https://www.tutorialspoint.com/c_standard_library/c_function_srand.htm.

[4] Brian Crafton, Samuel Spetalnick, and Arijit Raychowdhury. "Counting Cards: Exploiting Variance and Data Distributions for Robust Compute In-Memory". In: *arXiv preprint arXiv:2006.03117* (2020).

[5] DukeU. "Could mental math boost emotional health?" In: *EurekAlert!* (). URL: https://www.eurekalert.org/news-releases/820759.

[6] Ashton Turner. *Is playing card games good for our brain?* June 2019. URL: https://www.cnsuwo.ca/is-playing-card-games-good-for-our-brain/.

[7] *Why use games to teach?* May 2018. URL: https://serc.carleton.edu/introgeo/games/whygames.html.

A SOURCE CODE

```
/*Code Sources:
Arcada graphicstest (Display, Buttons)
*/
#include<stdio.h>
#include<cstdlib>
#include<string.h>
#include<time.h>
#include<unistd.h>
#include "Adafruit_Arcada.h"
//#include "game.c"
#include <Adafruit_NeoKey_1x4.h>
#include <seesaw_neopixel.h>
Adafruit_Arcada arcada;
Adafruit_NeoKey_1x4 neokey;

NeoKey1x4Callback blink(keyEvent evt) {
  uint8_t key = evt.bit.NUM;

  if (evt.bit.EDGE == SEESAW_KEYPAD_EDGE_RISING) {
    Serial.print("Key press ");
    Serial.println(key);
    neokey.pixels.setPixelColor(key, Wheel(map(key, 0,
   neokey.pixels.numPixels(), 0, 255)));

  } else if (evt.bit.EDGE == SEESAW_KEYPAD_EDGE_FALLING) {
    Serial.print("Key release ");
    Serial.println(key);

    neokey.pixels.setPixelColor(key, 0);
  }

  // Turn on/off the neopixels!
  neokey.pixels.show();
  return 0;
}

void setup(void) {
  Serial.begin(9600);
  Serial.begin(115200);
```

```
39    arcada.arcadaBegin();
40    arcada.displayBegin();
41    arcada.setBacklight(255);
42    arcada.display->fillScreen(ARCADA_BLACK);
43    arcada.display->setTextColor(ARCADA_WHITE);
44    //while (! Serial) delay(10);
45    if (! neokey.begin(0x30)) {     // begin with I2C address
      , default is 0x30
46      Serial.println("Could not start NeoKey, check wiring?")
      ;
47      while(1) delay(10);
48    }
49    Serial.println("NeoKey started!");
50    // Pulse all the LEDs on to show we're working
51    for (uint16_t i=0; i<neokey.pixels.numPixels(); i++) {
52      neokey.pixels.setPixelColor(i, 0x808080); // make each
      LED white
53      neokey.pixels.show();
54      delay(50);
55    }
56    for (uint16_t i=0; i<neokey.pixels.numPixels(); i++) {
57      neokey.pixels.setPixelColor(i, 0x000000);
58      neokey.pixels.show();
59      delay(50);
60    }
61  }
62
63  void loop() {
64      game();
65  }
66
67  uint32_t Wheel(byte WheelPos) {
68    if(WheelPos < 85) {
69     return seesaw_NeoPixel::Color(WheelPos * 3, 255 -
      WheelPos * 3, 0);
70    } else if(WheelPos < 170) {
71     WheelPos -= 85;
72     return seesaw_NeoPixel::Color(255 - WheelPos * 3, 0,
      WheelPos * 3);
73    } else {
```

```
74    WheelPos -= 170;
75    return seesaw_NeoPixel::Color(0, WheelPos * 3, 255 -
      WheelPos * 3);
76   }
77 }
78
79 void print_cards(int player_total,int player_cards,int
      dealer_total,int dealer_cards,int play,short dealer_hand
      [], short player_hand[],int cards[][4]) {
80 int card, card_num;
81   arcada.displayBegin();
82   arcada.setBacklight(255);
83   arcada.display->fillScreen(ARCADA_BLACK);
84   arcada.display->setTextColor(ARCADA_WHITE);
85  arcada.display->setTextSize(2);
86  arcada.display->setCursor(0,0);
87  arcada.display->println("Dealer:");
88  for(int i = 0; i < dealer_cards; i++){
89  arcada.display->setCursor(50*(i),20);
90  if (dealer_cards == 1) {
91   card = cards[dealer_hand[1]][1];
92   card_num = cards[dealer_hand[1]][0];
93  }
94  else {
95  card = cards[dealer_hand[i]][1];
96  card_num = cards[dealer_hand[i]][0];
97  }
98  if(card == 1 || card > 10){
99     high_val_print(card);
100 }
101 else {
102 arcada.display->println(card);
103 }
104 arcada.display->setCursor(50*(i)+23,20);
105  suit_print(card_num);
106 }
107  arcada.display->println("------------");
108  arcada.display->setCursor(0,50);
109  arcada.display->println("Your Cards:");
110  arcada.display->setCursor(170,50);
```

```
111   arcada.display->println("Hit >");
112   arcada.display->setCursor(150,100);
113   arcada.display->println("Stand >");
114   arcada.display->setCursor(130,140);
115   arcada.display->println("Counter >");
116   for(int i = 0; i < player_cards; i++){
117   arcada.display->setCursor(0,70+20*i);
118   int card = cards[player_hand[i]][1];
119   if(card == 1 || card > 10){
120     high_val_print(card);
121   }
122   else {
123     arcada.display->println(card);
124   }
125   arcada.display->setCursor(0+23,70+20*i);
126   card_num = cards[player_hand[i]][0];
127   suit_print(card_num);
128   }
129   arcada.display->setCursor(0,185);
130   arcada.display->println("-------");
131   arcada.display->setCursor(0,200);
132   arcada.display->println("Pl:");
133   arcada.display->setCursor(40,200);
134   arcada.display->println(player_total);
135   arcada.display->setCursor(0,220);
136   arcada.display->println("De:");
137   arcada.display->setCursor(40,220);
138   arcada.display->println(dealer_total);
139 }
140 void high_val_print(short face){
141     char card_face[1] = {0};
142     /*yellow();
143     printf("\tDEBUG: face = %d\n",face);
144     reset();*/
145     switch (face){
146         case 1: card_face[0] = 'A'; break;
147         case 11: card_face[0] = 'J'; break;
148         case 12: card_face[0] = 'Q'; break;
149         case 13: card_face[0] = 'K'; break;
150         default: card_face[0] = 'N'; break;
```

```
151     }
152     arcada.display->println(card_face[0]);
153 
154 }
155 
156 //void carddisplay() {
157 //size_t dealer_size = sizeof(dealer_hand)/sizeof(short);
158 //for(int i=0;i<dealer_size;i++) {
159 //  arcada.display->setTextSize(2);
160 //  arcada.display->setCursor(0,0);
161 //  arcada.display->println("Dealer:");
162 //  arcada.display->setCursor(0,15*i);
163 //  arcada.display->println(dealer_hand[i]);
164 //  arcada.display->println("------------");
165 //}
166 // size_t player_size = sizeof(player_hand)/sizeof(short);
167 // for(int i=0;i<player_size;i++) {
168 //   arcada.display->setCursor(0,50);
169 //   arcada.display->println("Your Cards:");
170 //   arcada.display->setCursor(0,60+15*i);
171 //   arcada.display->println(player_hand[i]);
172 //   arcada.display->setCursor(0,165);
173 //   arcada.display->println("---");
174 //   arcada.display->setCursor(0,180);
175 //   //arcada.display->println(player_total);
176 //   arcada.display->setCursor(160,165);
177 //   arcada.display->println("---");
178 //   arcada.display->setCursor(160,180);
179 //   //arcada.display->println(dealer_total);
180 // }
181 // }
182 
183 void counterdisplay(int current_count) {
184   arcada.display->setCursor(140,200);
185   arcada.display->println("Counter:");
186   arcada.display->setCursor(180,220);
187   arcada.display->println(current_count);
188 }
189 
190 // void windisplay() {
```

```
//     arcada.display->setTextColor(ARCADA_GREEN);
//     arcada.display->setTextSize(5);
//      arcada.display->setCursor(100,160);
//     arcada.display->println("You Win!");
//     neokey.pixels.setPixelColor(0, 0x00FF00); // red
//     neokey.pixels.setPixelColor(1, 0x00FF00); // red
//     neokey.pixels.setPixelColor(2, 0x00FF00); // red
//     neokey.pixels.setPixelColor(3, 0x00FF00); //
//     neokey.pixels.show();
// }

// void losedisplay() {
//     arcada.display->setTextColor(ARCADA_RED);
//     arcada.display->setTextSize(5);
//     arcada.display->setCursor(100,160);
//     arcada.display->println("You Lose!");
//     neokey.pixels.setPixelColor(0, 0xFF0000); // red
//     neokey.pixels.setPixelColor(1, 0xFF0000); // red
//     neokey.pixels.setPixelColor(2, 0xFF0000); // red
//     neokey.pixels.setPixelColor(3, 0xFF0000); //
//     neokey.pixels.show();
// }

int game(){
  int cards[52][4] = {
      {1,1,11,0},
      {2,2,2,0},
      {3,3,3,0},
      {4,4,4,0},
      {5,5,5,0},
      {6,6,6,0},
      {7,7,7,0},
      {8,8,8,0},
      {9,9,9,0},
      {10,10,10,0},
      {11,11,10,0},
      {12,12,10,0},
      {13,13,10,0},
      {14,1,11,0},
      {15,2,2,0},
```

```
231         {16,3,3,0},
232         {17,4,4,0},
233         {18,5,5,0},
234         {19,6,6,0},
235         {20,7,7,0},
236         {21,8,8,0},
237         {22,9,9,0},
238         {23,10,10,0},
239         {24,11,10,0},
240         {25,12,10,0},
241         {26,13,10,0},
242         {27,1,11,0},
243         {28,2,2,0},
244         {29,3,3,0},
245         {30,4,4,0},
246         {31,5,5,0},
247         {32,6,6,0},
248         {33,7,7,0},
249         {34,8,8,0},
250         {35,9,9,0},
251         {36,10,10,0},
252         {37,11,10,0},
253         {38,12,10,0},
254         {39,13,10,0},
255         {40,1,11,0},
256         {41,2,2,0},
257         {42,3,3,0},
258         {43,4,4,},
259         {44,5,5,0},
260         {45,6,6,0},
261         {46,7,7,0},
262         {47,8,8,0},
263         {48,9,9,0},
264         {49,10,10,0},
265         {50,11,10,0},
266         {51,12,10,0},
267         {52,13,10,0},
268     };
269     short deal[4];
270     short player_hand[6];
```

```
271     short dealer_hand[6];
272     short current_count;
273     short i,j,player_total,dealer_total,new_card;
274     int a,b,c,d,input;
275     char cards1[100];
276     bool player_done = false,dealer_done = false,play =
    true,excon = false;
277     //FILE * fp;
278     /*
279     fp = fopen("Card_List.csv","r");
280     if (fp == NULL){
281         fprintf(stderr,"Card list did not open correctly.\
    nEXITING PROGRAM\n");
282         exit(3);
283     }
284     */
285     /*
286     do{
287         fscanf(fp,"%d,%d,%d,%d",&a,&b,&c,&d);
288         cards[i][0] = a;
289         cards[i][1] = b;
290         cards[i][2] = c;
291         cards[i][3] = d;
292         i++;
293     }while(fgets(cards1,100,fp) != NULL);
294     */
295
296     printf("Welcome to Blackjack\n\n");
297 do{
298     player_done = false;
299     dealer_done = false;
300     player_total = 0;
301     dealer_total = 0;
302     printf("\e[1;1H\e[2J");
303     for (i = 1; i <= 4; i++){
304         int index = i-1;
305         //printf("Card: %d\n",card_draw(i));
306         deal[index] = card_draw(i) - 1;
307         //printf("deal[%d] = %d\n",index,deal[index]);
308         //printf("\t%d %d\n",cards[deal[index]][1],cards[
```

```
    deal[index]][2]);
309
310     }
311
312     //printf("Cards dealt: %hd\n",cards_dealt);
313
314     player_hand[0] = cards[deal[0]-1][0];
315     player_hand[1] = cards[deal[2]-1][0];
316     dealer_hand[0] = cards[deal[1]-1][0];
317     dealer_hand[1] = cards[deal[3]-1][0];
318
319     //printf("Player hand: %d  %d\n",player_hand[0],
    player_hand[1]);
320
321     for(i = 0; i < 2; i++){
322       player_total += cards[player_hand[i]][2];
323       current_count += checkVal(cards[player_hand[i]][2]);
324       dealer_total += cards[dealer_hand[i]][2];
325     }
326     current_count +=checkVal(cards[dealer_hand[0]][2]);
327         if(player_total > 21){
328                 player_total -= 10;
329             }
330
331     int player_cards = 2;
332     int dealer_cards = 2;
333     print_cards(player_total,player_cards,cards[dealer_hand
    [1]][2],1,play,dealer_hand,player_hand,cards);
334     delay(500);
335     if(player_total == 21 && dealer_total < 21){
336         green();
337         printf("BLACKJACK!\nPLAYER WINS!\n");
338         //reset();
339         player_done = true;
340         dealer_done = true;
341     }
342     else if(dealer_total == 21){
343         red();
344         printf("Dealer has BLACKJACK\nDEALER WINS!\n");
345         //reset();
```

```
346         player_done = true;
347         dealer_done = true;
348      }
349      else if(player_total == 21 && dealer_total == 21){
350         printf("Player and Dealer both have blackjack\nIt
    is a PUSH!\n");
351         player_done = true;
352         dealer_done = true;
353      }
354      //print_cards(player_total,player_cards,cards[
    dealer_hand[1]][2],1,play,dealer_hand,player_hand,cards)
    ;
355      do{
356        //stores button presses
357         uint8_t buttons_pressed = arcada.readButtons();
358        //if button B is pushed, display the card counter
359        if (buttons_pressed & ARCADA_BUTTONMASK_B) {
360          counterdisplay(current_count);
361        }
362        //if button a is pushed, remove the card counter
363        if (buttons_pressed & ARCADA_BUTTONMASK_A) {
364          arcada.displayBegin();
365          arcada.setBacklight(255);
366          arcada.display->fillScreen(ARCADA_BLACK);
367          arcada.display->setTextColor(ARCADA_WHITE);
368          print_cards(player_total,player_cards,cards[
    dealer_hand[1]][2],1,play,dealer_hand,player_hand,cards)
    ;
369        }
370         uint8_t buttons = neokey.read();
371          printf("\tEnter 1 to HIT or 0 to STAND: ");
372          scanf(" %d",&input);
373          neokey.pixels.setPixelColor(0, 0);
374          neokey.pixels.setPixelColor(1, 0);
375          neokey.pixels.show();
376          if(buttons & (1<<0)){
377              Serial.println("Button A");
378              neokey.pixels.setPixelColor(0, 0x808080); //
    white
379              neokey.pixels.show();
```

```
380             new_card = card_draw(1) - 1;
381             player_hand[player_cards] = cards[new_card][1];
382             player_total += cards[player_hand[player_cards
    ]][2];
383             for (i=0; i < player_cards; i++) {
384             if(cards[player_hand[i]][2] == 11 &&
    player_total > 21){
385                 player_total -= 10;
386             }
387             }
388
389             //printf("\tDEBUG: NEW_CARD is %d\t%d\n\n",
    new_card,player_hand[player_cards]);
390             player_cards++;
391             print_cards(player_total,player_cards,cards[
    dealer_hand[1]][2],1,play,dealer_hand,player_hand,cards)
    ;
392             delay(500);
393             if(player_total > 21){
394                 red();
395                 printf("Player Busts!\nDEALER WINS\n");
396                 //losedisplay();
397                 //reset();
398                 player_done = true;
399                 dealer_done = true;
400             }
401             else if(player_total == 21){
402                 printf("Player at 21!\n\tDEALER PLAYS OUT\n
    ");
403                 break;
404             }
405         }
406         else if(buttons & (1<<1)){
407           Serial.println("Button B");
408           neokey.pixels.setPixelColor(1, 0x808080); //
    white
409           neokey.pixels.show();
410             printf("\e[1;1H\e[2J");
411             printf("\tDEALER PLAYS OUT\n");
412             player_done = true;
```

```
413         }
414
415     }while (player_done == false);
416
417     print_cards(player_total,player_cards,dealer_total,
    dealer_cards,play,dealer_hand,player_hand,cards);
418     delay(500);
419     while(dealer_done == false){
420       //print_cards(player_total,player_cards,dealer_total,
    dealer_cards,play,dealer_hand,player_hand,cards);
421         //red();
422         //printf("\ndealer logic start\n");
423         //reset();
424
425   if(dealer_total <= 21){
426       if(dealer_total == 21){
427           red();
428           printf("Dealer HIT 21!\n\tDEALER WINS\n");
429                 //losedisplay();
430             dealer_done = true;
431       }
432       else if(dealer_total >= 17 && dealer_total >=
    player_total){
433           red();
434           printf("Dealer's total greater than player's!\n\
    tDealer WINS\n");
435           //losedisplay();
436             dealer_done = true;
437       }
438       else if(dealer_total >= 17 && dealer_total <
    player_total){
439           green();
440           printf("Player's total greater than dealer's!\n\
    tPlayer WINS\n");
441           //windisplay();
442             dealer_done = true;
443       }
444
445
446         else if(dealer_total==17){
```

```
447             for(i=0;i< dealer_cards;i++){
448                 if(dealer_hand[i]==11){
449                     dealer_total -= 10;
450                 }
451             }
452         }
453       else if(dealer_total < 17){
454           new_card = card_draw(1) - 1;
455           dealer_hand[dealer_cards] = cards[new_card][1];
456           dealer_total += cards[dealer_hand[dealer_cards
    ]][2];
457           if(cards[new_card][2] == 11 && dealer_total > 21)
    {
458                   dealer_total -= 10;
459     }
460     dealer_cards++;
461     print_cards(player_total,player_cards,dealer_total,
    dealer_cards,play,dealer_hand,player_hand,cards);
462     delay(500);
463     }
464   }
465         else if(dealer_total > 21){
466       green();
467       printf("Dealer Bust!\n\tPlayer WINS\n");
468       dealer_done = true;
469   }
470         else if(dealer_total == player_total){
471             yellow();
472             printf("Player has pushed with Dealer\n");
473             //reset();
474             dealer_done = true;
475         }
476   }
477     //printf("Player total: %d\n",player);
478     //printf("Dealer total: %d\n",dealer);
479  }while(play = true);
480     return 0;
481 }
482
483 /* print_cards(player_total,player_cards,dealer_total,
```

```
   dealer_cards,play,dealer_hand,player_hand,cards);{
484     int i;
485     short card;
486 
487     printf("\e[1;1H\e[2J");
488     printf("Dealer: ");
489     for(i = 0; i < dealer_cards; i++){
490         card = cards[dealer_hand[i]][1];
491         if(card == 1 || card > 10){
492             high_val_print(card);
493         }
494         else{
495             printf("%d  ",cards[dealer_hand[i]][1]);
496         }
497 
498     }
499     printf("\nDealer Total: %d\n\n",dealer_total);
500     printf("Player: ");
501     for(i = 0; i < player_cards; i++){
502         card = cards[player_hand[i]][1];
503         if(card == 1 || card > 10){
504             high_val_print(card);
505         }
506         else{
507             printf("%d  ",cards[player_hand[i]][1]);
508         }
509 
510     }
511     printf("\nPlayer Total: %d\n",player_total);
512 } */
513 
514 
515 void red() {
516   //printf("\033[1;31m");
517     arcada.display->setTextColor(ARCADA_RED);
518     arcada.display->setTextSize(2);
519     arcada.display->setCursor(100,165);
520     arcada.display->println("You Lose!");
521     neokey.pixels.setPixelColor(0, 0xFF0000); // red
522     neokey.pixels.setPixelColor(1, 0xFF0000); // red
```

```
523     neokey.pixels.setPixelColor(2, 0xFF0000); // red
524     neokey.pixels.setPixelColor(3, 0xFF0000); //
525     neokey.pixels.show();
526     delay(5000);
527     neokey.pixels.setPixelColor(0, 0); // red
528     neokey.pixels.setPixelColor(1, 0); // red
529     neokey.pixels.setPixelColor(2, 0); // red
530     neokey.pixels.setPixelColor(3, 0); //
531     neokey.pixels.show();
532 }
533
534 void green(){
535  // printf("\033[1;32m");
536     arcada.display->setTextColor(ARCADA_GREEN);
537     arcada.display->setTextSize(2);
538     arcada.display->setCursor(100,165);
539     arcada.display->println("You Win!");
540     neokey.pixels.setPixelColor(0, 0x00FF00); // green
541     neokey.pixels.setPixelColor(1, 0x00FF00); // green
542     neokey.pixels.setPixelColor(2, 0x00FF00); // green
543     neokey.pixels.setPixelColor(3, 0x00FF00); //
544     neokey.pixels.show();
545     delay(5000);
546     neokey.pixels.setPixelColor(0, 0); // red
547     neokey.pixels.setPixelColor(1, 0); // red
548     neokey.pixels.setPixelColor(2, 0); // red
549     neokey.pixels.setPixelColor(3, 0); //
550     neokey.pixels.show();
551 }
552
553 void yellow(){
554     //printf("\033[0;33m");
555     arcada.display->setTextColor(ARCADA_YELLOW);
556     arcada.display->setTextSize(5);
557     arcada.display->setCursor(100,165);
558     arcada.display->println("Tie!");
559     neokey.pixels.setPixelColor(0, 0xFFFF00); // yellow
560     neokey.pixels.setPixelColor(1, 0xFFFF00); // yellow
561     neokey.pixels.setPixelColor(2, 0xFFFF00); // yellow
562     neokey.pixels.setPixelColor(3, 0xFFFF00); // yellow
```

```
563     neokey.pixels.show();
564     delay(5000);
565     neokey.pixels.setPixelColor(0, 0); // red
566     neokey.pixels.setPixelColor(1, 0); // red
567     neokey.pixels.setPixelColor(2, 0); // red
568     neokey.pixels.setPixelColor(3, 0);
569     neokey.pixels.show();
570 }
571
572 void reset() {
573   printf("\033[0m");
574 }
575
576 int card_draw(int count){
577     int upper,lower,draw,i;
578     time_t t;
579
580     upper = 52;
581     lower = 1;
582
583     for(i = 0; i < count; i++){
584         draw = random(0,51);
585         //printf(" %d",draw);
586     }
587     //printf("\n");
588
589     return draw;
590 }
591
592 int checkVal(int card){
593     // function to return the value of a card in the High-
    Lo method of card counting
594     // card int must be the number value of the card (
    column 3)
595
596     int val=0;
597     if (card > 1 && card < 7)
598     {
599         val= 1;
600     }
```

```
601     else if (card >= 7 && card < 10)
602     {
603         val= 0;
604     }
605     else if (card == 10 || card == 1)
606     {
607         val= -1;
608     }
609     else
610     //debug
611         printf("Invalid Range (cardval check)\n");
612     return val;
613 }
614 void suit_print(int card_num){
615     if(card_num <= 13){
616         arcada.display->println("H");
617     }
618     else if(card_num <= 26){
619         arcada.display->println("C");
620     }
621     else if(card_num <= 39){
622         arcada.display->println("D");
623     }
624     else{
625         arcada.display->println("S");
626     }
627 }
```

../../Firmware/src/ProjectHardwareCode.ino