# MECE E3430: Senior Design

**Professor Josh Browne**
**May 10, 2019**

# Final Paper: Card Dealer 3000

**Team Dealin' Dirty**

**Andrew DePerro, Connor Finn,**
**John Michael Long, & Brian Nicholas**

**Columbia University**
**Department of Mechanical Engineering**

# Contents

# 1 Executive Summary

The Card Dealer 3000 brings the extraordinary casino experience to the comfort of your own home. It is a compact, table-sized, home entertainment device that automates the card dealing process and allows the user to play multiple games with a variable number of players. The goal of this project is to eliminate the dealer when playing card games such as Texas Hold'em or Blackjack. To begin your hours of card game fun, simply load a pre-shuffled deck of cards into the machine, select your card game of choice, select the number of players, and the Card Dealer 3000 completes all the tedious dealing work for you! The Card Dealer 3000 features a removable card shoe for easy card deck loading, an efficient flipping mechanism, a ranged card shooter, programmability for a wide array of card games, and the ability to accommodate a variable number of players.

This device has the ability to do anything a casino dealer can do once the card deck is loaded into the shoe. It is equipped with 5 individual systems that combine into one amazing product. The first, and most important system of the Card Dealer 3000 is the user interface. A custom made controller is utilized to house a LCD board to display the options of the device as well as a 12-character keypad to choose the desired options. Once the options have been selected, the Card dealer 3000 is ready to do the rest.

The card picking system is the first mechanism the card travels through. After a shuffled deck is placed into the card shoe and the card shoe is inserted into the device, a stepper motor with a foam grip picks the top card from the deck. This moves the card to the flipping system. The flipping system uses a stepper motor with a timing belt to move the housing in card non-flipping and flipping orientation. In the non-flipping orientation, the card moves from the card shoe to an angled slide that deposits the card to the shooting system. In the flipping orientation, the card moves from the card shoe onto a thin angled arm to flip the card along its stable axis. The card then flips onto the same angled slide as the non-flip orientation and is deposited to the shooting system.

The shooting system is a two motor system that the card travels through to be dealt out to a player. A stepper motor with a foam gripper pushes the card forward on the angled slide into a compact DC motor. Once the card comes in contact with the spinning rod attached to the DC motor, it is dealt to the player. The compact DC motor has the ability to deal the card to different distances.

The final system is the rotating system, which includes a Lazy Susan bearing and a DC motor with an attached encoder. The encoder allows the Card Dealer 3000 to rotate accurately. This is important when the device is dealing to different players. One last unique feature about the Card Dealer 3000 is that it uses RFID technology so that the machine knows what cards are being dealt to the players, but more importantly, it knows what it also has. This is key for certain games to be successful such as Blackjack. The Card Dealer 3000's sleek design and easy to use user interface will surely impress any guests at your future game nights. It is a one of a kind product that brings the casino to you!

# 2 Introduction

The entertainment industry, in particular the casino industry, is one of the largest industries in the world, bringing in over $450 billion dollars in earnings from players and gamblers alike [1]. Additionally, an added benefit of the casino industry is its affect on the government. For example, in 2013 in the United States alone, the industry paid $38 billion in local, state and federal taxes through gambling fees, property taxes and income taxes [1]. Therefore, the Card Dealer 3000 and its entertainment capabilities are directly related to the casino industry as it is an automated dealer using RFID technology. This added technology built in to the Card Dealer 3000 is relevant to what is currently being introduced to the casino industry. Recently, RFID (radio-frequency identification) is being installed into casino chips and playing cards to help combat chip and game security [2].

The Card Dealer 3000 is a compact, automated, card dealing machine with 5 games programmed into it with the capabilities of adding more in the future. Additionally, up to 6 players can play at a time, and because it is equipped with RFID technology, it can read all cards being played in the selected game. Therefore, it can deal and play games where the dealer has input such as blackjack and casino war. Another

great feature of the Card Dealer 3000 is its removable card shoe. The point of having the card shoe be removable is that it can easily be removed to add a hand-shuffled deck of cards. Once the shuffled cards are loaded into the card shoe, the shoe slides into the carriage and locks into place by the attraction force of magnets. One last key feature of the Card Dealer 3000 is the user interface custom controller used to control overall game-play. This controller has a surface mounted LCD screen and 12 button keypad. The LCD prompts you to select the number of players and which game to play. Once this is done and the selected game has started, depending on the game, it prompts you to add user judgment on the game. For example, in blackjack, once your hand has been dealt, it will as you if you want to hit, stay, or double down. Also, within blackjack, it will notify you of your current score, and if you bust or not. In terms of mechanical systems, the Card Dealer 3000 can be broken down into a few smaller subsystems. These systems are task-based and titled card picking, card flipping, card shooting and player base rotation. The first system, card picking, is a Nema-17 stepper motor with a D-shaped gripper/roller rotates to pick the first card off of the deck. If the card does not need to be flipped, it will simply slide down a ramp and enter the card shooting mechanism, which is another Nema-17 stepper motor with Nerf bullet tips to feed the card into a rapidly spinning DC motor to shoot out the card. If the card needs to be flipped, it enters the card flipping mechanism, in which the card shoe and carriage will move linearly via a third Nema-17 stepper motor equipped with 2 sprockets and a timing belt. This moves the assembly a few inches before the card is picked. Once picked, the card then falls onto a rail which causes it to flip over its primary axis of rotation before it slides down the ramp to where the shooter mechanism awaits it. If the machine is dealing cards to itself (the dealer), the shooter mechanism slows to a lower number of rotations per minute so that the card drops right in front of the device. The last mechanism is the player base rotation, which is a directly-driven DC motor with an attached encoder. The Card Dealer 3000 is all mounted on an 18" lazy susan bearing which allows it to rotate with minimal friction so the dealer can be delivered to any position on the table at ease.

# 3 Background Information

A literature search was conducted on various card handling and dealing mechanisms to draw inspiration for this project. There are many potential solutions that allow for cards to be dealt easily and consistently. One existing solution is a product called the Wheel-R-Dealer, which is a simple mechanism that accelerates the deal of a card via the push of a finger. The product is simple, cheap, and offers a successful manner of executing the deal. This device and its high speed motor served as a source of inspiration for this project's card shooting mechanism [3].

A few existing card dealers were looked at including a past Columbia University senior design project. This dealer was moderately successful, but it limited the game to four players and the cards were not shot out to the players as a real dealer would. Another result of the internet search was another senior design project by students from UC Berkeley. Their design offers one elegant solution to the problem of flipping a card without restricting the card shooting capabilities of the dealer. The project employs a flipping ledge similar to the one used by the past Columbia University students. However, the flipping mechanism occurs before the deal, allowing for more variety of game play. There are some limitations to this design, such that it is very large relative to a standard game table [4]. The Card Dealer 3000 aims to flip the card in advance of the deal, as the UC Berkeley project does, but this design is more compact and has a more visually appealing design. Common shortcomings for existing dealers included bulky designs, dealing methods that did not follow casino protocols, limitations to the number of players, and the inability to shoot out cards. The goal of this project was to mimic the actions of a human casino dealer as closely as possible. Numerous solutions were considered for drawing a card from a deck, flipping the card, and ultimately shooting out a card.

A thorough patent search was also conducted to investigate existing relevant inventions around the world. This search resulted in patent discoveries concerning card holders, card presses, card shooting devices, card game logic, and card dealing machines. The most relevant results from this patent search are summarized in Figure 1 below:

| Patent Number | Patent Description |
|---|---|
| CN201940009 (U) | Card shooter device with a card storage compartment |
| US2018161665 (A1) | A card storage device that removes a cards individually from the bottom of the deck |
| WO2015134619 (A1) | Card holder with slot to receive/eject a card |
| US2017109971 (A1) | Program for scoring and determining winner of electronic blackjack |
| WO2016100770 | A card dealer with an oscillating arm |

Figure 1: Chart of All Patents Referenced in the Design Process

# 4 Analysis

Engineering analysis was a key component throughout our design process of the Card Dealer 3000. In particular, analysis of the spring constant within the card shoe, card flipping analysis about its stable axis of rotation, and torque analysis to properly spec out our desired motors can all be seen in more depth in the below subsections.

## 4.1 Card Shoe Analysis

The size of a playing card is the driving dimension in the entire system. The card shoe was designed around the size of a card in order to keep the shoe as compact as possible. The assembly was designed to flip the card over the stable axis of rotation, and the easiest way to do so was to eject the cards from the shoe along their longest length. Initial card shoe prototypes were laser cut from balsa wood to test the ease at which cards can be drawn from the deck. Typically, casino card shoes are angled at 45 degrees, however, the card tilt angle of 30 degrees from the horizontal was chosen due to the greater ease of removing the card. Later iterations of the card shoe were constructed from acrylic since acrylic is light weight and has a low coefficient of friction. Some key dimensions and characteristics of the final card shoe can be seen in Table 1 below:

| Measurement | Value |
|---|---|
| Static coefficient of Friction for Acrylic | 0.20 |
| Length | 5.993 (in) |
| Width (outside dimension) | 2.98 (in) |
| Width (inside dimension) | 2.52 (in) |
| Max Height | 3.418 (in) |
| Card Angle | 30 (degrees) |
| Card Shoe Tilt Angle | 6 (degrees) |

Table 1: Chart of Card Shoe Dimensions

Two primary considerations were made in selecting the optimal spring to keep the cards pressed into the card shoe. First, the spring needs an uncompressed length long enough to maintain a pressure on the cards up until the last card is drawn. The length dimensions of our card shoe, and triangle block support require a spring of at least 2.2 inches. Secondly, the spring coefficient must result in a force that resists the weight of the cards and prevents slipping, but does not hinder the roller's ability to draw the top card. A static analysis of the deck of cards resting against the spring supported block was completed to determine the correct dimensions of the spring. A free body diagram of the card shoe block is shown in Figure 2, where $F_s$ is the spring force, $W_b$ is the weight of the block, $W_c$ is the weight of the cards, $N$ is the normal force, $f$ is the friction force, and $\theta$ is the angle of the block face relative to the bottom of the card shoe. A symbol key for the free body diagrams is also included in Figure 4. Table 2 provides the specifications for a deck of

cards used in the analysis. A summation of the forces in the x and y directions results in the following:

$$\sum F_y = N - W_c cos(\theta) cos(\theta) = 0 \tag{1}$$

$$\sum F_x = F_s + f - W_c cos(\theta) sin(\theta) = 0 \tag{2}$$

From equation 1, one can solve for the normal force, $N$, using $\theta = 30$ degrees, and $W_c = 0.22$ lbs. The value for the normal force is used in equation 2 for the friction force, $f = \mu * N$, where $\mu = 0.2$ is the coefficient of static friction. To find the minimum allowable spring force, the forces in the x-direction is set equal to zero.

To find the minimum allowable spring force, it is necessary to consider the change in spring compression and deck weight as cards are dealt. A python file, provided in Appendix A, was created to continuously solve for the spring force after the removal of each card. The spring force required to prevent slipping was 0.031 lbs, which corresponds to a spring constant of .012 lbs/in. Based on this analysis, a spring with the required length of 2.5 inches and lowest available spring coefficient of 0.92 lbs/in was selected and purchased on McMaster.com.

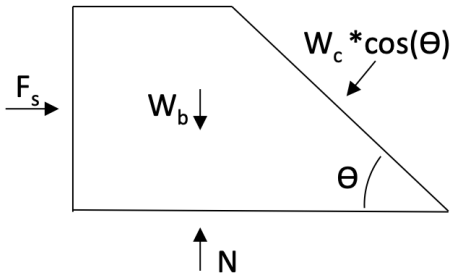



Figure 2: Free Body Diagram of the Card Shoe Block

| Deck of Cards Dimensions | Value |
|---|---|
| Weight (pound force) | 0.22 |
| Width (inch) | 2.5 |
| Height (inch) | 3.5 |

Table 2: Deck of Cards Specifications

The correct motor must be able to pull a single card off the top of the deck. To accomplish this, the torque of the motor must be greater than the friction force between the top card and the front face of the card shoe. The analysis above showed that the largest spring force occurred when an entire deck was loaded. This resulted in a fully compressed spring.

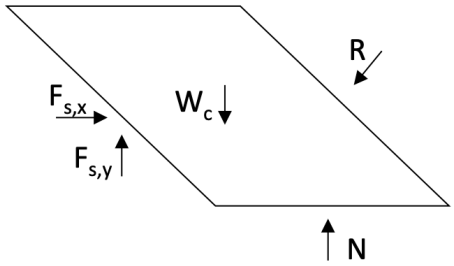



Figure 3: Free Body Diagram of the Deck of Cards

N = Normal Force
$F_s$ = Spring Force
$W_b$ = Weight of Block
$W_c$ = Weight of Card Deck
R = Card Shoe Front Reaction Force

Figure 4: Free Body Diagram Key

Static equilibrium in the x-direction gives:

$$\sum F_x = F_{s,x} + R_x = 0 \tag{3}$$

where $F_{s,x}$ is the spring force in the x-direction, and $R_x$ is the reaction force from the card shoe. Solving for the reaction force gives:

$$R_x = k\Delta x cos(30) = 0.92 \cdot 2.5 cos(30) = 2.0 \tag{4}$$

where $k$ is the spring force constant, and $\Delta x$ is the displacement from spring's uncompressed position. The resultant force is in lbs force. Similarly, in the y-direction:

$$\sum F_y = F_{s,y} + R_y - W_c + N = 0 \tag{5}$$

where $W$ is the weight equal to 0.22 lbs force, and $F_{s,y}$ and $R_y$ are the spring and reaction forces, respectively. Again, solving for the reaction force in the y-direction:

$$R_y = -k\Delta x sin(60) + W_c - N = 0.92 \cdot 2.5 sin(30) + 0.22 - N \tag{6}$$

The upward spring force exceeds the weight of the cards, so the normal force N in Figure 3, is considered negligible and $R_y = 1.37$ lbs. The torque applied by the card shoe on the motor will be the friction force, $f = N\mu$, with $\mu$ representing the coefficient of friction. Acrylic has a coefficient of static friction of 0.2 and a coefficient of dynamic friction of 0.15, so in this analysis only the static friction was considered.

The normal force initiated on the top surface of the card, $R$ is calculated from the static force analysis as follows:

$$R = \sqrt{R_x^2 + R_y^2} = \sqrt{1.37^2 + 2.0^2} = 2.42 lbs \tag{7}$$

The friction force incident on the top card is $f = R\mu = 2.42 \cdot 0.2 = 0.484$. The applied torque on the motor from the cards is equal to the friction force times the radius of the roller, $\tau = 0.484 \cdot 0.625 = 0.303$ lbs*in. This required torque of the motor is very small, and thus, just about any functional motor we select will be able to pull a card. For our design, we selected a Nema 17, 12V, 1.7 amp, bipolar stepper motor.

## 4.2 Card Flipping Analysis

A card will be flipped by landing on an acrylic laser-cut "arm" which will initiate a rotation about the card's central axis as shown in (b) of Figure 5. The acrylic ledge will be mounted on the base plate and will be slightly longer than the length of a playing card ( 2.5 inches). When the game play requires that a card is flipped, the card shoe assembly will be moved using a stepper motor and timing belt to cause the trajectory of the card to fall onto this ledge which will cause the flip.

Of the three axis shown in Figure 5, (a) and (b) are stable, while (c) is unstable. A rectangular prism will maintain it's trajectory while rotating about a stable axis, but will move unpredictably when rotating about an unstable axis. A playing card can be modeled as a rectangular prism, of length 3.5 inches, width 2.5 inches and thickness 0.012 inches. Therefore, flipping the card about its 3.5 inch axis should result in consistently stable rotation.

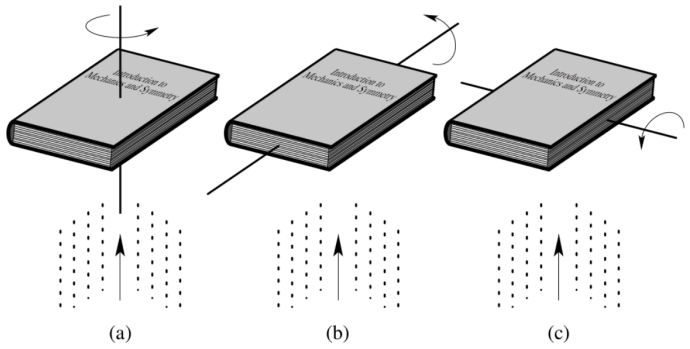



Figure 5: Axis of Rotation for a Rectangular Prism. [5]

This understanding of rotational stability led to a change in design for the movement of the card shoe relative to the flipping "arm" from rotational to linear movement. At the time of design review I, the original design was for the card shoe to rotate such that the cards fell on the ledge at an angle. However, if the card shoe was in fact rotated five degrees, the cards would also flip along an axis offset by that angle. By instead moving perpendicularly to the ledge, the cards will land with its long central axis parallel to the ledge which should result in stable rotation for every flip.

## 4.3 Card Shooting Analysis

One of the most important aspects of this machine is to have the card physically shoot out to the player, emulating a human dealer at casino table. To shoot a card to each player, each card will come into contact with a DC motor rotating at a high rpm to project that card off from an acrylic surface. Acrylic has a coefficient of static friction of only 0.2 $\mu$, as provided in Table 1. The torque necessary for delivering the card then is as follows:

$$T = f * r = N * r = W_{singlecard} * \mu * r \tag{8}$$

where $r$ is the radius of the roller, $N$ is the normal force, $W_{singleCard}$ is the weight of a single card, $\mu$ is the coefficient of static friction for acrylic, $f$ is the friction force, and $T$ is the torque required to shoot out the card. This analysis results in a friction force of $f = N * \mu = 0.004 \cdot 0.2 = 0.008$ lbs, and with the roller of radius, 0.625 in, the necessary torque is only 0.005 lbs*in.

## 4.4 Electrical Analyses

The Card Dealer 3000 is powered by 5 motors, provided in Table 3, which work to efficiently deal cards at variable speeds and orientations to a number of players. The motors were selected due to the torque

needed to operate each task. A preliminary analysis was done with a python script, shown in Appendix A, in order to correctly size the motor for picking a card. This analysis proved that small motors were sufficient for picking a card off the top of the deck; however, a more powerful Stepper motor was selected because this would allow the Card Dealer 300 to project cards onto the slide at faster speeds. The Nema 17 stepper motor, provided in Table 3 is also used to drive the timing belt and rotate the shooting mechanism's card pusher which feeds the card into the NMB Technologies Brush Motor. Purchasing motors in bulk allows for a cost reduction, and the large factor of safety results in very consistent production. To operate each of these steppers, a TB6600 motor driver is used because the rated current and voltage of the stepper motors fall within the specifications of the TB6600 motor driver. Two DC motors are used in the Card Dealer 3000. The first is the NMB Technologies Brush Motor (High RPM) used to project the card to each player. The second is the Polulu 131:1 Metal Gearmotor used to rotate the base. The first motor was selected because there are low torque requirements, but it was necessary to rotate the motor at very high speeds. The second motor had entirely opposite requirements; hence a low rpm and high torque motor was selected. The L298N motor driver, provided in Table 4 is selected to drive both stepper motors because the current requirement is lower.

Each motor is driven at 12V; thus, AC to DC converters rated for 12V DC were selected. This allows the Card Dealer 3000 to be conveniently plugged into an the wall using an extension cord. The Arduino Mega 2560 also hosts 5V and 3.3V output pins, which are useful for driving other electrical systems, such as the RFID module, LCD board, and end stops. The voltage and current requirements for these systems are provided in Table 5. A complete wiring diagram is provided in Appendix C.

| Motor | Quantity | Voltage (V) | Current (A) | Holding Torque (oz $\cdot in$) | Rated Torque (oz $\cdot in$) |
|---|---|---|---|---|---|
| Nema 17 BiPolar Stepper Motor | 3 | 12 | 1.7 | 61 | N/A |
| DC Motor, Polulu 131:1 Metal Gearmotor | 1 | 12 | 0.3 | N/A | 250 |
| DC Motor, NMB Technologies Brush Motor (High RPM) | 1 | 12 | 0.1 | N/A | 2.4 |

Table 3: Motor Selection

| Motor Driver | Quantity | Voltage (V) | Current (A) |
|---|---|---|---|
| L298N | 1 | 5 - 35 | 0 - 2 |
| TB6600 | 3 | 9 - 42 | 0.5 - 4 |

Table 4: Motor Driver Selection

| Electrical Part | Quantity | Voltage (V) |
|---|---|---|
| MFRC-522 RFID Reader | 1 | 3.3 |
| 20 x 4 Character LCD Display Screen | 1 | 5 |
| Mechanical Limit Switch / Endstop | 2 | 5 |

Table 5: Additional Parts Voltage Requirements

# 5 Design Overview

The main mechanical sub-systems in the Card Dealer 3000 are the card picking system, the card flipping system, the card shooting system, the base rotation system, and the user interface. The electrical systems, the RFID system, and the programming of the dealer also play vital roles in the function of the dealer.

## 5.1 Card Picking

The first major decision in designing the Card Dealer 3000 was determining how the cards would be stored. Many existing automatic card dealers stack cards vertically, like in a card box. However, in an effort

to mimic casino dealers, a card shoe was designed with the capability of storing multiple decks at an angle in order to allow cards to be easily removed from the deck. A card shoe quickly proved to be an easier way to remove cards from the top of the deck. A wide array of printer grips were tested in removing cards from the deck, and after adjusting both the grip type and the angle of the card shoe, the grip-card shoe combination in the final design was selected. A stepper motor was mounted with the selected gripper and the final card picking design is shown in Figure 6.

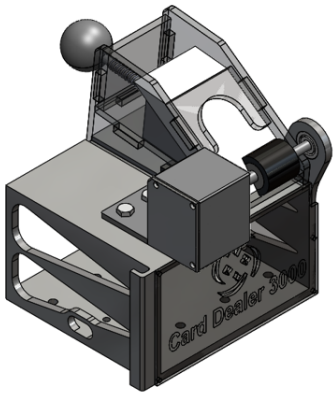



Figure 6: CAD Image for the Card Picking System

## 5.2 Card Flipping

Numerous options were also considered for how to flip a card. These ideas included flipping the card after it has been dealt, flipping the entire deck before it deals a card, and dealing a card onto an extrusion that would cause the card to flip. Flipping a card after it had been dealt is an idea that was set aside because it limited how far away from the dealer that the card could be dealt. The idea of flipping the deck was also set aside due to compromises of integrity that could be caused by exposing cards other than the top card. Dealing cards off the bottom of the deck was tabled since it could also compromise the integrity of the game. Finally, it was decided that cards would be flipped using an extrusion before they were shot out to the players. The original plan was to have the card deck rotate towards and away from the extrusion, but eventually this rotation was replaced by a linear motor to result in only one axis of rotation as further explained in Section 4.2.

The flipping system uses a stepper motor with a timing belt to move the card shoe between card flipping and non-flipping orientation. In the non-flipping orientation, the card moves from the card shoe directly to an angled slide that deposits the card to the shooting system. In the flipping orientation, the card moves from the card shoe onto a thin angled arm to flip the card along a stable axis of rotation. The flipped card then drops onto the angled slide and is delivered to the shooter assembly. This system is shown in Figure 7.

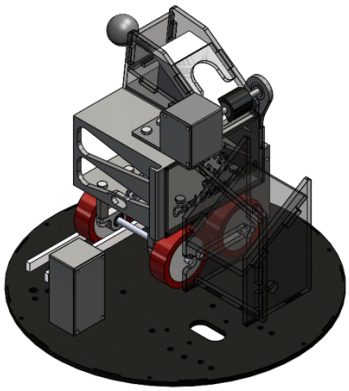


Figure 7: CAD Image for the Card Flipping System

## 5.3 Card Shooting

Many options were also considered for how to deliver cards to players. After examining multiple existing card dealers, it was clear that the ideal dealer would be able to deal cards to any number of players. It was also desired that the cards could be shot at a range of distances to simulate how a casino dealer slides cards out to players at the table and to him or herself. After the card is dealt, a pair of ramps allow the card to slide down to a rapidly spinning DC motor with a grip capable of shooting out the card. The entire assembly has the ability to rotate 180 degrees to simulate the range of motion that a casino dealer covers when dealing out cards.

The shooting system is a two motor system allows cards to be dealt across the game table. A stepper motor mounted with six foam Nerf darts push the cards forward on the angled slide into the shooting platform armed with a NMB Technologies Brush Motor DC 18mm X 25mm which has a very high RPM. Once the card comes into contact with the rubber grippers mounted on the shooter rod, it is delivered to the player. The compact DC motor pictured above is capable of dealing cards across the table or to the dealer itself. This system is shown in Figure 8.

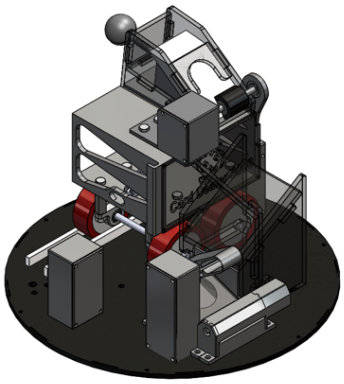


Figure 8: CAD Image for the Card Shooting System

## 5.4 Base Rotation

In order to deal cards to a variable number of players, the Card Dealer 3000 can rotate 180 degrees about the central axis. The upper plate of the dealer, which holds the assembly, is mounted onto a Lazy Susan

bearing. The Lazy Susan bearing reduced friction allowed for a less expensive motor with a lower torque to be used. It also ensured a consistent rotation throughout the duration of game play. The base is rotated using a direct drive from a 131:1 Metal 37D x 73L mm Gearmotor. This motor is equipped with a shaft mounted 64 CPR encoder which enabled the accurate rotation necessary for game play. Prior to each game, the base was homed to its zero point, located on the far right hand side of the table. An end-stop limit switch was mounted on the base plate, such that an extruded part from the rotating plate would activate the limit switch once the assembly had rotated to the home location. A shaft mounted encoder is an incremental system, meaning it has no absolute awareness of its location. By homing the device each game, the Card Dealer 3000 ensured that any deviation from its initial location was reset before the game started. In this way, error due to player interaction with the device would not compound as more and more games are played. The base rotation system is shown in Figure 9.

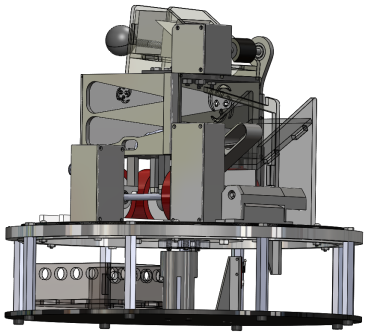


Figure 9: CAD Image for the Base Rotation System

## 5.5 User Interface

The Card Dealer 3000's user interface features a custom designed controller, shown in Figure 10, which houses a 4 row by 20 character LCD board and a 9-character keypad. The interface first prompts the players to select the number of participants and their desired game. Once the inputs have been selected, the Card dealer 3000 is ready to do the rest. At the conclusion of each game, the player is asked whether they would like to restart the game or return home. In this way, the game play is efficient and user friendly.

The interface also plays an important role in the game play of Blackjack and Casino War. In these games the user competes against the Card Dealer 3000, so it is necessary for information to be displayed. During Casino War, if any player ties the dealer, they are prompted to either surrender of engage in a war. If a war is selected, then the game play would continue. In Blackjack, the LCD board displays each user's current score and prompts the standard BlackJack options (hit, stay, double, or split). At the conclusion of these games, the interface will display the winning players.


Figure 10: CAD Image for Customed Designed User Interface Controller.

## 5.6 Electrical Systems

The electrical system incorporated in the Card Dealer 3000 is shown in Figure 11.

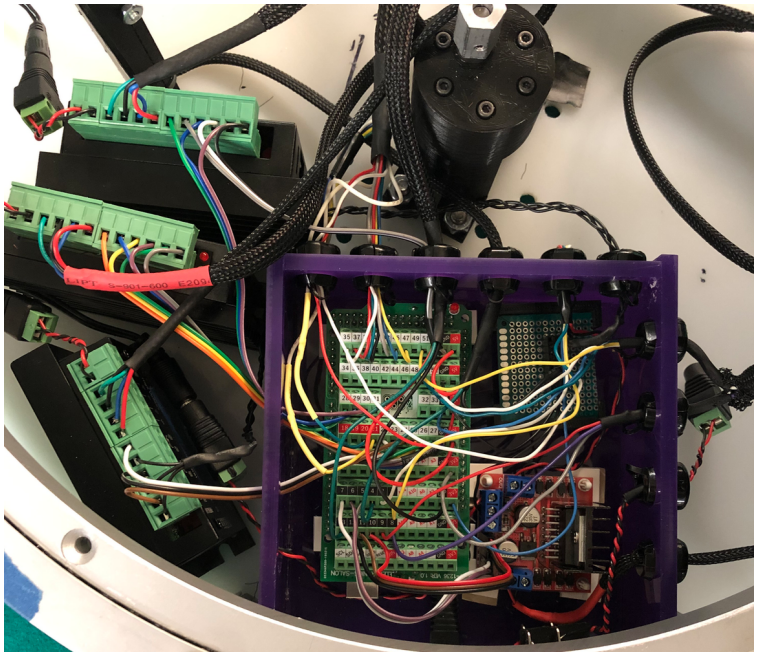


Figure 11: Wiring housing inside Card Dealer 3000

The purple acrylic electrical box pictured above houses the Arduino Mega as well as a circuit board acting as a common ground and a L289N motor driver used to power both DC motors. On the left side of the electrical box, three TB6600 motor drivers stand vertically, and each one powers a separate Nema 17 stepper motor. Looking at the perimeter walls of the electrical box, every wire passes through a strain relief clip, which assures that no wires can be pulled from the brains of the machine. This is important because there are many moving parts within the Card Dealer 3000, so these strain reliefs prevent wires from being pulled out of the common ground or Arduino as the base rotates.

## 5.7 Radio-Frequency Identification (RFID)

The Card Dealer 3000 hosts an RFID module mounted beneath the shooting ramp, and each card is tagged with a slim RFID chip. Each chip was pre-programmed by Dealin' Dirty to the exact value and suit of the card. The RFID technology plays two important roles for the operation of the Card Dealer 3000. The first is that the RFID reader is a point of feedback for the dealer itself. Within the dealer's programming, it is written that a the shooting mechanism will not be activated unless a card is read by the RFID module. Inspiration for this feature comes from the common jamming of paper in standard paper printers. By requiring that the Card Dealer 3000 waits for the playing card to be situated correctly on top of the shooter, the dealer ensures that the card will not get jammed within the device. Once the card is read, the card is set and the game play continues as usual. The second function of the RFID module is to provide in formation on card values and suits to the dealer. As the game continues, the Card Dealer 3000 keeps track of each player's cards which allows for interaction between the players and the dealer in games such as Blackjack and Casino War.

## 5.8 Programming

The Card Dealer 3000 operates based on the logic written into an Arduino Mega Board, shown in Figure 12. Arduino IDE is an environment within the C++ language which allows for interaction with Arduino boards. The Dealin' Dirty Card Dealer 3000 program controls each of the sub assemblies, detailed in Section 5, in order to efficiently deal out each card game. For efficiency, each action used to move a card, picking the top card, sliding the assembly to a flipping location, rotating the base a specific angle. and shooting the card to the player are written as functions. These functions are then called into each game's program in accordance with the game play logic. Arrays are used to store important gameplay information, such as the value and suit of the cards held by each player. While loops, for loops and if statements are the building blocks for the logic throughout the program. The dealer's program is provided in Appendix B.

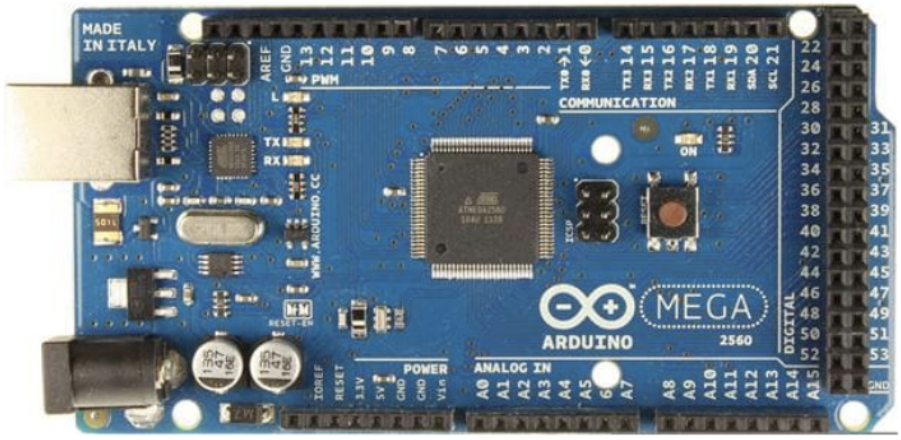



Figure 12: Arduino Mega used for the Card Dealer 3000. [6]

## 5.9 Final Design

All of the sub-assemblies described above in Section 5 allow for the Card Dealer 3000 to function as an automated card game dealer. An isometric view of the complete assembly is shown in Figure 13, and a detailed Drawing is displayed in Figure 14. The bill of materials for the CAD drawing is shown in Figure 15.

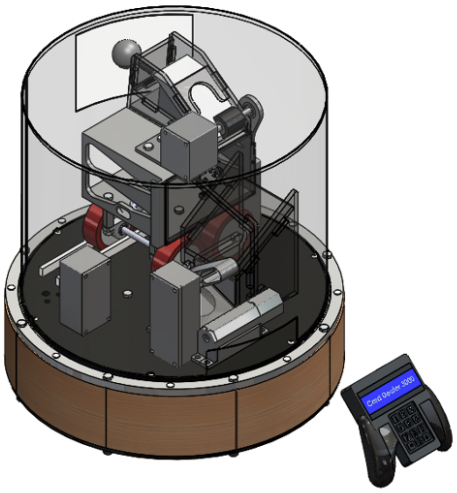


Figure 13: Full CAD Design Image for the Card Dealer 3000

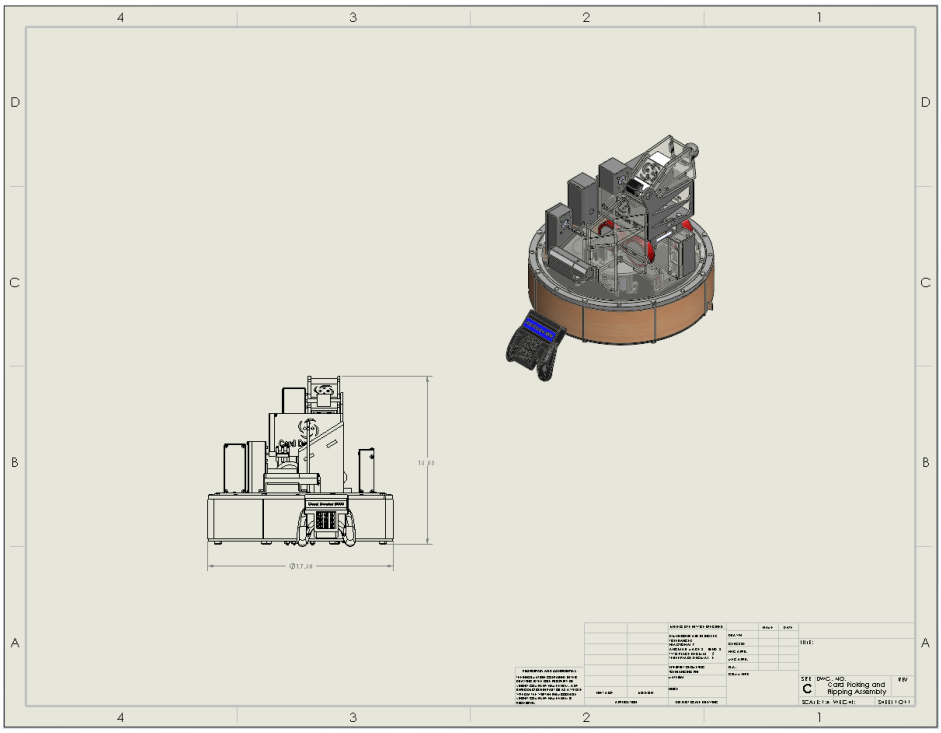


Figure 14: Detailed Engineering Drawing

| ITEM NO. | PART NAME | QTY. | ITEM NO. | PART NAME | QTY. |
|---|---|---|---|---|---|
| 1 | Triangle 60 V5 | 1 | 43 | Base Wrap Panels 2 (rear) | 1 |
| 2 | Back Plate 60 V5 | 1 | 44 | Base Wrap Panels 2 (main) | 6 |
| 3 | Leg 2 60 V5 | 1 | 45 | Limit Switch Mount | 1 |
| 4 | Leg 1 60 V5 | 1 | 46 | Limit Switch Trigger Mount | 1 |
| 5 | Bottom Plate 60 V5 | 1 | 47 | Base Wrap Panels 2 (power) | 1 |
| 6 | Cards 60 V5 | 1 | 48 | PCB^Endstop_Microswitch_1A125VAC_DesjgnByMakerbot | 2 |
| 7 | Top Plate 60 V5 | 1 | 49 | Microswitch_1A_125vAC (seul) | 2 |
| 8 | 92395A116 | 1 | 50 | Connecteur^Endstop_Microswitch_1A125VAC_DesjgnByMakerbot | 2 |
| 9 | 9657K413 | 1 | 51 | Large Wheel Spacers | 8 |
| 10 | 92198A558 | 1 | 52 | Red 3 inch Wheels | 4 |
| 11 | Magnet Mount 60 V5 | 1 | 53 | 91309A551 | 8 |
| 12 | Sphere Knob Fixed | 1 | 54 | C Shape Card Flipper - Left | 1 |
| 13 | Aluminum Tube | 1 | 55 | Big Triangle Ramp Support - Left | 1 |
| 14 | Axle | 2 | 56 | Card Ramp - Left | 1 |
| 15 | 0.25 - 20 hexnut.step | 40 | 57 | Sprocket Suport | 1 |
| 16 | mounted bearing.step | 4 | 58 | Sprocket Motor Mount (small sprocket) | 1 |
| 17 | Small Triangle Ramp Support - Left | 1 | 59 | Wheel Rail | 2 |
| 18 | 6383K15 | 1 | 60 | Sprocket Motor Mount Cover (small sprocket) | 1 |
| 19 | 93306A540 | 33 | 61 | Sprocket Support Cover | 1 |
| 20 | Picker Motor | 2 | 62 | Sprocket | 2 |
| 21 | roller cam V2 | 1 | 63 | Shooter Ramp (0.125 acrylic top) | 1 |
| 22 | Picker Motor Bracket | 1 | 64 | 0.125 acrylic top | 2 |
| 23 | 91841A011 | 2 | 65 | RFID Card Holder | 1 |
| 24 | 93075A244 | 2 | 66 | Shooter Motor 2 | 1 |
| 25 | Magnet Mount | 1 | 67 | Shooter Motor 2 Connector RUBBER | 1 |
| 26 | 5848K73 | 1 | 68 | Shooter Motor 2 Mount | 1 |
| 27 | 18635A540_LARGE-DIAMETER RING-STYLE TURNTABLE | 1 | 69 | Shooter Picker Motor | 1 |
| 28 | Circle Base Plate | 1 | 70 | Shooter Picker Motor Bracket | 1 |
| 29 | 91772A199 | 29 | 71 | Shooter Picker (Nerf) | 1 |
| 30 | 91841A009 | 13 | 72 | Shooter RUBBER Roller | 2 |
| 31 | Aluminum Legs | 8 | 73 | Timing Belt Clip (1) | 2 |
| 32 | Base | 1 | 74 | Timing Belt Clip Tab | 2 |
| 33 | 9541K1 | 8 | 75 | Shooter Picker Motor Bracket Cover | 1 |
| 34 | Picker Motor Bracket Cover | 1 | 76 | Shooter Motor 2 Mount Cover | 1 |
| 35 | Aluminum Tube Cover | 1 | 77 | Shooter Motor 2 Mount Top | 1 |
| 36 | Aluminum Tube Rear Cover | 1 | 78 | Dealing Dirty Controller | 1 |
| 37 | Under Plate | 1 | 79 | 91290A113 | 12 |
| 38 | base rotation gear | 1 | 80 | 91251A203 | 1 |
| 39 | hex motor shaft attachment | 1 | 81 | 91772A128 | 16 |
| 40 | Motor 2 | 1 | 82 | 91841A006 | 2 |
| 41 | Base Motor 2 Mount | 1 | 83 | 91772A074 | 7 |
| 42 | Electrical Box | 1 | | | |

Figure 15: Bill of Materials

The base of the final design is approximately seventeen and a half inches in diameter and should easily fit on most card tables. Standard hardware sizes of 1/4-20, 10-24, 8-32, 4-40, and M3 were used for the entire assembly.

## 5.10 Experiments and Test Results

Testing was performed throughout the design progression of the Card Dealer 3000 in order to track and improve the consistency of the dealer. A final testing was completed after the design had been finalized. The final testing results for the dealer are summarized in Table 6. Ten full 52 playing card decks were used to test the success of the dealer in both the flip and non-flip configurations. Provided that the deck of card was firmly secured in the lead edge of the card shoe, there were minimal failures in the picking of the card and the delivery of the card to the shooter for both the flip and non-flip configurations. Two failures for the RFID reader were recorded, and no failures for the shooting of the cards were observed.

| System | Number of Trials | Number of Failures | Percentage of Successful Trials (%) |
|---|---|---|---|
| Card Picking (No Flip) | 520 | 14 | 97.3 |
| Card Picking (Flip) | 520 | 12 | 97.7 |
| Card Delivery to Shooter (No Flip) | 506 | 4 | 99.2 |
| Card Delivery to Shooter (Flip) | 508 | 21 | 95.9 |
| RFID Reading | 989 | 2 | 99.8 |
| Card Shooting | 987 | 0 | 100 |
| Total | 1040 | 53 | 95.0 |

Table 6: Dealer Testing Result Summary

## 5.11 Testing Analysis

A failure for the picking mechanism was recorded if the picker did not move the card onto the slide. It is important to note that if the first card was not picked, one failure was recorded (not 52) and the cards were reset into the card shoe. Additionally, the majority of the card picking failures occurred for the first card or for card towards the bottom of the deck; thus game play is unlikely to be impacted by a failed pick. For this reason, A failure was recorded for the card delivery to the shooter if the card got stuck on the slide. This happened more frequently during the flipping configuration. During game play, it is possible that this will occur more than 1 percent of the time, if the timing belt does not slide precisely to the perfect position. The RFID reader consistently recorded the correct card value. However, a failure was recorded if the reader did not read the card, and a restarted game was necessary. Finally, a failure for the shooter would have been recorded if the card was not projected from the mechanism. If failed cards are reset after a failure, the Card Dealer 3000 will successfully deal a card 95 percent of the time.

The Card Dealer 3000 hosts plenty of feedback within its design. One feature is that if a card is not read by the RFID technology, the program pauses until a card is read. This prevents a card from jamming, and eliminates the need for a restarted program in the event of a failed deal. For this reason, only the 0.2 percent failure which occurs from the reading of the RFID tag would result in the restart of a program. Thus, the Card Dealer 3000 there is a 99.8 percent that each card dealt allows for a game to successfully continue.

## 5.12 Cost Estimate

The cost estimate shown in Figure 16 breaks down all the materials needed to construct a card dealer. The total cost includes replacement parts, and the premium add-ons include parts that ease the assembly and repair of the dealer and improve the aesthetics of the dealer. In addition to the parts listed below, access to a drill press, a laser cutter, a 3D printer, soldering equipment is required as well as miscellaneous hardware in the following sizes: 1/4-20, 10-24, 8-32, 4-40, and M3. Instructions for manufacturing and assembly are provided at carddealer3000.weebly.com along with a link to the Card Dealer 3000 program code on Github and all STL files needed for 3D printed parts.

| Part | Supplier | Unit Cost | Quantity | Total | Use |
|---|---|---|---|---|---|
| Cast Black Acrylic, 24" x 36" x 1/4" | McMaster-Carr | $ 62.65 | 1 | $ 62.65 | Base Plates |
| 6061 Aluminum Hex Bar, 1/2" Wide x 3' L | McMaster-Carr | $ 17.04 | 1 | $ 17.04 | Dealer Legs |
| Round Bumpers with Threaded Stud | McMaster-Carr | $ 6.64 | 1 | $ 6.64 | Rubber Feet |
| Round Turntable, 17.4" OD | McMaster-Carr | $ 71.27 | 1 | $ 71.27 | Lazy Susan Bearing |
| Cast Black Acrylic, 12" x 24" x 1/4" | McMaster-Carr | $ 19.93 | 1 | $ 19.93 | Slide, Card Shoe, Base Track, Electrical Box |
| Cast Black Acrylic, 12" x 12" x 1/8" | McMaster-Carr | $ 7.14 | 1 | $ 7.14 | Shooter Ramp Cover, Main Box Covers |
| Neodymium Magnet | McMaster-Carr | $ 10.07 | 2 | $ 20.14 | Secure Card Shoe |
| Lightweight Polyurethane Wheel, 3" Diameter x 15/16" Wide | McMaster-Carr | $ 4.08 | 4 | $ 16.32 | Main Box Wheels |
| Mounted Bearing, 5/16" ID | McMaster-Carr | $ 4.42 | 4 | $ 17.68 | Main Box Wheel Bearings |
| Threaded Rod, 5/16"-18 x 6" | McMaster-Carr | $ 2.28 | 2 | $ 4.56 | Main Box Wheel Axles |
| Ball Bearing, Open, 3/16" Shaft Diameter | McMaster-Carr | $ 6.90 | 1 | $ 6.90 | Shooter Bearing |
| Ball Bearing, 5/16" Shaft Diameter and 7/8" Housing ID | McMaster-Carr | $ 5.70 | 1 | $ 5.70 | Card Picker Bearing |
| Printer Pick Up Rollers | Amazon | $ 10.00 | 1 | $ 10.00 | Gripper for Picking Cards |
| Nerf Dart Refill Pack | Amazon | $ 5.18 | 1 | $ 5.18 | Card Shooter Grippers |
| 302 Stainless Steel Corrosion-Resistant Compression Springs | McMaster-Carr | $ 6.59 | 1 | $ 6.59 | Card Shoe Spring |
| 5m GT2 Timing Belt, Timing Pulley Wheel, and Belt Locking Spring | Amazon | $ 11.99 | 1 | $ 11.99 | Main Box Timing Belt |
| PLA Spool, 1kg | Amazon | $ 19.99 | 3 | $ 59.97 | 3D Printed Parts |
| Nema 17 Stepper Motor | Amazon | $ 12.99 | 1 | $ 12.99 | Stepper Motors |
| NMB Technologies Brush Motor DC 18mm X 25mm | Digikey | $ 9.15 | 1 | $ 9.15 | DC Shooter Motor |
| 131:1 Metal Gearmotor 37Dx73L mm with 64 CPR Encoder | Pololu | $ 39.95 | 1 | $ 39.95 | DC Base Motor |
| MYSWEETY TB6600 4A 9-42V Stepper Motor Driver | Amazon | $ 14.39 | 3 | $ 43.17 | Stepper Motor Drivers |
| Qunqi L298N Motor Drive Controller | Amazon | $ 6.99 | 1 | $ 6.99 | DC Motor Driver |
| Elegoo Arduino Mega | Amazon | $ 14.99 | 1 | $ 14.99 | Arduino Mega for Motor Control |
| HiLetgo RFID Kit | Amazon | $ 5.49 | 1 | $ 5.49 | RFID Tag Reader |
| Yarongtech Mifare Classic 1K Dia 25mm adhesive RFID Sticker | Amazon | $ 11.99 | 1 | $ 11.99 | RFID Tags |
| Longruner Mechanical Endstop Limit Switch | Amazon | $ 9.59 | 1 | $ 9.59 | Limit Switches for Base and Main Box |
| Power Supply 12V 2A | Amazon | $ 6.99 | 4 | $ 27.96 | Power Supplies |
| Standard LCD 20X4 | Digikey | $ 17.95 | 1 | $ 17.95 | Controller LCD screen |
| I2C/SPI Character LCD Backpack | Digikey | $ 9.95 | 1 | $ 9.95 | Controller LCD screen Backpack |
| Sparkfun Keypad - 12 Button | Digikey | $ 3.95 | 1 | $ 3.95 | Controller Keypad |
| Bicycle Playing Cards | Amazon | $ 4.00 | 1 | $ 4.00 | Playing Cards |
| Electronics-Salon Screw Terminal Block for Arduino MEGA | Amazon | $ 32.00 | 1 | $ 32.00 | Arduino Mega Adapter for Screw-In Ports (Premium) |
| Snap-In Supports | McMaster-Carr | $ 3.00 | 1 | $ 3.00 | RFID and Arduino Supports (Premium) |
| 1-to-4 Power Cord Splitter | Amazon | $ 10.49 | 1 | $ 10.49 | Power Strip (Premium |
| Vinyl Wood Wrap | Amazon | $ 10.00 | 1 | $ 10.00 | Base Wood Wrap Stickers (Premium) |
| Snap-In Plastic Cord Grips | McMaster-Carr | $ 13.83 | 1 | $ 13.83 | Wire Stress Relief Clips (Premium) |
| Tech Flex Braided Cable Sleeve | Amazon | $ 7.49 | 1 | $ 7.49 | Wire Wrap (Premium) |

| | |
|---|---|
| **Standard Total** | **$ 567.82** |
| **Total with Premium Add-Ons** | **$ 644.63** |

Figure 16: Card Dealer 3000 Cost Estimate

## 5.13 Codes and Standards

The Card Dealer 3000 follows the standard ISO 8124-1:2018 for safety aspects related to mechanical and physical properties of toys. This standard applies to any product intended for use by children fourteen and under, and it applies to the toy as it is received by a consumer and also considering a reasonable wear and tear on the toy after its purchase. The structural characteristics of a toy such as its shape, size, contour, spacing, sharp edges must be considered as well as maximum kinetic energy values for projectiles. The base of the Card Dealer 3000 is constructed by bolted into two acrylic plates by half inch aluminum hex rods that are covered by two tenths of an inch of PLA. This protects the consumer from the base motor and provides the dealer with long-lasting structural support. The printed items and cut acrylic items on the top of the base have filleted edges which limit the dangers associated with sharp edges. All the motors on top of the base are also fully enclosed to discourage users from coming into contact with moving parts. Appropriate warnings and instructions will be provided with the packaging of the Card Dealer 3000 if it reaches production. The card shooter motor is a low torque motor which limits the force of the card being ejected from the dealer, but it also will be labelled with a warning to discourage consumers from getting too close to the shooter. In production, the Card Dealer 3000 would be encased in a platic or fiberglass cover to prevent contact with moving parts.

The Card Dealer 3000 also complies with the standard ISO / ASTM 52910 for design requirements, guidelines and recommendations in additive manufacturing. This standard covers the requirements, guidelines and recommendations applicable for uses of additive manufacturing in product design. This standard

applies to designers who are using additive manufacturing as a fabrication process as well as to students who are are learning about computer aided design and the design of mechanical systems. This standard supports the identification and guidance of issues in additive manufacturing but it is still the responsibility of users of additive manufacturing processes to follow proper safety practices.

Finally, the Card Dealer 3000 complies with the standard ASME Y14.5 - 2009 for dimensioning and tolerancing. This standard establishes a uniform practice for presenting dimensions and tolerances in engineering drawings. The standardization of dimension presentation allows for an easier manufacturing process and can improve the quality of the product. This standard is intended for mechanical design and manufacturing for project engineers, CAD specialists, and those involved in education. Adhering to this standard was paramount for the Card Dealer 3000 due to the tight tolerancing necessary in the movement of playing cards. Slight variations in tolerances for certain items can be the difference between the lack of control of the cards and the jamming of cards.

# 6 Conclusion

If the Dealin' Dirty Team was tasked to redesign the Card Dealer 3000, a few changes would be made to place more focus on mass production. All 3D printed parts would be slightly altered to ensure they could all be injection molded. Cheaper material would be chosen for all parts that do not come in contact with the playing cards. Since acrylic has such a low coefficient of friction, it was a suitable surface for the playing cards to travel on, but it is an expensive material. An alternative for acrylic may be difficult to find but research would be conducted to find something that is cheaper but also has the same surface properties. Another thing we would work on is the size of the device. The Card Dealer 3000 is a little over a foot tall and about a foot and a half in diameter, which is able to fit comfortably on a poker table, but it would be ideal if it was more compact. Lastly, Bluetooth technology would be implemented for the hand held controller so there is not a long wire attached to the dealer, which sometimes blocked the card from being shot fully across the table. In the end, a device consisting everything the Dealin' Dirty Team aimed for and more was created. An intelligent autonomous card dealer with a 99.8% success rate is something we didn't think was feasible due to the small thickness of playing cards and the difficult required to handle and move them. This engineering feat overcame all those challenges due to the relentless effort of Team Dealin' Dirty and a product was produced that could one day change the future of game nights all over the world. The Card Dealer 3000 is an elegant, lightweight, and portable device that is simple to use with its sleek user interface embedded in a hand held controller. Cutting edge RFID technology alerts the dealer to the cards that are dealt and allows the opportunity for players to face off with the dealer in games such as Blackjack and Casino War. Card games requiring face-up cards to be flipped are easily accommodated due to the dealer's efficient flipping mechanism. The dealer also has the ability to shoot cards directly to every player sitting at the table.

# Acknowledgements

Thanks to Dr. Joshua Browne and PhD Candidate Joni Mici for their guidance. Special thanks to Robert Stark, William Miller, and Andrei Shylo from the Columbia University machine shop staff for their assistance in designing and manufacturing the Card Dealer 3000.

# References


[1] Medium.com. *https://medium.com/@providence.casino/how-big-the-casino-industry-really-is-6554905b4f11* , Website.

[2] homepokergames.com. *https://www.homepokergames.com/rfid-casino-chips.php* , Website.

[3] Wheel-R-Dealer. *https://www.youtube.com/watch?v=9Df416dSuxw* , Website.

[4] UC Berkeley. *https://www.youtube.com/watch?v=dtiFUmgmdVQ* , Website.

[5] Physics Stack Exchange. *https://physics.stackexchange.com/questions/67957/stability-of-rotation-of-a-rectangular-prism/68359* , Website.

[6] Mouser.com. *https://www.mouser.com/ProductDetail/Arduino/A000067?qs=gMoqXxk, Website.*


# 7 Appendix

## 7.1 Appendix A: Python Script to Optimize Card Shoe Spring Constant

This is a python program that optimized our spring force as the thickness of the deck decreased over time (as cards were getting picked out of the shoe).

```
import numpy as np
weight = 0.22 # pound force
width = 2.44 # inch
height = 3.46 # inch
# dimension of a card
cWeight = weight/52
cWidth = width/52
cHeight = height/52
# card shoe dimensions
angle = np.radians(24)
maxLength = 2.5 #inch
blockWeight = .156 # lb force
cOF = 0.2 # coefficient of friction
# this is the perpendicular force the deck of cards puts onto the block
weightOnBlock = weight * np.cos(angle)
maxCOS = 0.0 # max spring constant
num = 0. # simple counter
maxSpringForce = 0 # max spring force
curLength = maxLength # place holder
currWeightOnBlock = weightOnBlock # place holder
for i in range(52):
       # use sum of the forces in the Y direction to get normal
       normalForce = currWeightOnBlock * np.cos(angle) + blockWeight
       friction = cOF * normalForce
       fSpring = currWeightOnBlock * np.sin(angle) - friction
       cOS = fSpring / curLength
       if cOS >= maxCOS:
              maxCOS = cOS
              num = i
              maxSpringForce = fSpring
       curLength = curLength - cWidth * np.cos(np.pi / 2 - angle) # adjust the compression
       currWeightOnBlock = currWeightOnBlock - cWeight * np.cos(angle) # adjust the force due to card weight
```

## 7.2 Appendix B: Dealin' Dirty Program

This is the master arduino code that runs all aspects of the Card Dealer 3000.

```
/*
 * Author: Dealin' Dirty: Columbia University Senior Design Group 13, Class of 2019
 * This program runs the Card Dealer 300, automatic card dealer and interactive gaming systemn
 * Five games are encorperated into the device
 */



// _____________________________________________Library
    ↪ Packages_____________________________________________________________
#include "Wire.h"
#include "Adafruit_LiquidCrystal.h"
#include "Keypad.h"
#include <Stepper.h>


//_____________________________________________Global
    ↪ Variables____________________________________________________________
int numStepsPush = 200; // The number of steps is specific to your motor
int numStepsSlide = 200; // The number of steps is specific to your motor
int numStepsPick = 200; // The number of steps is specific to your motor
float rollerRadius = 0.32; // what is the radius in inches of our rollers
int pulsesPerRev =16 * 131; // the gear ration is 131 // the gear ratio is 131 . This is adjusted for error
volatile long counter = 0; // this will change in the background



int topSpeed = 255; // 255 is the maximum speed
int lowSpeed = 130; // play around with this number
float distance = 3.2; // How far in inches do you want to travel
int baseSpeed = 160; // base speed
int homeSpeed = 140; // homing speed for the base



char dir; // base motor initialize
char dirBase; // base motor initialize
char reset = 'P';
int screenNum = 0; // initialize ithere

//// list of games
char gameOptions[] = {'1' , '2' , '3', '4', '5'}; // this is how we make sure we selected a game which is
    ↪ availible
int numberOfGames = 5;
//_____________________________________________Define
    ↪ Pins_________________________________________________________________

// encoder pins
 const int outputA = 2;
 const int outputB = 3;

// endstop pins
 int endStopTop = 13;
 int endStopBot = 4;

//_____________________________________________LCD Set
    ↪ UP___________________________________________________________________

// Connect via i2c, default address #0 (A0-A2 not jumpered)
Adafruit_LiquidCrystal lcd(0);
```

```
//_____________________________________________Keypad Set
    ↪ UP_____________________________________________________________

const byte ROWS = 4; //four rows
const byte COLS = 3; //three columns
char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
//byte rowPins[ROWS] = {37 , 43 , 45 , 47}; //connect to the row pinouts of the keypad . (mega)
//byte colPins[COLS] = {35 , 39 ,43}; //connect to the column pinouts of the keypad . (mega)
//Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

byte rowPins[ROWS] = {39 , 45 , 47 , 49}; //connect to the row pinouts of the keypad . (mega)
byte colPins[COLS] = {37 , 41 ,43}; //connect to the column pinouts of the keypad . (mega)
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

//_____________________________________________Stepper Motor Set
    ↪ UP_____________________________________________________________


// initialize pins for card pusher stepper motor
const int stepPinPush = 28;
const int dirPinPush = 29;
const int enPinPush = 30;

// initialize pins for card picker stepper motor
const int stepPinPick = 22;
const int dirPinPick = 23;
const int enPinPick = 24;

// initialize pins for Slide stepper motor
const int stepPinSlide = 25;
const int dirPinSlide = 26;
const int enPinSlide = 27;

//_____________________________________________DC Motor Set
    ↪ UP_____________________________________________________________

// shooter motor
int enB = 5;
int in3 = 7;
int in4 = 6;

// Base motor
int enA = 10;
int in1 = 11;
int in2 = 12;

//_____________________________________________RFID Set
    ↪ Up_____________________________________________________________

#include <SPI.h>
#include <MFRC522.h>

#define RST_PIN 8 // Configurable, see typical pin layout above
#define SS_PIN 9 // Configurable, see typical pin layout above

MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance

//_____________________________________________Void Set
    ↪ Up_____________________________________________________________
```

```
void setup() {

  Serial.begin(115200);

// Set up for RFID
  SPI.begin(); // Init SPI bus
  mfrc522.PCD_Init(); // Init MFRC522 card
// initialize endstop pins
  pinMode(endStopTop , INPUT);

// initialize both encoder input pins
  pinMode (outputA,INPUT);
  pinMode (outputB,INPUT);

// initialize endstop pins
  pinMode(endStopTop , INPUT);
  pinMode(endStopBot , INPUT);
// initialize the DC motor pins
  pinMode(enA , OUTPUT);
  pinMode(in1 , OUTPUT);
  pinMode(in2 , OUTPUT);
  pinMode(enB , OUTPUT);
  pinMode(in3 , OUTPUT);
  pinMode(in4 , OUTPUT);

// initialize the Slide Stepper pins
  pinMode(stepPinSlide,OUTPUT);
  pinMode(dirPinSlide,OUTPUT);

  pinMode(enPinSlide,OUTPUT);
  digitalWrite(enPinSlide,LOW); // this powers on the motor . (could use it to turn off during program)

// initialize the pick Stepper pins
  pinMode(stepPinPick,OUTPUT);
  pinMode(dirPinPick,OUTPUT);

  pinMode(enPinPick,OUTPUT);
  digitalWrite(enPinPick,LOW);


// initialize the push Stepper pins
  pinMode(stepPinPush,OUTPUT);
  pinMode(dirPinPush,OUTPUT);

  pinMode(enPinPush,OUTPUT);
  digitalWrite(enPinPush,LOW);



// Attatch interrupts to pin 2
  attachInterrupt(0, pin_A, RISING);

// set up the LCD's number of rows and columns:
  lcd.begin(4, 20);

// start with key set to No_KEY
  char key = NO_KEY;




}
//_____________________________________________Void Loop
    ↪ ____________________________________________________________________
```

```
void loop() {
// initialize variables
  bool gameFinished = false;
  int numPlayers;
  char gameChoice = '0';


// home screen
  while (screenNum == 0) {
     gameChoice = '0';
     screenNum = startScreen(); // always returns 1
  }
// get number of players
  while (screenNum == 1) {

    gameChoice = '0';
    numPlayers = getPlayers();
    if (numPlayers == 0) {
      screenNum = 0;
      break;
    }
    else {
    screenNum = 2;
    }
  }
// select your game
  while (screenNum ==2) {
    if (gameChoice == '0') {
      gameChoice = selectGame();
    }
    // This should return to the previous screen
    else if (gameChoice == '*') {

      screenNum = 1;

      }

// if they choose Texas holdem
    else if ((gameChoice =='1') && (gameFinished == false)) {
      gameChoice = checkGame(gameChoice);
      if ((gameChoice =='1') && (gameFinished == false)) {
        homing();
        gameFinished = texasHoldem(numPlayers);
      }
    }
// if they choose blackjack
    else if ((gameChoice =='2') && (gameFinished == false)) {
      gameChoice = checkGame(gameChoice);
      if ((gameChoice =='2') && (gameFinished == false)) {
        homing();
        gameFinished = blackJack(numPlayers);
      }
    }
// if they choose Cacino War
    else if ((gameChoice =='3') && (gameFinished == false)) {
      gameChoice = checkGame(gameChoice);
      if ((gameChoice =='3') && (gameFinished == false)) {
       homing();
       gameFinished = casinoWar(numPlayers);
      }
    }
// if they Choose Five Card Draw
    else if ((gameChoice =='4') && (gameFinished == false)) {
      gameChoice = checkGame(gameChoice);
```

```
    if ((gameChoice =='4') && (gameFinished == false)) {
      homing();
      gameFinished = five_card_draw(numPlayers);
    }
   }

// if they Choose Go Fish
   else if ((gameChoice =='5') && (gameFinished == false)) {
     gameChoice = checkGame(gameChoice);
     if ((gameChoice =='5') && (gameFinished == false)) {
       homing();
       gameFinished = go_fish(numPlayers);
     }
   }


  else if (gameFinished == true) {

     screenNum = closingScreen(gameChoice);
     gameFinished = false;
  }

 }

}


//_____________________________________________LCD Screen
    ↪ Functions_____________________________________________________________________

// starting screen. This takes in nothing, but will return the number of players
int startScreen() {
  char key = NO_KEY; // this indicates that the key will be selected from the keypad




while ( key == NO_KEY) { // while no key has been selected, present the opening screen
  lcd.setCursor(0,0);
  lcd.print(" DEALIN' DIRTY");
  lcd.setCursor(0,1);
  lcd.print(" CARD DEALER 3000");
  lcd.setCursor(0,2);
  lcd.print(" PRESS ANY KEY!");
  key = keypad.getKey();

}
  lcd.clear(); // . clear the screen and set the key variable to something not on the keypad

                // . clear the screen and set the key variable to something not on the keypad
   return 1;
}


int getPlayers() {

  int numPlayers = 0; // initialize the number of players to be zero
  int shift2 = 0; // this will be used to toggle
  char key = keypad.getKey(); // we will be getting the value from the keypad
  while (numPlayers == 0) { // This loop will get the number of players

    if (shift2 == 0 ) { // prompt the user
    lcd.setCursor(0,0);
    lcd.print(" SELECT THE NUMBER ");
```

```
    lcd.setCursor(0,1);
    lcd.print(" OF PLAYERS ");
    lcd.setCursor(0,3);
    lcd.print("<- * ");
    key = keypad.getKey(); // we will be getting the value from the keypad
      if (key == '#' || key == '0' || key == '9' || key == '8' || key == '7') { // these are all the situations
           ↪ where we return an error message
        shift2 = 1;
    }


      if (key == '1' || key == '2' || key == '3' || key == '4' || key == '5' || key == '6') {
        shift2 = 2;
      }
      if ( key == '*') {
        lcd.clear();
        return 0;
      }
    }




    else if (shift2 == 1) { // these are all the situations where we return an error message
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print(" SELECT A NUMBER ");
      lcd.setCursor(0,1);
      lcd.print(" BETWEEN 1 AND 6! ");
      delay(1000);
      lcd.clear();
      key = reset;
      shift2 =0;
    }
    else { // return the number of players as an integer
      String number = String(key);
      lcd.clear();
      numPlayers = number.toInt();

    }
}

return numPlayers;
}


// Game Select. This will initiate after the number of players has been indicated

char selectGame() {

int shift = 0; // this will be used to toggle
bool gameSel = false; // This will be used to terminate this function
char key = keypad.getKey();

while ((gameSel== false) ) { // when a game has not been selected
  key = keypad.getKey();

    if (shift ==0) { // the first screemn

      lcd.setCursor(0,0);
      lcd.print(" SELECT A GAME");
      lcd.setCursor(0,1);
      lcd.print("1 TEXAS HOLD'EM");
      lcd.setCursor(0,2);
      lcd.print("2 BLACKJACK");
```

```
    lcd.setCursor(0,3);
    lcd.print(" <- * # ->");

     if (key == '#') { // use the number sign to move forward
      shift = shift + 1;
      lcd.clear();
      key = reset;

    }
    else if (key == '*') {
      lcd.clear();
      return key;

    }
    else {
      for( int i=0; i< numberOfGames; i++) // at any time, a player can select their game (even if it isn't
          ↪ showing on their screen)
        if ( gameOptions[i] == key){
          lcd.clear();
          gameSel = true; // terminates the while loop

  }
    }

  }

  else if (shift == 1) { // this is the second screen
  lcd.setCursor(0,0);
  lcd.print("3 CASINO WAR");
  lcd.setCursor(0,1);
  lcd.print("4 FIVE CARD DRAW");
  lcd.setCursor(0,2);
  lcd.print("5 Go Fish");
  lcd.setCursor(0,3);
  lcd.print(" <- * ");

    if (key == '*') { // use the star symbol to move to the previous screen
      shift = shift - 1;
      key = reset;
      lcd.clear();
    }
    else {
      for( int i=0; i< numberOfGames; i++) // at any time, a player can select their game (even if it isn't
          ↪ showing on their screen)
        if ( gameOptions[i] == key){
          lcd.clear();
          gameSel = true; // terminates the while loop


  }
    }
    }

  }

  return key; // returns the selected game key
}


/////// Code for the closing screen interface /////////

char closingScreen(char number) {
  char key = keypad.getKey();

  while ((key != '*') && (key != '#')) {
```

```
    lcd.setCursor(0,0);
    lcd.print("THANKS FOR PLAYING!");
    lcd.setCursor(0,2);
    lcd.print("PRESS # TO RESTART");
    lcd.setCursor(0,3);
    lcd.print("PRESS * FOR HOME");
    key = keypad.getKey();
  }

  if (key == '*') {
    lcd.clear();
    return 0;
      }
  else if (key == '#') {
    lcd.clear();
    return 2;
  }

}

//_____________________________________________Confirm Screen
    ↪ Functions_____________________________________________________________
//// Programs Used to doubkle check User's Game Choice

char checkGame(char number) {
/*
 * this function confirms the game selection
 * it takes in the char which was selected for the game during the game select screen
 * it either returns '0' to return to the previous games or the char that was brought in
 */
  String gameTitle;
  char key;

  if (number == '1') {
    gameTitle = " TEXAS HOLD'EM";
  }
  else if (number == '2') {
    gameTitle = " BLACKJACK";
  }
  else if (number =='3') {
    gameTitle = " CASINO WAR";
  }
  else if (number == '4') {
    gameTitle = " FIVE CARD POKER";
  }
  else if ( number == '5') {
    gameTitle = " GO FISH";
  }
  while ((key != '*') && (key != '#')) {

    lcd.setCursor(0,0);
    lcd.print(gameTitle);
    lcd.setCursor(0,1);
    lcd.print("#) CONFIRM");
    lcd.setCursor(0,2);
    lcd.print("*) RETURN");
    key = keypad.getKey();
  }

  if (key == '*') {
    lcd.clear();
    return '0';
      }
  else if (key == '#') {
```

```
    lcd.clear();
    return number;
  }
}



//_____________________________________________GamePlay
    ↪ Functions_____________________________________________________________________


//_______________________________Texas Holdem___________________________________

bool texasHoldem(int numPlayers) {

  bool gameFinished = false; // this will terminate the game
  char key = NO_KEY; // we will be getting keys from the keypad
  int betNumber = 0; // this will be used for the different betting periods

//// These are the variables which indicate base motor movement for the deal ////////

  int num = numPlayers - 1;
  int turn = (180 / numPlayers) ;
  float adjustFloat = turn / 2.0;
  int adjust = floor(adjustFloat);
  if (numPlayers ==0) {
    turn = 0;
  }
  int init = 90 - adjust; // this is useful for a dealer starting directly center
  int backToStart = turn * (num) ; // first iteration of the deal
  int backToStart2 = turn * (num + 1); // second iteration of the deal
  int burn = 58 ; // this is the distance from horizontal
  int flopStart = 70 ; // initial location of flop
  int cardSpace = 12 ; // spacing between cards on flop, turn, river
  int tempBurn1 = backToStart + adjust - burn; // adjustment variable for first burn
  int moveToFlop = flopStart - burn; // adjustment variable for the flop
  int tempBurn2 = 2* cardSpace + flopStart - burn; // adjustment variable for the second burn
  int turnLoc = flopStart + cardSpace*3 - burn; // adjustment variable for the turn
  int tempBurn3 = 3* cardSpace + flopStart - burn; // adjustment variable for the third burn
  int riverLoc = flopStart + cardSpace*4 - burn; // adjustment variable for the river
/////// This is the Deal //////////
  moveBase(false , adjust, baseSpeed); // move to first player
  dealCard(topSpeed); // Deal a Card close to the player

  for (int j = 0; j<=1 ; j++){
    if (j==0) { // this is the first round
      for (int i=0; i<=num-1; i++){
        moveBase(false , turn , baseSpeed ); // move to the next player
        dealCard(topSpeed); // Deal a Card close to the player
      }
    moveBase(true, backToStart , baseSpeed ); // move to first player
    }

  else{ // this is the second round
    for (int p=0; p<=num-1; p++){
      dealCard(topSpeed);
      moveBase(false , turn, baseSpeed );
      }
  dealCard(topSpeed); // Deal a Card close to the final player

  }
}
```

```
   moveBase(true , tempBurn1 , baseSpeed ); // move to burn pile

betNumber = 1; // start the first betting period

while ( betNumber ==1) { // wait for the players to finishing betting
   key = keypad.getKey();

  if ((key != '#') && (key != '*')) {
  lcd.setCursor(0,0);
  lcd.print(" RAISE! FOLD! CALL! ");
  lcd.setCursor(0,2);
  lcd.print(" *) End Round ");
  lcd.setCursor(0,3);
  lcd.print(" # ->");

  }
  else if (key == '#') { // the players are ready, reset the variables and clear the screen
    lcd.clear();
    key = NO_KEY;
    betNumber = 0;
  }
  else {
    gameFinished = true;
    lcd.clear();
    return gameFinished;
  }
}
// Burn

  dealCard(lowSpeed); // Burn a Card close to the dealer

// Flop

   moveBase(false , moveToFlop , baseSpeed );
   slideTop(false); // Needs to be face up
   dealCard(lowSpeed); // deal first card
   for ( int k = 0 ; k <=1 ; k++){ // deal 2 more cards
    moveBase(false , cardSpace , baseSpeed );
    dealCard(lowSpeed); // Deal a Card close to the dealer
   }

// burn another facedown Card
  moveBase(true , tempBurn2 , baseSpeed ); // move to burn pile
  slideTop(true); // needs to be facedown

betNumber = 2; // start the second betting period

while ( betNumber ==2) { // wait for the players to finishing betting
  key = keypad.getKey();

  if ((key != '#') && (key != '*')) {
  lcd.setCursor(0,0);
  lcd.print(" RAISE! FOLD! CALL! ");
  lcd.setCursor(0,2);
  lcd.print(" *) End Round ");
  lcd.setCursor(0,3);
  lcd.print(" # ->");

  }
  else if (key == '#') { // the players are ready, reset the variables and clear the screen
    lcd.clear();
    key = NO_KEY;
    betNumber = 0;
  }
  else {
```

```
    gameFinished = true;
    lcd.clear();
    return gameFinished;
  }
}

  dealCard(lowSpeed); // Burn a Card close to the dealer

// NOW DO THE TURN
  moveBase(false , turnLoc , baseSpeed ); // go to turn position
  slideTop(false); // card needs to be face up
  dealCard(lowSpeed); // deal card close to the dealer

// burn one face down card
  moveBase(true , tempBurn3 , baseSpeed ); // move to burn pile
  slideTop(true); // card needs to be face down


betNumber = 3;
while ( betNumber ==3) { // wait for the players to finishing betting
   key = keypad.getKey();

  if ((key != '#') && (key != '*')) {
  lcd.setCursor(0,0);
  lcd.print(" RAISE! FOLD! CALL! ");
  lcd.setCursor(0,2);
  lcd.print(" *) End Round ");
  lcd.setCursor(0,3);
  lcd.print(" # ->");

  }
  else if (key == '#') { // the players are ready, reset the variables and clear the screen
    lcd.clear();
    key = NO_KEY;
    betNumber = 0;
  }
  else {
    gameFinished = true;
    lcd.clear();
    return gameFinished;
  }
}


  dealCard(lowSpeed); // Burn a card close to the dealer

// NOW DO THE RIVER
 moveBase(false , riverLoc , baseSpeed ); // move to the river location
 slideTop(false); // the card needs to be face up
 dealCard(lowSpeed); // deal a card close to the dealer

 betNumber = 4;
while ( betNumber ==4) { // wait for the players to finishing betting
   key = keypad.getKey();

  if ((key != '#') && (key != '*')) {
  lcd.setCursor(0,0);
  lcd.print(" RAISE! FOLD! CALL! ");
  lcd.setCursor(0,3);
  lcd.print(" # ->");

  }
  else if (key == '#') { // the players are ready, reset the variables and clear the screen
    lcd.clear();
    key = NO_KEY;
```

```
    betNumber = 0;
  }
}

  gameFinished = true; // tell the system the game is concluded
  lcd.clear(); // clear the screen

  return gameFinished;
}

// ____________________________________________________________________________________Black
    ↪ Jack_____________________________________________________________________________________
//_________________________________Game_____________________________________________________

bool blackJack(int numPlayers) { // Thjs requires RFID
/*
 * This function will play a game of Black Jack
 * The dealer will hold on 17 or higher. Ace, 6 is a hold for the dealer
 * The dealer's First card is face up. second is face down
 *
 */

  bool gameFinished = false;
// Variables needed for base motion
  int num = numPlayers - 1;
  int turn = (180 / numPlayers) ;
  float adjustFloat = turn / 2.0;
  int adjust = floor(adjustFloat);
  if (numPlayers ==0) {
    turn = 0;
  }
  int init = 90 - adjust; // this is useful for a dealer starting directly center
  int backToStart = turn * (num) ; // first iteration of the deal
  int backToStart2 = turn * (num + 1); // second iteration of the deal
  int lastPlayerToDealer = turn * (num) - init ;
  int dealerToFirst = init;

// variables needed for game play
  int playerCard1[numPlayers] = {0}; // initialize array of player first card
  int playerCard2[numPlayers] = {0}; // initialize array of player second card
  int playerScores[numPlayers] = {0}; // initialize array of player scores
  String playerCard1Str[numPlayers] = {"0"};
  String playerCard2Str[numPlayers] = {"0"};
  int playerSplitScores[numPlayers] = {0}; // scores for second hand if the player splits
  int dealerScore = 0; // initialize score of dealer
  int dealerCard1 = 0; // initialize dealer's first card
  int dealerCard2 = 0;

  String card; // card place holder (will change)
  int cardValue; // card value place holder (will change)
  int currScore; // another useful place holder (will change)
  bool moveOn; // useful when prompting players
  int hitStay; // used to give players answer for hit_stay_double_split
  bool split; // used if the player has opportunity to split
  bool playerDouble; // used to see if player wants to double

/////// This is the Deal //////////
  slideTop(false); // card needs to be face up
  moveBase(false , adjust , baseSpeed); // move to first player
  card = dealCard(topSpeed); // Deal a Card close to the player
  cardValue = blackJackGetCard(card); // get card Value
  currScore = playerScores[0] + cardValue; // update the player's score
  playerScores[0] = currScore;
  playerCard1[0] = cardValue; // update player's first card
  playerCard1Str[0] = card;
```

```
for (int j = 0; j<=1 ; j++){
  if (j==0) { // this is the first round
    for (int i=1; i<=num; i++){
      int currPlayerNumber = i +1;
      moveBase(false , turn , baseSpeed ); // move to the next player
      card = dealCard(topSpeed); // Deal a Card close to the player
      cardValue = blackJackGetCard(card); // get card Value
      currScore = playerScores[i] + cardValue; // update the player's score
      playerScores[i] = currScore;
      playerCard1[i] = cardValue; // update player's first card
      playerCard1Str[i] = card;


     }
    moveBase(true , lastPlayerToDealer , baseSpeed ); // move to dealer
    card = dealCard(lowSpeed); // deal a card close to dealer
    cardValue = blackJackGetCard(card); // get card Value
    dealerScore = dealerScore + cardValue; // update dealer's score
    dealerCard1 = dealerCard1 + cardValue; // update the dealer's first card


    moveBase(true, dealerToFirst , baseSpeed ); // move to first player
  }

  else{ // this is the second round
    for (int p=0; p<num; p++){
      int currPlayerNumber = p +1;
      card = dealCard(topSpeed); // Deal a Card close to the player
      cardValue = blackJackGetCard(card); // get card Value
      currScore = playerScores[p] + cardValue; // update the player's score
      playerScores[p] = currScore;
      playerCard2[p] = cardValue; // update player's second card
      playerCard2Str[p] = card;
      moveBase(false , turn , baseSpeed );

      }
  card = dealCard(topSpeed); // Deal a Card close to the last player
  cardValue = blackJackGetCard(card); // get card Value
  currScore = playerScores[num] + cardValue; // update the player's score

  playerScores[num] = currScore;
  playerCard2[num] = cardValue; // update player's second card
  playerCard2Str[num] = card;

  moveBase(true , lastPlayerToDealer , baseSpeed ); // move to Dealer
  slideTop(true);
  card = dealCard(lowSpeed); // deal a card close to dealer
  cardValue = blackJackGetCard(card); // get card Value
  dealerScore = dealerScore + cardValue; // update dealer's score
  dealerCard2 = dealerCard2 + cardValue; // update the dealer's first card
  moveBase(true , dealerToFirst , baseSpeed ); // move to first player
  slideTop(false); // face up the rest of the game
  }
}
```

```
////// GAME PLAY Set up ////////

// check to see if the dealer should offer an insurance
  if (dealerCard1 == 11) {
    insurance(); // wait for players to make insurance bets

  }
    if (dealerScore == 21) { // the dealer has blackJack
       dealerBlackJack();
       gameFinished = true;
       return gameFinished;
    }

  int currPosition = 1; // need some positional awareness

// need a loop that will check ahead of time whether or not the player will split
// it also needs to get the number of acs
int splits[numPlayers] = {0};
int numAces[numPlayers] = {0};
  for (int l = 0; l < numPlayers; l ++ ) {
    int playerNum = l + 1; // need the acutal player number
    int card1 = playerCard1[l]; // get player information
    int card2 = playerCard2[l];
    int score = playerScores[l];
    String card1Str = playerCard1Str[l];
    String card2Str = playerCard2Str[l];
    int playerAce = 0;
    if (card1 == 11) {
      playerAce = playerAce +1 ;
    }
    if (card2 == 11) {
      playerAce = playerAce +1 ;
    }
    numAces[l] = playerAce;

        split = false; // only will change if player decides to split
        if (card1 == card2) { // if the player can split
          if (card1 != 10) {

            splits[l] = 1;

          }
          else {
            if (((card1Str.indexOf("1") > 0) && (card2Str.indexOf("1") > 0)) || ((card1Str.indexOf("K") > 0) &&
                 ↪  (card2Str.indexOf("K") > 0)) || ((card1Str.indexOf("Q") > 0) && (card2Str.indexOf("Q") >
                 ↪ 0)) || ((card1Str.indexOf("J") > 0) && (card2Str.indexOf("J") > 0)) ){

            splits[l] = 1;
```

```
          }
          else {

            continue;
          }
        }
      }
  }



// Game Play ////

// see which player wants to hit another card
  for (int k = 0; k < numPlayers; k ++ ) {
    int playerNum = k + 1; // need the acutal player number
    int card1 = playerCard1[k]; // get player information
    int card2 = playerCard2[k];
    int score = playerScores[k];
    String card1Str = playerCard1Str[k];
    String card2Str = playerCard2Str[k];



    // is there a blackJack
       if (score == 21) { // black Jack!
         black_jack_display(playerNum); // display screen .
         continue;
       }
       int currPositionAngle = currPosition * turn + adjust;
       int playerAngle = playerNum * turn + adjust;
       int moveDiff = playerAngle - currPositionAngle ;

        moveBase(false, moveDiff , baseSpeed);
        currPosition = playerNum;


       // see if player wants to split
       split = false; // only will change if player decides to split
       if (splits[k] ==1) {
          split = playerSplit(playerNum);
        }
        else {

            split = false;
          }



       if (split == true) {
         card = dealCard(topSpeed);
         cardValue = blackJackGetCard(card);
         int score1 = playerCard1[k] + cardValue;
         int score2 = playerCard2[k];
         if (playerCard1[k] != 11) {
         score1 = hit_stay_interaction(playerNum, score1 , numAces[k]);
         }
         playerScores[k] = score1;
         // second split
         moveBase(false , 8 , baseSpeed);
         String card2 = dealCard(topSpeed);
         int cardValue2 = blackJackGetCard(card2);
         score2 = score2 + cardValue2;
         if (playerCard1[k] != 11) {
```

```
        score2 = hit_stay_interaction(playerNum, score2 , numAces[k]);
        }
        playerSplitScores[k] = score2;
        moveBase(true , 8 , baseSpeed);
       }
       else if (split == false) {


          score = hit_stay_interaction(playerNum, score , numAces[k]);
          playerScores[k] = score;
       }
// if (k < numPlayers - 1) {
// moveBase(false, turn , baseSpeed);
// }


  }
// Logic to move back to the dealer
   int currPositionAngle = (currPosition -1) * turn + adjust;
   if (currPositionAngle <90) {
    int diff = 90 - currPositionAngle;
    moveBase( false , diff , baseSpeed);
   }
   else {
    int diff = currPositionAngle - 90 ;
    moveBase( true , diff , baseSpeed);
   }
  flip_dealer_card();
// if dealer's score is less than 17
  int dealerAces= 0;
  if (dealerCard1 ==11 ) {
    dealerAces = dealerAces + 1;
  }
  if (dealerCard2 == 11) {
    dealerAces = dealerAces + 1;
  }

   int numInGame = 0;
   for (int n = 0; n < numPlayers; n++) {
      if ((playerScores[n] < 22) && (playerScores[n] != 0)) {
        numInGame = numInGame + 1;
      }
      if ((playerSplitScores[n] < 22) && (playerSplitScores[n] !=0)) {
        numInGame = numInGame + 1;
      }
   }


  while ((dealerScore < 17) && (numInGame != 0)) {
    card = dealCard(lowSpeed); // deal card close to dealer
    cardValue = blackJackGetCard(card); // get card Value
    dealerScore = dealerScore + cardValue; // update dealer's score
    if (cardValue == 11) {
      dealerAces = dealerAces + 1;
    }







    if (dealerScore >21) {
      if (dealerAces != 0) {
        dealerScore = dealerScore - 10;
        dealerAces = dealerAces - 1;
```

```
    }
   }
  }

  displayDealerScore(dealerScore);



  int listOfWinners[numPlayers] = {0};



  for (int x = 0; x < numPlayers; x ++ ) {
    int currPlayerNum = x + 1;
    if ((playerScores[x] < 22) && (playerScores[x] > dealerScore) || (playerScores[x] < 22) && (dealerScore
        ↪ >= 22) ) {
      listOfWinners[x] = currPlayerNum;
    }
    if ((playerSplitScores[x] < 22) && (playerSplitScores[x] > dealerScore)) {
      listOfWinners[x] = currPlayerNum;
    }
  }

  int dealerWin = 0;
  for (int z = 0; z < numPlayers; z ++ ) {
    dealerWin = dealerWin + listOfWinners[z];
  }

  if ((dealerWin == 0 )&& (dealerScore <= 21)) {
    casinoWarDealerWins();
  }
  else {
    casinoWarWinners(listOfWinners , numPlayers * 2);
  }


  gameFinished = true;
  lcd.clear();

  return gameFinished;
}

// _____________________________________________________Helper
    ↪ functions_____________________________________________________

void flip_dealer_card() {
  /*
 * this function prompts the user to flip the dealer's card
 * it does not need to return anything
 */
   char key = NO_KEY;

  while (key != '#') {

    lcd.setCursor(0,0);
    lcd.print(" FLIP DEALER'S ");
    lcd.setCursor(0,1);
    lcd.print(" CARD ");
    lcd.setCursor(0,3);
    lcd.print(" # ->");
    key = keypad.getKey(); // we will be getting values from the keyPad
  }
  if (key == '#') {
    lcd.clear();
    return;
```

```
  }

}
```

```
int hit_stay_double(int player , int score) {
  /*
 * this function will ask the player if he/she would like to hit or stay or doub;e
 * it will return 0 1 or 2
 */
 bool answer = false;
 int numCards;
 String playerLine = get_line(player);

 while (answer == false) {
    lcd.setCursor(0,0);
    lcd.print(playerLine);
    lcd.setCursor(18 , 0);
    lcd.print(score);
    lcd.setCursor(0,1);
    lcd.print("1) Hit ");
    lcd.setCursor(0,2);
    lcd.print("2) Stay ");
    lcd.setCursor(0,3);
    lcd.print("3) Double ");
    char key = keypad.getKey(); // we will be getting the value from the keypad

    if (key == '1') {
      numCards = 1;
      answer = true;
    }
    else if (key == '2') {
      numCards = 0;
      answer = true;
    }

    else if (key == '3') {
      numCards = 3;
      answer = true;
    }


 }
 lcd.clear();
 return numCards;

}
```

```
int hit_stay_interaction(int player, int score , int numAces ) {
/*
 * this function is a loop which continuously promots the player if they would like to hit or stay.
 * it will break if the score exceeds 21
 */
```

```
 bool moveOn = false; // prep for interacting with the player
 bool hitStay;
 String card;
 int cardValue;

 if (score == 22) {
  score = 12;
  numAces = numAces - 1;
 }

 int choice = hit_stay_double(player, score);
 if (choice ==3) {
  card = dealCard(topSpeed); // player gets exactly one card
  cardValue = blackJackGetCard(card); // get card Value
  score = score + cardValue; // update player's score
  int playerAcesDouble = numAces;
  if (cardValue == 11) {
  playerAcesDouble = playerAcesDouble + 1;

  }
  if ((score > 21) && (playerAcesDouble != 0)) {
  score = score - 10;
  }

  return score;
 }


else if (choice == 0) {
    moveOn = true;
    return score;
 }

 else{
    card = dealCard(topSpeed); //non double deal the firs card
    cardValue = blackJackGetCard(card); // get card Value
    score = score + cardValue; // update player's score

       if (cardValue == 11) {
     numAces = numAces + 1;
    }
    if (score >= 22) {

     if (numAces != 0) {

       score = score -10;
       numAces = numAces - 1;
     }
     else {
       bust_display(player);
       return score; // the player has busted
    }

 }

 }

// this is non double

 while (moveOn == false) {
   hitStay = hit_or_stay(player , score);
   if (hitStay == 0) {
     moveOn = true;
     return score;
   }
```

```
    else {
      card = dealCard(topSpeed); // player gets exactly one card
      cardValue = blackJackGetCard(card); // get card Value
      score = score + cardValue; // update player's score

      if (cardValue == 11) {
       numAces = numAces + 1;
      }
      if (score >= 22) {

       if (numAces != 0) {

         score = score -10;
         numAces = numAces - 1;
         continue;
       }
       bust_display(player);
       return score; // the player has busted
      }
    }
  }



return score;

}



int hit_or_stay(int player , int score) {
/*
 * this function will ask the player if he/she would like to hit or stay
 * it will return 0 or 1
 */
 bool answer = false;
 int numCards;
 String playerLine = get_line(player);

 while (answer == false) {
   lcd.setCursor(0,0);
   lcd.print(playerLine);
   lcd.setCursor(18 , 0);
   lcd.print(score);
   lcd.setCursor(0,1);
   lcd.print("1) Hit ");
   lcd.setCursor(0,2);
   lcd.print("2) Stay ");
   char key = keypad.getKey(); // we will be getting the value from the keypad

   if (key == '1') {
     numCards = 1;
     answer = true;
   }
   else if (key == '2') {
     numCards = 0;
     answer = true;
   }

 }
 lcd.clear();
 return numCards;

}
```

```
void insurance() {
/*
 * this function prompts the user's for an insurance
 * it does not need to return anything
 */
   char key = NO_KEY;

  while (key != '#') {

    lcd.setCursor(0,0);
    lcd.print(" PLACE INSURANCE ");
    lcd.setCursor(0,1);
    lcd.print(" CHIPS ");
    lcd.setCursor(0,3);
    lcd.print(" # ->");
    key = keypad.getKey(); // we will be getting values from the keyPad
  }
  if (key == '#') {
    lcd.clear();
    return;
  }

}


void dealerBlackJack() {
  /*
   * This is the Dealer's black Jack screen.
   */

     char key = NO_KEY;

  while (key != '#') { // while loop waits for players to move forward
    lcd.setCursor(0,0);
    lcd.print(" CARD DEALER 3000 ");
    lcd.setCursor(0,1);
    lcd.print(" HAS BLACKJACK!");
    lcd.setCursor(0,3);
    lcd.print(" # ->");
    key = keypad.getKey(); // we will be getting values from the keyPad
  }

  if (key == '#') {
    lcd.clear();
    return;
  }

}



void displayDealerScore(int dealerScore) {
  /*
   * This is the Dealer's black Jack score.
   */

  char key = NO_KEY;

  while (key != '#') { // while loop waits for players to move forward
    lcd.setCursor(0,0);
    lcd.print(" CARD DEALER 3000 ");
    lcd.setCursor(9,1);
    lcd.print(dealerScore);
    lcd.setCursor(0,3);
```

```
    lcd.print(" # ->");
    key = keypad.getKey(); // we will be getting values from the keyPad
  }

  if (key == '#') {
    lcd.clear();
    return;
  }

}


bool playerDoublePrompt(int player) {
/*
 * this function will ask the player if he/she would like to double
 * it will return true or false
 */
 bool answer = false;
 bool playerDouble;
 String playerLine = get_line(player);

 while (answer == false) {
    lcd.setCursor(0,0);
    lcd.print(playerLine);
    lcd.setCursor(0,1);
    lcd.print("1) Double ");
    lcd.setCursor(0,2);
    lcd.print("2) Do Not Double ");
    char key = keypad.getKey(); // we will be getting the value from the keypad

    if (key == '1') {
      playerDouble = true;
      answer = true;
    }
    else if (key == '2') {
      playerDouble = false;
      answer = true;
    }

 }
 lcd.clear();
 return playerDouble;
}

bool playerSplit(int player) {
  /*
 * this function will ask the player if he/she would like to split
 * it will return true or false
 */

 bool answer = false;
 bool split;
 String playerLine = get_line(player);

 while (answer == false) {
    lcd.setCursor(0,0);
    lcd.print(playerLine);
    lcd.setCursor(0,1);
    lcd.print("1) Split ");
    lcd.setCursor(0,2);
    lcd.print("2) Do Not Split ");
    char key = keypad.getKey(); // we will be getting the value from the keypad

    if (key == '1') {
      split = true;
```

```
    answer = true;
   }
   else if (key == '2') {
    split = false;
    answer = true;
   }

 }
 lcd.clear();
 return split;

}



// ____________________________________________________________________________________________Casino War
    ↪ ____________________________________________________________________________________________
//_________________________________Game_____________________________________________________________


bool casinoWar(int numPlayers) { // this requires RFID

  bool gameFinished = false; // this will terminate the game
  char key = NO_KEY; // we will be getting keys from the keypad
//// These are the variables which indicate base motor movement for the deal ////////

  int num = numPlayers - 1;
  int turn = (180 / numPlayers) ;
  float adjustFloat = turn / 2.0;
  int adjust = floor(adjustFloat);
  if (numPlayers ==0) {
    turn = 0;
  }
  int init = 90 - adjust; // this is useful for a dealer starting directly center
  int backToStart = turn * (num) ; // first iteration of the deal

  int lastPlayerToDealer = turn * (num) - init ;
  int dealerToFirst = init;
  int toPlayer;
  int burnToDealer = 90;
  String card;
  int cardVal;
  String dealerCard;
  int dealerCardVal;
  int playerCards[numPlayers];
  int playerNumber;
  bool match;


/////// This is the Deal //////////
  moveBase(false , adjust , baseSpeed ); // move to first player
  slideTop(false); // cards are face up
  card = dealCard(topSpeed); // Deal a Card close to the player
  cardVal = casinoWarGetCard(card); // this needs to be integrated into dealCard . _____________________
      ↪ CHANGE HERE ____________________
  playerCards[0] = cardVal;
  for (int i=1; i<=num; i++){
    moveBase(false , turn , baseSpeed ); // move to the next player
    card = dealCard(topSpeed); // Deal a Card close to the player
    cardVal = casinoWarGetCard(card); // this needs to be integrated into dealCard . _____________________
        ↪ CHANGE HERE ____________________
    playerCards[i] = cardVal;
  }
  moveBase(true , lastPlayerToDealer , baseSpeed ); // move to dealer
```

```
    dealerCard = dealCard(lowSpeed);
    dealerCardVal = casinoWarGetCard(dealerCard); // this needs to be integrated into dealCard .
        ↪ ____________________ CHANGE HERE ____________________
    moveBase(true, burnToDealer , baseSpeed ); // move to burn pile



/// START GAMEPLAY AT Burn
  int listOfWinners[numPlayers] = {0};
  int listOfTies[numPlayers] = {0};

// STEP 1: GO THROUGH TABLE, SEE WHO WON, AND WHO TIED
  for(int i = 0; i < numPlayers; i++){
     int playerNumber = i + 1;
     bool match;
     if (dealerCardVal == playerCards[i]){
       listOfTies[i] = playerNumber;
     }

     else if (playerCards[i] > dealerCardVal) { // update winner list
      listOfWinners[i] = playerNumber;
     }
  }
// STEP 2: ASK EACH TIED PLAYER IF THEY WANT A MATH OR SURRENDER, THEN DEAL

// need a while loop for sequential ties.

//This loop checks to see if there are ties
int playersInWar = 0;

 for(int j = 0; j < numPlayers; j++){
    if (listOfTies[j] != 0) {
      playersInWar = playersInWar+1;
    }
 }
while (playersInWar != 0) {
 int listOfTiesScores[numPlayers] = {0};
 for(int j = 0; j < numPlayers; j++){
    if (listOfTies[j] == 0) {
      continue;
    }
    int playerNumber = j + 1;
    String newPlayerCard; // read by rfid
    int newPlayerCardVal = 0; // converted to int
    int toPlayer = 0;
    match = casinoWarTie(playerNumber); // see if player wants to match or surrender
    Serial.println(match);
    if (match == false) { // if surrender move to next player
       listOfTies[j] = 0;
       listOfTiesScores[j] = 0;
       playersInWar = playersInWar - 1; // update while loop vairable.
    }
    else {

       slideTop(true); // burn face down cards
       for (int k = 0; k < 3; k ++) {
         dealCard(lowSpeed);
       }
       toPlayer = adjust + (turn * j); // move back to player
       moveBase(false , toPlayer , baseSpeed );
       slideTop(false); // face up card
       newPlayerCard = dealCard(topSpeed);
       newPlayerCardVal = casinoWarGetCard(newPlayerCard);
       Serial.println("newPlayerCard");
       Serial.println(newPlayerCardVal);
```

```
      listOfTiesScores[j] = newPlayerCardVal;

      moveBase(true , toPlayer , baseSpeed ); // back to burn
      }

   }

// STEP 3: COMPARE NEW SCORES WITH DEALERS
  int doWeDeal = 0;
  for(int m = 0; m < numPlayers; m++){
    int playerNumber = m + 1;
    Serial.println("comparing with dealer");
    if (listOfTies[m] == 0) {
      continue;
    }
    else {
      doWeDeal = 1;
    }
  }

  if (doWeDeal ==1) {
       String newDealerCard; // read by rfid
       int newDealerCardVal = 0; // converted to int
       slideTop(true); // burn face down cards
       for (int k = 0; k < 3; k ++) {
          dealCard(lowSpeed);
       }
       moveBase(false , burnToDealer, baseSpeed ); // move to dealer
       slideTop(false);
       newDealerCard = dealCard(lowSpeed);
       newDealerCardVal = casinoWarGetCard(newDealerCard);
       Serial.println("newDealerCard");
       Serial.println(newDealerCardVal);

       for (int z = 0; z < numPlayers ; z++) {
       int playerNumber = z + 1;
       if ( listOfTiesScores[z] > newDealerCardVal) { // update winner list
            Serial.println("updateWinnerList and drop players in war");
            listOfWinners[z] = playerNumber;
            playersInWar = playersInWar -1;
            Serial.println(playersInWar);
            listOfTies[z] = 0;
       }
      else if ((listOfTiesScores[z] < newDealerCardVal) && (listOfTiesScores[z] != 0)) { // remove losers from
          ↪ the list of ties
         Serial.println("drop players in war");
         playersInWar = playersInWar -1;
         Serial.println(playersInWar);
         listOfTies[z] = 0;

      }
       }
   }
moveBase(true , burnToDealer, baseSpeed ); // move to burn
Serial.println("number in war is");
Serial.println(playersInWar);
}


// STEP FOUR: PROVIDE WINNERS
  int dealerWin = 0;
  for ( int p = 0; p < numPlayers; p ++) {
    int playerValue = listOfWinners[p];
    dealerWin = dealerWin + playerValue;
  }
```

```
  if (dealerWin == 0) {
    casinoWarDealerWins();
  }
  else {
    casinoWarWinners(listOfWinners , numPlayers);
  }

  gameFinished = true;
  lcd.clear();
  return gameFinished;
}







bool casinoWarTie(int playerNumber) {
/*
 * THIS FUNCTION prompts the player on whether or not they would like to continue
 * returns true to continue
 * returns false to surrender
 */
  char key = NO_KEY;
  Serial.println("casinoWarTie");
  String playerLine = get_line(playerNumber);

  while ((key != '#') && (key != '*')) {

    lcd.setCursor(0,0);
    lcd.print(playerLine);
    lcd.setCursor(0,1);
    lcd.print("#) MATCH BET");
    lcd.setCursor(0,2);
    lcd.print("*) SURRENDER");
    key = keypad.getKey(); // we will be getting values from the keyPad
  }
  if (key == '#') {
    lcd.clear();
    return true;
  }
  else if (key == '*') {
    lcd.clear();
    return false;
  }
}

void casinoWarWinners(int listOfPlayers[], int numPlayers ) {
  /*
   * THIS FUNCTION DISPLAYS A LIST OF WINNING PLAYERS
   */
  char key = NO_KEY;
  Serial.println("casinoWarWinners");
  while (key != '#') {

    lcd.setCursor(0,0);
    lcd.print("WINNING PLAYERS:");
    for ( int i = 0; i < numPlayers; i ++) {
      int playerNum = listOfPlayers[i];
      if (playerNum != 0) {
       lcd.setCursor(2 * i, 1);
        lcd.print(playerNum);
      }
```

```
    }
    lcd.setCursor(0,3);
    lcd.print(" PRESS #");
    key = keypad.getKey(); // we will be getting values from the keyPad
  }
  if (key == '#') {
    lcd.clear();
    return;
  }

}


void casinoWarDealerWins() {
  char key = NO_KEY;
  Serial.println("casinoWarDealerWins");
  while (key != '#') {

    lcd.setCursor(0,0);
    lcd.print(" DEALIN' DIRTY");
    lcd.setCursor(0,1);
    lcd.print(" DID YOU DIRTY!");
    lcd.setCursor(0,3);
    lcd.print(" PRESS #");
    key = keypad.getKey(); // we will be getting values from the keyPad
  }
  if (key == '#') {
    lcd.clear();
    return;
  }

}









//_________________________________Five Card Draw___________________________________

bool five_card_draw(int numPlayers) {
  bool gameFinished = false; // this will terminate the game
  char key = NO_KEY; // we will be getting keys from the keypad
  int betNumber = 0; // this will be used for the different betting periods
  int exchangeNumber = 0; // this will be used for the different exchange periods
  int numCards = 0; // this is used to see how many cards to exchange for each player
//// These are the variables which indicate base motor movement for the deal ////////

  int num = numPlayers - 1;
  int turn = (180 / numPlayers) ;
  float adjustFloat = turn / 2.0;
  int adjust = floor(adjustFloat);
  if (numPlayers ==0) {
    turn = 0;
  }
  int init = 90 - adjust; // this is useful for a dealer starting directly center
  int backToStart = turn * (num) ; // first iteration of the deal
  int backToStart2 = turn * (num + 1); // second iteration of the deal
```

```
///// first need the ante /////

betNumber = 1; // start the first betting period


while ( betNumber ==1) { // wait for the players to finishing betting
  key = keypad.getKey();

  if (key != '#') {
    lcd.setCursor(0,0);
    lcd.print(" ANTE UP! ");
    lcd.setCursor(0,3);
    lcd.print(" # ->");

  }
// escape the while loop
  else {
  betNumber = 0;
  lcd.clear();
  }

}
/////// This is the Deal //////////
  moveBase(false, adjust , baseSpeed ); // move to first player
  dealCard(topSpeed); // Deal a Card close to the player

  for (int j = 0; j<=4 ; j++){
    if (j==0) { // this is the first round
      for (int i=0; i<=num-1; i++){
        moveBase(false, turn , baseSpeed ); // move to the next player
        dealCard(topSpeed); // Deal a Card close to the player
      }
    moveBase( true, backToStart , baseSpeed ); // move to first player
    }

  else{ // this is the second round
    for (int p=0; p<=num-1; p++){
      dealCard(topSpeed);
      moveBase(false, turn, baseSpeed );
      }
    dealCard(topSpeed); // Deal a Card close to the final player

    moveBase(true, backToStart , baseSpeed ); // move to first player
  }
}


  betNumber = 2; // start the second betting period

  while ( betNumber ==2) { // wait for the players to finishing betting
    key = keypad.getKey();

    if (key != '#') {
    lcd.setCursor(0,0);
    lcd.print(" RAISE! FOLD! CALL! ");
    lcd.setCursor(0,3);
    lcd.print(" # ->");

    }
    else { // the players are ready, reset the variables and clear the screen
      lcd.clear();
      key = NO_KEY;
      betNumber = 0;
    }
  }
```

```
  ////////// Exchange cards //////////

  int numDeals = how_many_to_exchange(1); // asking player 1
  for (int i = 0; i<numDeals ; i++){
     dealCard(topSpeed);
  }

  // player one has his cards, escape the while loop
  exchangeNumber = 0;
  numDeals = 0; // reset variable

  // now loop to get the rest of the players
  for (int k = 2; k<=num+1; k++){
      moveBase(false, turn , baseSpeed );
      exchangeNumber = k;
       int numDeals = how_many_to_exchange(k); // asking the other players
       for (int i = 0; i<numDeals ; i++){
          dealCard(topSpeed);
       }


   }


  betNumber = 3; // start the second betting period

  while ( betNumber ==3) { // wait for the players to finishing betting
    key = keypad.getKey();

    if (key != '#') {
    lcd.setCursor(0,0);
    lcd.print(" RAISE! FOLD! CALL! ");
    lcd.setCursor(0,3);
    lcd.print(" # ->");

    }
    else { // the players are ready, reset the variables and clear the screen
      lcd.clear();
      key = NO_KEY;
      betNumber = 0;
    }
  }

// this is the end of the game!
  gameFinished = true;

  return gameFinished;

  }

//________________________________Go Fish___________________________________


bool go_fish(int numPlayers) {

  bool gameFinished = false; // this will terminate the game
  char key = NO_KEY; // we will be getting keys from the keypad
  int numCardsLeft = 52; // this is used to count the number of cards left in the deck
//// These are the variables which indicate base motor movement for the deal ////////

  int num = numPlayers - 1;
  int turn = (180 / numPlayers) ;
  float adjustFloat = turn / 2.0;
```

```
  int adjust = floor(adjustFloat);
  if (numPlayers ==0) {
    turn = 0;
  }
  int init = 90 - adjust; // this is useful for a dealer starting directly center
  int backToStart = turn * (num) ; // first iteration of the deal
  int backToStart2 = turn * (num + 1); // second iteration of the deal
  int numDealtCards;
  int numFish;
// get the number of card to deal each player
  if (numPlayers ==2) {
    numDealtCards = 7;
  }
  else {
    numDealtCards = 5;
  }


/////// This is the Deal //////////
  moveBase(false , adjust , baseSpeed ); // move to first player
  dealCard(topSpeed); // Deal a Card close to the player

  for (int j = 0; j<= numDealtCards -1 ; j++){
    if (j==0) { // this is the first round
      for (int i=0; i<=num-1; i++){
        moveBase(false , turn , baseSpeed ); // move to the next player
        dealCard(topSpeed); // Deal a Card close to the player
      }
    moveBase(true, backToStart , baseSpeed ); // move to first player
    }

  else{ // this is the second round
    for (int p=0; p<=num-1; p++){
      dealCard(topSpeed);
      moveBase(false , turn , baseSpeed );
      }
    dealCard(topSpeed); // Deal a Card close to the final player

    moveBase(true , backToStart , baseSpeed ); // move to first player
  }
}

// subtract cards from the deal
  numCardsLeft = numCardsLeft - (numDealtCards * numPlayers);

// start the gameplay

  while (numCardsLeft >= 1) { // while there are still cards, continuously prompt each playyer

    for (int p = 0 ; p <= num ; p ++) { // need to cycle through all players
        numFish = fishCards(numCardsLeft); // get number of cards
        for ( int k = 0; k <= numFish - 1; k ++ ) { // deal number of cards
          dealCard(topSpeed);
          numCardsLeft = numCardsLeft - 1; // subtract from the card count
          if (numCardsLeft == 0) { // If you run out of cards
            completeGoFish();
            return gameFinished;

          }
        }
        if ( p < num) {
          moveBase(false, turn , baseSpeed );

        }
        else {
```

```
        moveBase(true , backToStart , baseSpeed ); // move back to first player
       }


     }



   }


  gameFinished = true;
  return gameFinished;
}

void completeGoFish() {

  char key;

  while (key != '#') { // while loop waits for players to move forward
    lcd.setCursor(0,0);
    lcd.print(" ENJOY GO FISH ");
    lcd.setCursor(0,1);
    lcd.print(" SELECT # WHEN DONE ");
    lcd.setCursor(0,3);
    lcd.print(" # ->");
    key = keypad.getKey(); // we will be getting values from the keyPad
  }

  if (key == '#') {
    lcd.clear();
    return;
  }

}




//_________________________________Game Play Help Functions___________________________________

/*
 * this is a function specific to go_fish
 * it is designed to retrieve the number of cards a player would like to get
 */

int fishCards(int numCardsLeft){

  int numCards = 0; // set to zero for now
  int shift4 = 0; // this will be used to toggle
  char key = keypad.getKey(); // we will be getting the value from the keypad


while (numCards == 0) { // This loop will get the number of players

    if (shift4 == 0 ) { // prompt the user
    lcd.setCursor(0,0);
    lcd.print(" Go Fish! ");
    lcd.setCursor(0,1);
    lcd.print(" 1) Go Fish ");
    lcd.setCursor(0,2);
    lcd.print(" 2) New Hand ");
    key = keypad.getKey(); // we will be getting the value from the keypad
      if (key == '#' || key == '*' || key == '0' || key == '9' || key == '8' || key == '7'|| key == '3' || key
```

```
          ↪ == '4' || key == '5' || key == '6') { // these are all the situations where we return an error
          ↪ message
        shift4 = 1;
    }


      if (key == '1') {
        shift4 = 2;
      }
      if (key == '2') {
        shift4 = 3;
      }
    }



    else if (shift4 == 1) { // these are all the situations where we return an error message
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print(" Please Select ");
      lcd.setCursor(0,1);
      lcd.print(" 1 or 2! ");
      delay(1000);
      lcd.clear();
      key = reset;
      shift4 =0;
    }
    else if (shift4 == 2) { // return the number of players as an integer
      lcd.clear();
      numCards = 1;

    }
    else if (shift4 == 3) { // return the number of players as an integer
      if (numCardsLeft <= 5) {
        numCards = numCardsLeft;
      }
      else{
        numCards =5;
      }


    }
}
lcd.clear();
return numCards;

}




String get_line(int playerNum){
/*
 * this function is used because concatenation of strings in C++ is not ideal
 */
  String playerLine;
  if (playerNum == 1){
    playerLine = "PLAYER 1: ";
  }
  else if (playerNum ==2) {
    playerLine = "PLAYER 2: ";
  }
  else if (playerNum == 3){
    playerLine = "PLAYER 3: ";
  }
```

```
  else if (playerNum ==4) {
    playerLine = "PLAYER 4: ";
  }
  else if (playerNum == 5){
    playerLine = "PLAYER 5: ";
  }
  else if (playerNum ==6) {
    playerLine = "PLAYER 6: ";
  }
  return playerLine;
}


void bust_display(int player) {
/*
 */

    String playerLine = get_line(player);

     lcd.setCursor(0,0);
     lcd.print(playerLine);
     lcd.setCursor(0,2);
     lcd.print(" BUST! ");

     delay(500);
     lcd.clear();

     lcd.setCursor(0,0);
     lcd.print(playerLine);
     lcd.setCursor(0,2);
     lcd.print(" BUST! ");

     delay(500);
     lcd.clear();


     lcd.setCursor(0,0);
     lcd.print(playerLine);
     lcd.setCursor(0,2);
     lcd.print(" BUST! ");

     delay(1000);
     lcd.clear();
}


void black_jack_display(int player) {
/*
 */

    String playerLine = get_line(player);

     lcd.setCursor(0,0);
     lcd.print(playerLine);
     lcd.setCursor(0,2);
     lcd.print(" BLACK JACK! ");

     delay(500);
     lcd.clear();

     lcd.setCursor(0,0);
     lcd.print(playerLine);
     lcd.setCursor(0,2);
     lcd.print(" BLACK JACK! ");
```

```
     delay(500);
     lcd.clear();


     lcd.setCursor(0,0);
     lcd.print(playerLine);
     lcd.setCursor(0,2);
     lcd.print(" BLACK JACK! ");

     delay(1000);
     lcd.clear();
}



int how_many_to_exchange(int playerNum) {
/*
 * this is a function specific to five_card_draw
 * it is designed to retrieve the number of cards a player would like to get
 */
  int numCards = 0; // set to zero for now
  int shift3 = 0; // flip screens
  String playerLine = get_line(playerNum);
  char key = keypad.getKey(); // we will be getting the value from the keypad
  while (numCards == 0) { // This loop will get the number of players

     if (shift3 == 0 ) { // prompt the user
     lcd.setCursor(0,0);
     lcd.print(playerLine);
     lcd.setCursor(0,1);
     lcd.print(" SELECT NUMBER OF ");
     lcd.setCursor(0,2);
     lcd.print(" NEW CARDS ");
     key = keypad.getKey();

       if (key == '#' || key == '*' || key == '6' || key == '9' || key == '8' || key == '7' || key == '4' ||
            ↪ key == '5') { // these are all the situations where we return an error message
         shift3 = 1;
     }


       else if (key == '1' || key == '2' || key == '3' || key == '0') {
         shift3 = 2;
       }
     }



     else if (shift3 == 1) { // these are all the situations where we return an error message
       lcd.clear();
       lcd.setCursor(0,0);
       lcd.print(" SELECT A NUMBER ");
       lcd.setCursor(0,1);
       lcd.print(" BETWEEN 0 AND 3! ");
       delay(1000);
       lcd.clear();
       key = reset;
       shift3 = 0;
     }
     else if (shift3 == 2){ // return the number of players as an integer
       String number = String(key);
       lcd.clear();
       numCards = number.toInt();

     }
```

```
  }
  Serial.println(numCards);
  return numCards;
}



//_____________________________________________Motor Control
    ↪ Functions_____________________________________________________________


bool testStart() {
  char key = keypad.getKey();

  while ((key != '*') && (key != '#')) {

    lcd.setCursor(0,0);
    lcd.print(" Start Test ");
    lcd.setCursor(0,3);
    lcd.print("<- * # ->");
    key = keypad.getKey();


  }

  if (key == '#') {
    lcd.clear();
    return true;


}
  else if (key == '*') {
    lcd.clear();
    return false;


}

}

bool testStart2() {
  char key = keypad.getKey();

  while ((key != '*') && (key != '#')) {

    lcd.setCursor(0,0);
    lcd.print(" Move Base ");
    lcd.setCursor(0,3);
    lcd.print("<- * # ->");
    key = keypad.getKey();

  }

  if (key == '#') {
    lcd.clear();
    return true;


}
  else if (key == '*') {
    lcd.clear();
    return false;


}
```

```
}



void failed_card() {
    char key = NO_KEY;

  while (key != '#') {

    lcd.setCursor(0,0);
    lcd.print(" PLEASE RESET CARD ");
    lcd.setCursor(0,1);
    lcd.print(" ON TOP OF THE SLIDE ");
    lcd.setCursor(0,3);
    lcd.print("<- * # ->");
    key = keypad.getKey();

  }

  if (key == '#') {
    lcd.clear();
    return true;


}

}


String dealCard(int pace) {
/*
 * This function deals a card by combining some of the below motor control functions.
 * It will return the value of the Card dealt as a String
 */
 pickCard(false);

 String val = getCardVal();
 Serial.println(val);
 while (val == "false") {
  failed_card();
  val = getCardVal();
  Serial.println(val);
 }

 shoot_card(pace);

 return val;
//
//pickCard(false);
//delay(1000);
//shoot_card(pace);

}

void shoot_card(int pace) {
/*
 * This functions operates the dealing functionality
 * The DC motor is turned on, then the push stepper feeds the card into the DC shooter motor
 * The motor is then released
 */

// turn on the Dc motor
   digitalWrite(in3 , LOW);
```

```
   digitalWrite(in4 , HIGH);
// set the speed of the DC motor
  analogWrite(enB , pace);
// push the card into the DC motor
  pushCard(false);
// delay for one half Second
  delay(500);
// turn off the motor
  analogWrite(enB , 0); // this should brake the motor
  digitalWrite(in3 , LOW); // this will kill the power
  digitalWrite(in4 , LOW); // kill the power
}

void pushCard( bool dir) {
/*
 * This function runs the pusher motor in a 360 degree rotation.
 * The selected motor has 200 Steps per revolution
 */

// Case 1, turn the motor one direction
if (dir ==true) {
  digitalWrite(dirPinPush,HIGH);
}
// case 2, turn the motor in the other direction
else {
  digitalWrite(dirPinPush,LOW);
}

// Run: move the motor in a full rotation
  for(int x = 0; x < numStepsPick; x++) {
    digitalWrite(stepPinPush,HIGH);
    delayMicroseconds(1250);
    digitalWrite(stepPinPush,LOW);
    delayMicroseconds(1250);
  }

}


void moveBase(bool dir, int angle , int pace) {
/*
 * This function moves the base an indicated direction, speed, and number of degrees
 * Each movement is incrimental. i.e. the base hase no positional awareness.
 * dir (boolean): true indicates a counter clockwise movement. false indicates clockwise
 * angle (integer): angular adjustment, indicated in degrees
 * pace (int): speed of the motor, range: 0 - 255
 */

// Calculation: determines the number of encoder pulses, provided the given angular adjustment
 float numPulses = (angle / 360.0) * pulsesPerRev;
 int numPulsesInt = floor(numPulses);

// Case 1: Counter Clockwise
if (dir == true) {
  digitalWrite(in1 , HIGH);
  digitalWrite(in2 , LOW);
}
// case 2: Clockwise
else {
  digitalWrite(in1 , LOW);
  digitalWrite(in2 , HIGH);
}
```

```
// move: Power the motor while the counter reads less than the number of encoder pulses
while (counter < numPulsesInt) {
 analogWrite(enA , pace);
 if (digitalRead(endStopBot) == 0) {
  counter = numPulsesInt;
  // MAYBE A BASE HOME CODE IMPLIMENTED HERE
  break;
 }
}

// Brake:
analogWrite(enA , 0); // this should brake the motor
digitalWrite(in1 , LOW); // this will kill the power
digitalWrite(in2 , LOW); // kill the power


delay(500); // short delay to allow motor to stop before resetting the counter
counter = 0;

}



void pickCard( bool dir) {
/*
 * This function runs the picker motor in a 360 degree rotation.
 * The selected motor has 200 Steps per revolution
 */

// Case 1, turn the motor one direction
if (dir ==true) {
  digitalWrite(dirPinPick,HIGH);
}
// case 2, turn the motor in the other direction
else {
  digitalWrite(dirPinPick,LOW);
}

// Run: move the motor in a full rotation
  for(int x = 0; x < numStepsPick; x++) {
    digitalWrite(stepPinPick,HIGH);
    delayMicroseconds(900);
    digitalWrite(stepPinPick,LOW);
    delayMicroseconds(900);
  }

}


void slideTop(bool dir) {

  if (dir == true) {
    digitalWrite(dirPinSlide,HIGH);
  }
  else {
    digitalWrite(dirPinSlide,LOW);
  }


  float circ = 2 * M_PI * rollerRadius;
  float howFar = (distance / circ) * numStepsSlide;
  int rotateStepper= floor(howFar); // it needs to be an integer, use the floor function

  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < rotateStepper; x++) {
```

```
    digitalWrite(stepPinSlide,HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPinSlide,LOW);
    delayMicroseconds(1000);
  }
}

void homing(){
/*
 * A function used to home the Dealer
 * It makes use of two endStops: It first homes the top of the dealer, then homes the base
 * The LCD Board prompts the User to wait as it homes the device
 * The EndStops function as Normally Closed (read High while not engaged)
 */

// Interface: Display Message
  digitalWrite(dirPinSlide,HIGH);
    lcd.setCursor(0,0);
    lcd.print(" Homing Your ");
    lcd.setCursor(0,1);
    lcd.print(" Device ");
    char key = keypad.getKey();

// home the top of the device
  while (digitalRead(endStopTop) == 1) {

    digitalWrite(stepPinSlide,HIGH);
    delayMicroseconds(4000);
    digitalWrite(stepPinSlide,LOW);
    delayMicroseconds(4000);
  }

  digitalWrite(dirPinSlide,LOW);
  for(int x = 0; x < 15; x++) {
    digitalWrite(stepPinSlide,HIGH);
    delayMicroseconds(8000);
    digitalWrite(stepPinSlide,LOW);
    delayMicroseconds(8000);

}



// home the bottom

  digitalWrite(in1 , HIGH);
  digitalWrite(in2 , LOW);

  while (digitalRead(endStopBot) == 1) {
    analogWrite(enA , homeSpeed);

  }

  analogWrite(enA , 0); // this should brake the motor
  digitalWrite(in1 , LOW); // this will kill the power
  digitalWrite(in2 , LOW); // kill the power

  digitalWrite(in1 , LOW);
  digitalWrite(in2 , HIGH);
  while (digitalRead(endStopBot) == 0) {
  analogWrite(enA , homeSpeed);
}
  analogWrite(enA , 0); // this should brake the motor
  digitalWrite(in1 , LOW); // this will kill the power
  digitalWrite(in2 , LOW); // kill the power
```

```
  delay(500);
  counter = 0;
  moveBase(false, 7 , homeSpeed);
  lcd.clear();

}



//_____________________________________________RFID
     ↪ function_____________________________________________________________________
// encorporate this into the dealCard Function. should return an int.


String getCardVal() {
/*
 * This function reads an RFID Tag Which is placed on a Card
 * The cards are preWritten using MFRC552 Library Example Code "rfid_write_personal_data".
 * This is modified from the MFRC552 Library Example Code "rfid_read_personal_data"
 * Each has the Card Value as "first name" and Suit as "last Name"
 * The Card and Suit Values are represented as Bytes. This code converts them to a string and returns it
 * using while loops, the program waits until a valid card is read, then returns the card value as a String.
 */

// Indication: Variables used in the program
  bool foundCard = false;
  bool test = false;
  String cardVal;
  String newCard;
  String lastName = "";


  while (foundCard ==false) {
    while (test == false) {
    // Prepare key - all keys are set to FFFFFFFFFFFFh at chip delivery from the factory.
  MFRC522::MIFARE_Key key;
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;

  //some variables we need
  byte block;
  byte len;
  MFRC522::StatusCode status;

  // Look for new cards
  if ( ! mfrc522.PICC_IsNewCardPresent()) {
continue;
    }

  // Select one of the cards
  if ( ! mfrc522.PICC_ReadCardSerial()) {
    continue;
  }

// Useful Variables
  byte buffer1[18];
  block = 4;
  len = 18;

  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, 4, &key, &(mfrc522.uid)); //line 834 of
       ↪ MFRC522.cpp file
  if (status != MFRC522::STATUS_OK) {
   return "false";
  }

  status = mfrc522.MIFARE_Read(block, buffer1, &len);
```

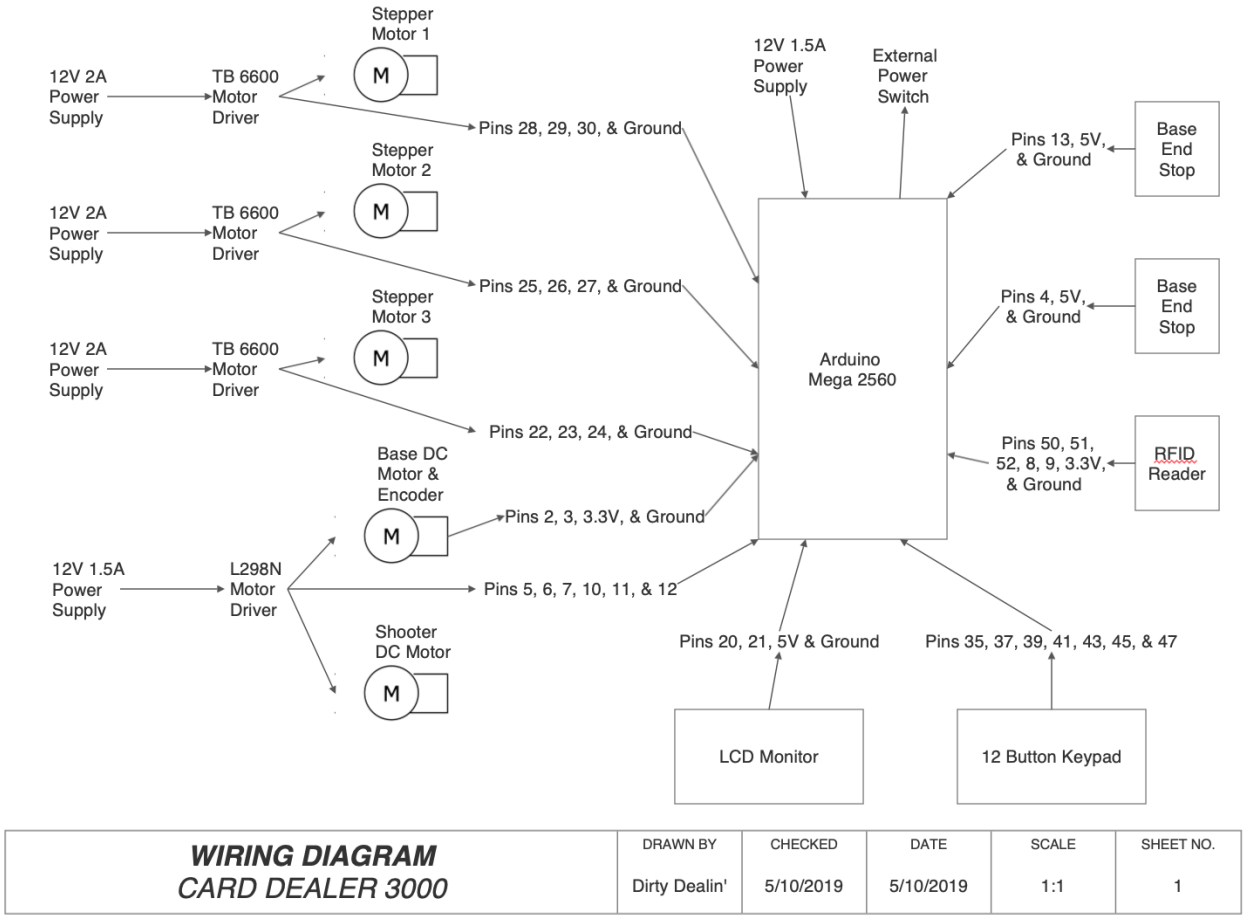
```
  if (status != MFRC522::STATUS_OK) {
    return "false";
  }

// Retreive: This Converts the Bytes which represent Card Value. Converts to a string
  String cardVal = "";
  for (uint8_t i = 0; i < 16; i++)
  {
    if (buffer1[i] != 32)
    {
      char card = (char) buffer1[i];
      if (card !=""); {
        cardVal += card;
      }
    }
  }
  newCard = cardVal;

  byte buffer2[18];
  block = 1;

  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, 1, &key, &(mfrc522.uid)); //line 834
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Authentication failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return "false";

  }

  status = mfrc522.MIFARE_Read(block, buffer2, &len);
  if (status != MFRC522::STATUS_OK) {
    Serial.print(F("Reading failed: "));
    Serial.println(mfrc522.GetStatusCodeName(status));
    return "false";

  }

// This Gets the suit of the Card
  String cardSuit = "";
  for (uint8_t i = 0; i < 16; i++) {

    char val = (char) buffer2[i];
    cardSuit += val;
  }

  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();

// Confirm: if a card has been found, exit the while loops

    if (cardSuit.substring(1, 6) == "SPADE" || cardSuit.substring(1, 6) == "HEART" || cardSuit.substring(1, 5)
        ↪  == "CLUB" || cardSuit.substring(1,8) == "DIAMOND" ) {
      foundCard = true;

      test = true;

    }

    mfrc522.PICC_HaltA();
    mfrc522.PCD_StopCrypto1();
  }
  }
// Return: the card value as a string
   return newCard;
 }
```

```
int casinoWarGetCard(String card){
/*
 * Convert the cardValue recorded by the RFID module into integer card values for Casino War Gameplay
 * integers will allow for < , > , = , comparison which is useful during gameplay
 * Ace is high
 */

// initialize: CardValue as an integer
  int cardValue;

// indicate: Logic which sets integer values to the card characters
  if (card.charAt(1) == 'J') {
    cardValue = 11;
  }
  else if (card.charAt(1) == 'Q') {
    cardValue = 12;
  }
  else if (card.charAt(1) == 'K') {
    cardValue = 13;
  }
  else if (card.charAt(1) == 'A') {
    cardValue = 14;
  }
  else if (card.substring(1, 2) == "10") {
    cardValue = 10;

  }
  else {
    cardValue = card.toInt();
  }
// Return:
  return cardValue;
}

int blackJackGetCard(String card ){
/*
 * Convert the cardValue recorded by the RFID module into integer card values for Casino War Gameplay
 * integers will allow for < , > , = , comparison which is useful during gameplay
 * Ace is considered high until the player reaches a value above 21
 */

 // initialize: cardValue as an integer
  int cardValue;

// logic: set each card Value
  if (card.charAt(1) == 'J' || card.charAt(1) == 'Q' || card.charAt(1) == 'K' || card.substring(1, 2) == "10")
      ↪  {
    cardValue = 10;
  }
  else if (card.charAt(1) == 'A') {
    cardValue = 11;
  }

  else {
    cardValue = card.toInt();
  }

// Return:
  return cardValue;
}
```

```
//_____________________________________________interupt
    ↪ function_____________________________________________________________________

void pin_A() {
  counter ++;
}
```

## 7.3 Appendix C: Wiring Schematic

This is a wiring schematic of the Card Dealer 3000. It contains, 5 power supplies, 4 motor drivers, 1 Arduino Mega 2560, 3 stepper motors, 2 DC motors, 2 end stops, 1 RFID reader, 1 12-button keypad, and 1 LCD screen.




| WIRING DIAGRAM<br>CARD DEALER 3000 | DRAWN BY | CHECKED | DATE | SCALE | SHEET NO. |
|---|---|---|---|---|---|
| | Dirty Dealin' | 5/10/2019 | 5/10/2019 | 1:1 | 1 |