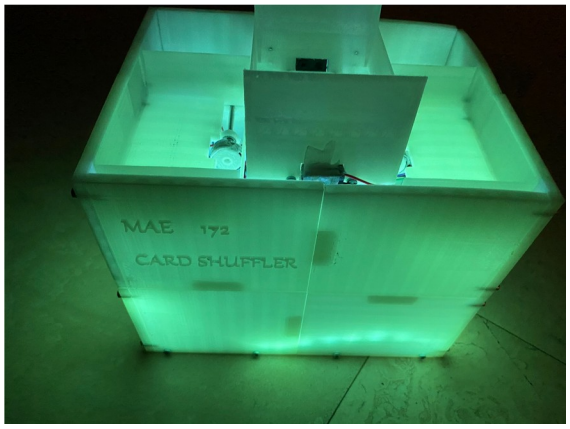
MAE 172
Final Report
Card Shuffler & Card Dealer



&

Mingqing Yuan
Xinyi Zhong
Jiexuan He
Neo Meng

1.Motivation

a. Many games need card shuffling are popular

Card games, such as BlackJack and Texas Hold' em, are super popular for parties or family. Hundreds of times of shuffling are required for one night games. Automatic card shuffler and dealer can greatly help with people with card shuffling.

b. Cards shuffled by hand are easy to be curved

According to comments on Amazon, a deck of average quality playing cards will be curved after about 100 times of shuffling by hand by average.

c. No product with acceptable price for family market

No card shuffler under $100 can shuffle cards many times automatically in market

d. Card dealing by hand takes a lot of time and makes errors.

Dealing cards by hand will take a lot of time, and when the people who are dealing the cards are talking at the same time, it will be easy to forget where the card should go next.

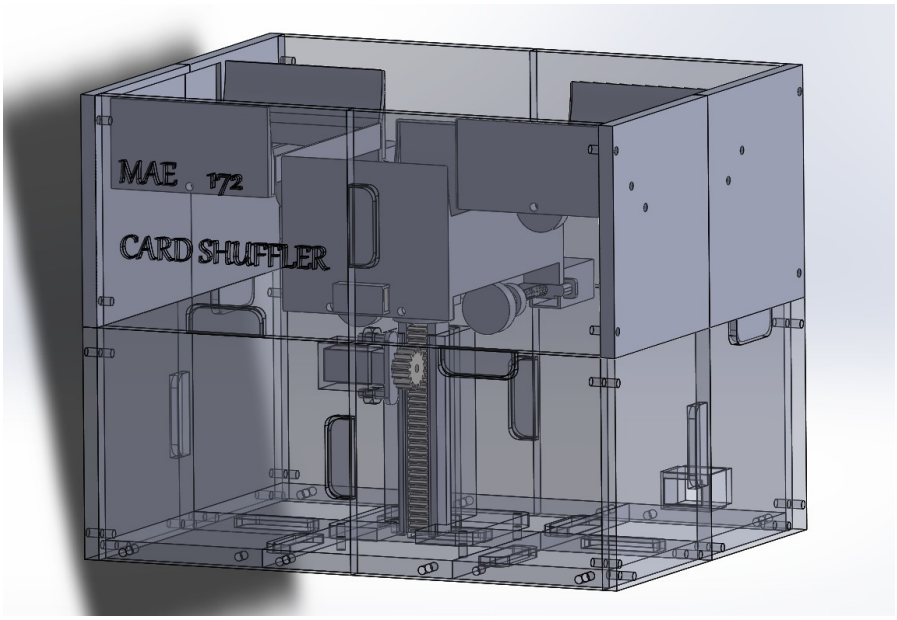
2.Design

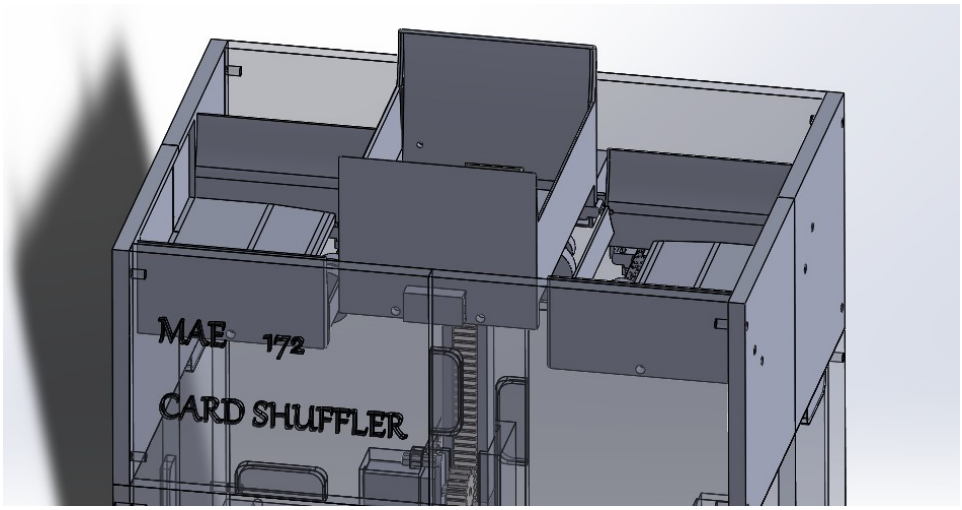
a. Card shuffler

I. Description

The mechanism of this design is very straightforward. In the beginning, the user turns on the machine then the rack and pinion elevator will lift the poker case approximately 55mm up. Next, the user put one deck of poker into the case. While IR sensors detect cards, two rollers connected to motors on the poker case begin to roll in opposite directions until after the IR sensor detects there are no cards in the poker case continuously 12 times. After distributing cards onto the left and right platform, the elevator goes down and motors beneath two platforms start working and drive cards back to the poker case. When all cards are dispensed back to the poker case, the elevator rises to the highest point and shakes 0.5 seconds to the cards to make them into a deck so the user can easily remove cards. When all cards are removed and IR sensors detect nothing, the elevator goes down to the original position. One complete shuffling process takes approximately 50 seconds. Moreover, we have a switch for the purple LED lights, which can make the shuffler look cool at night and users can play cards in the dark.

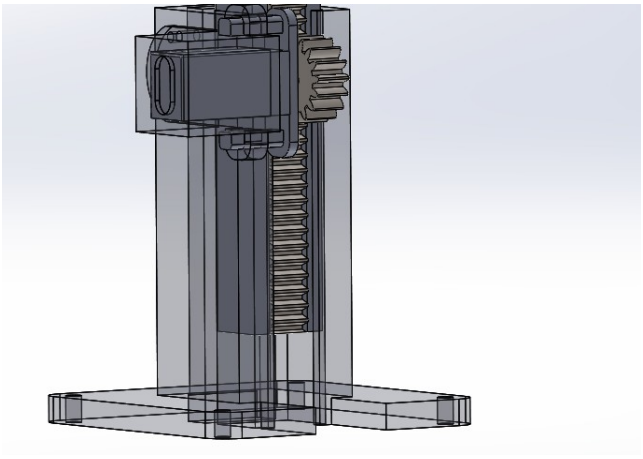
Ii. Overall CAD Model





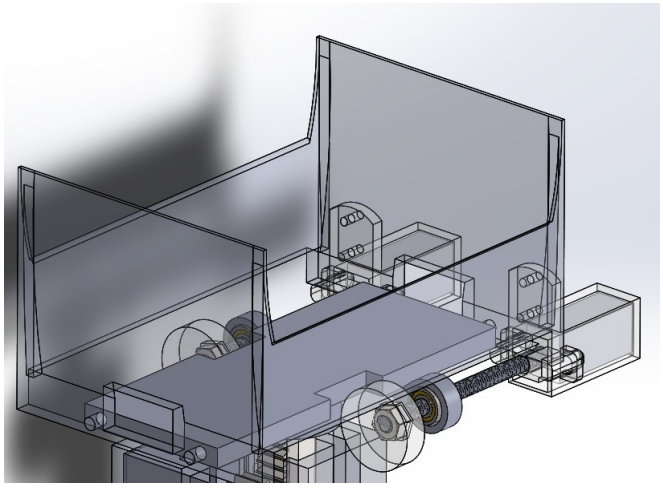
III. Detailed Design

1. Rack and pinion elevator



The rack has a length of 65mm and the elevator can move up and down 55mm. The major challenge of this design is to figure out a way to ensure the elevator lifts and lowers straightly without obliques front or back. The rack enclosure is the key to achieve the desired result. The distance between enclosure and rack should be adequate since it determines the friction.

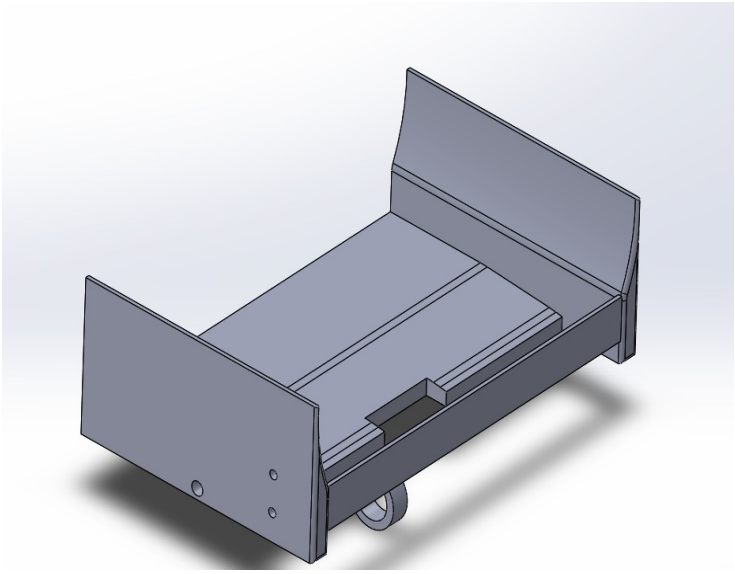
2. Poker case



Four walls of the poker case have small degree deflection towards the bottom since this design prevents the card from clogging. The front and back walls are taller than side

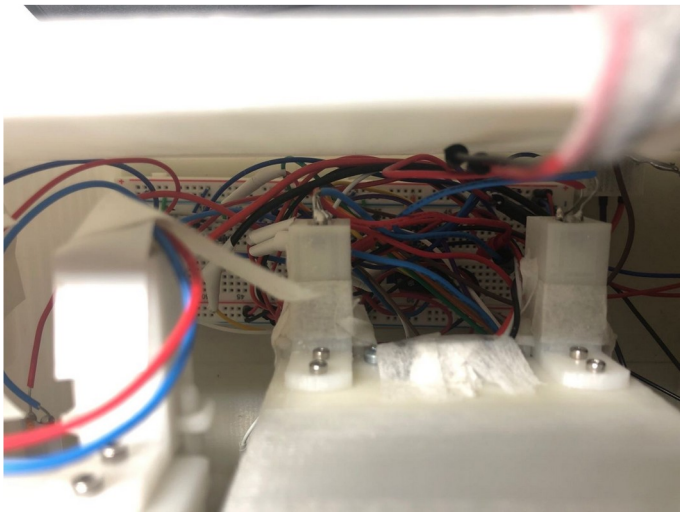
walls so cards will not clog when dispensing from left/right platform. Below the bottom surface, there are two holes for bearings so the motor shafts will not oscillate up and down unstably. Also, there are two small rectangular holes on the front and back walls for placing IR sensors.

3. Left and right platform



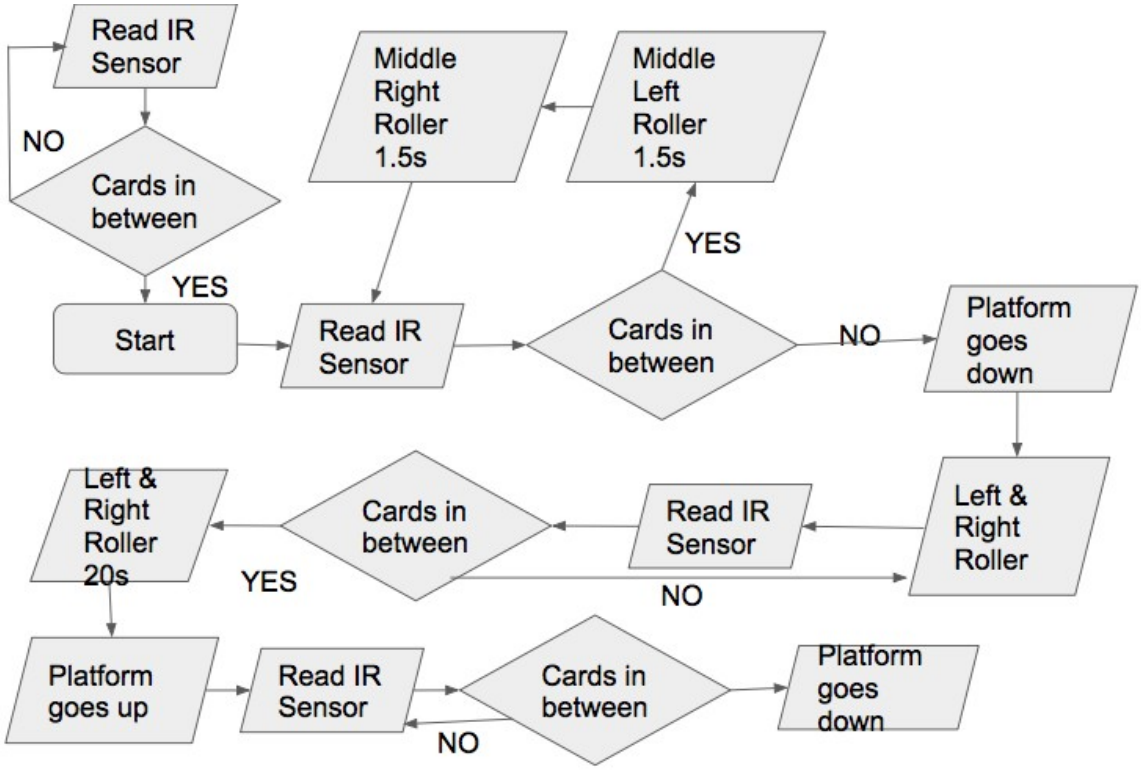
Similar to the poker case, the front and back walls are taller and have small inner deflection to prevent the car from strucking. The bottom surface inclines 10 degrees down so cards can be dispensed easily..

IV. Circuit



This circuit contains a 9V battery, an arduino uno, two switches, 18 purple LEDs, 5 motors, an IR Sensor, a motor driver and several resistors,transistors and diodes. One switch controls the whole shuffler and the other one only controls the LED light.

V. Program Flow Chart



b. Design of the Integrated Dealer Machine:

I. Overall CAD Model:

ii. How it operates:

1. Put the cards in the center stand.
2. Select mode with the two buttons.
3. Cards will be passed to each drawer randomly:
    a. The rollers will pass the cards to either of the two directions
    b. The card will drop into one of the two drawers at the bottom, a small piece of plastic can determine which drawer the card goes.
4. When finished, players can pick up their hands of cards from the drawers

iii.Design Details and Processes:

Structural design:

The structure of the machine is primarily made from laser-cut plywood(maple hard plywood, 0.2 inch thickness), and 3D printed parts(luminous PLA, white). The plywood parts serve as the skeleton, and the 3D printed parts serve as joints. Many electronic parts(motors, solenoids and sensors) are fixed directly onto the 3D printed parts.

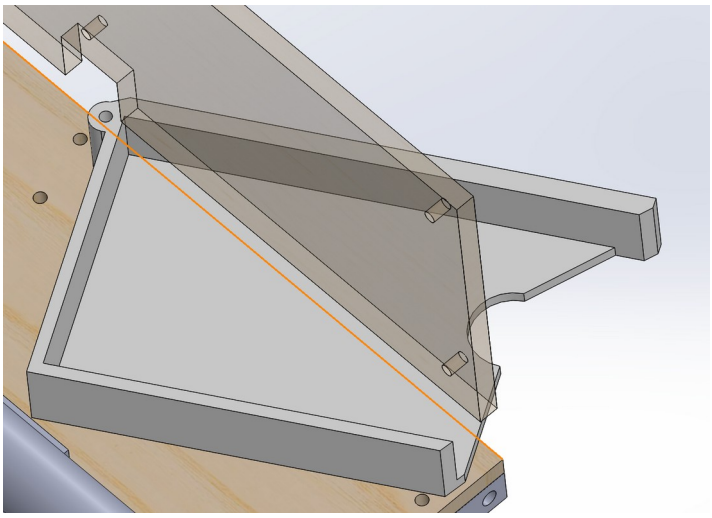
To mount the 3D printed parts and plywood parts together, we used 8mm long M3-0.5 stainless steel flat head screws and M3 thread brass heat-set insert for plastic. We used soldering iron at 220 degree celsius to gently press the insert into the holes on the 3D printed parts. Holes on the 3D printed parts are printed with 4.1mm diameter, because they would shrink to around 4mm(measured). We also used a countersink bit to add countersinks to the holes on plywood parts, so when we put the flat head screw on, the surface is flat.

Bottom face:

There are 4 3D printed parts that connect the side walls and the plastic parts also act as feet for the machine.

Lower layer:

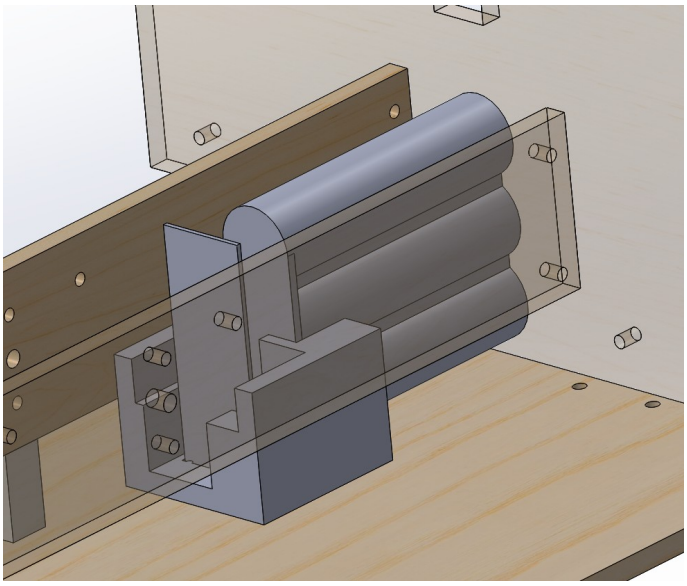
Drawers:



There are 4 drawers that would hold the cards as the dealt hands of cards for each player. The drawers are fixed to the machine by 3mm diameter steel shafts.We use drawers rather than an open port so players cannot see the dealing progress, so they cannot cheat. We chose rotation over linear movement for the drawers, because rotation is simple and results in much less friction. Besides, if the drawers are linearly pulled out, they may get stuck, unless we use rails, which would be too complicated

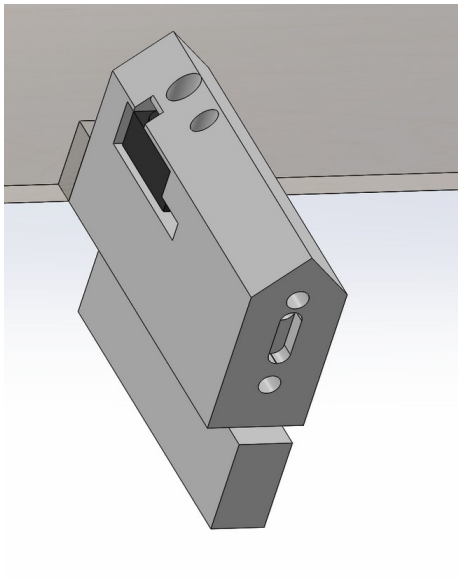
.

Power supply:



We used the power bank from MAE 106 course as our power supply. It has a 12V DC in and out port, and a 5V usb out port with an on-off switch. The original power bank is too big to fit anywhere, so we took its case apart and designed a stand to hold its circuit board and its battery pack. The stand also restricts the battery from moving.

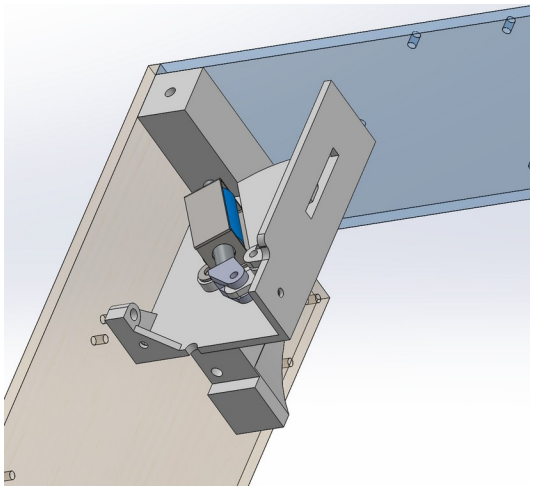
Card dividers & corner stands:



The card dividers are at the middle of the machine, that separates the cards to either of their adjacent drawers. The receivers of the infrared ray sensors are mounted inside this part. The wires go through the part to the central channel.

Each of the corner stands connect 5 faces(front/rear, outer and inner left/right, top and bottom plywood plates) of the machine. They ensure the rigidity of the machine, to a point that we believe it will be intact even if we accidentally drop it to the ground(which we have not tested).

Upper Layer:



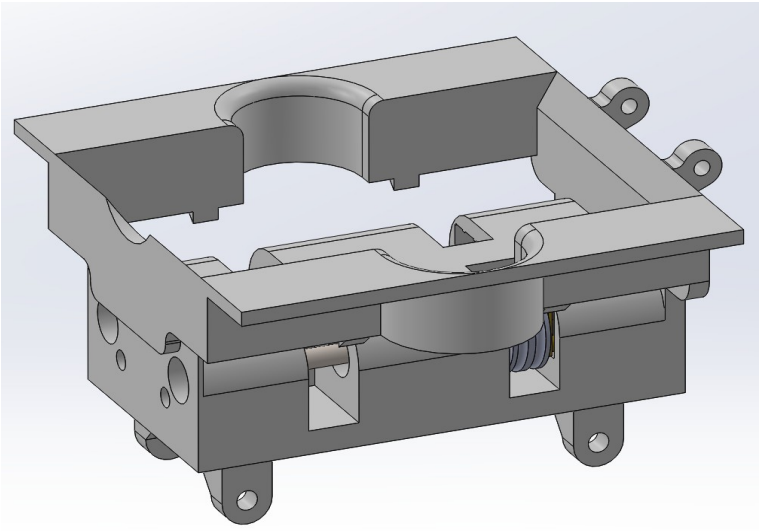
Card diverter:

This part of the machine helps deivert the cards to either drawers. Since the card dispenser is off centered, cards tend to fall to the drawers on one side. However, when a card blocker(a small plate) pops out, the cards will fall to the drawer on the other side of the divider. The small plates are controlled by two small push-pull solenoids.

As the idea of blocking cards to choose sides came out, we were considering using just a small pin to block the cards. As we gradually built our machine on SolidWorks, we noticed that small push-pull solenoids only have 4.5mm of stroke, and bigger ones draw over 1A of current at 12V. So we need to make a lever of some kind to extend the 4.5mm of stroke to around 20mm so cards will definitely touch it. And to avoid cards sliding under the blocker(it was a small pin), we made the blocker much wider. We could have tested our machine with the small pin to block the cards, but if it fails, we would reprint several parts, detach the wire, and even may change the

solenoid, so we decided to prepare for the worst on this design. The result of this is the diverter works perfectly at choosing the path for each card.

Card dispenser:



This part is the most complicated 3D printed part in this machine. It houses the cards to be dealt, and makes sure only one card goes out each time. It also houses two motors, each of which drives two roller wheels to dispense the cards.

Each motor needs 12V DC, has 0.5kg cm of rated torque and 400 rpm of rated speed. The wheel has a diameter of 20mm, so the max force from the motor can be 0.5kg cm/1cm*10N/kg = 5N The poker card we had is very slippery, so slippery that we cannot find a way to measure it, but the motor torque is quite capable of getting these cards out.

The wheels each have two grooves on them to fit two rubber o’rings to increase friction between the wheels and cards so the wheels can grab the cards more easily. The wheels have OD of 18mm, and the o’rings has OD of 20mm, ID of 16mm, thickness of 2mm. However, the o’ring would be dirty after a period of use and the friction would decrease.

For the rollers and motors, we figured that we only need 3 restrains:

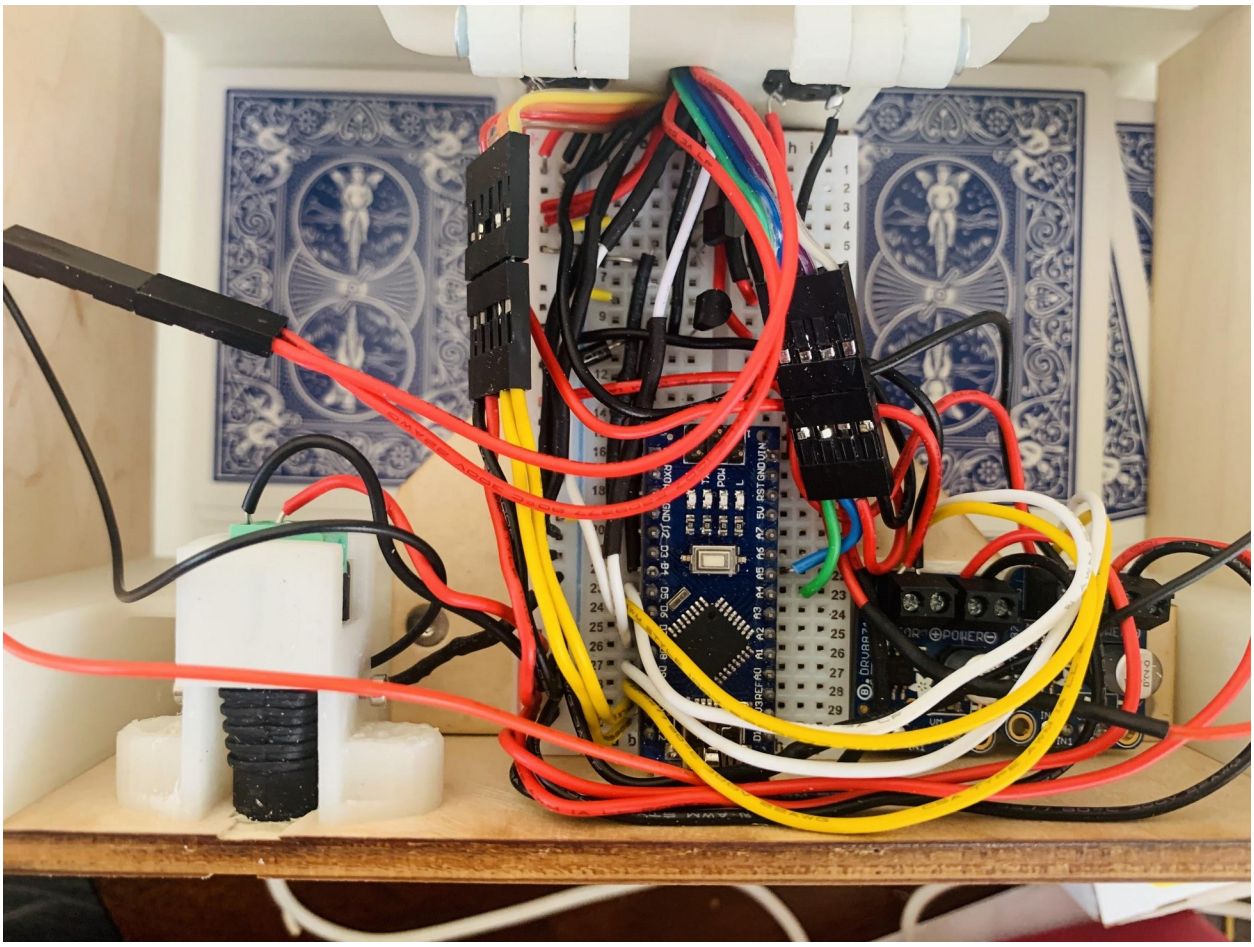
1. We first design the holes for the shaft of the motor, along with bearings to reduce friction. By restricting the shaft to an axis, the motors and rollers now can only slide back and forth and rotate around this axis.
2. Then we fixed the wheels onto the shaft by screwing it on, since the shaft has M4 thread. In this way, the wheels clamping the shaft restrict the shaft from sliding back and forth.

3. Finally, the motor and the wheels can only rotate about the axis, and we cut a square to tightly fit the motor inside, so the whole thing is properly restrained.

To make sure this part only dispenses one card at a time, we put two teeth on each side of the exit wall. When the cards are rolled out, the upper cards will touch the teeth and be blocked, only the lowest one card will go out.

We printed and modified this part 3 times till it finally worked as we expected.

Circuit:



The circuit connecting all the electronic parts together. And the wiring is pretty complicated because we have many parts that need to connect with the arduino and the battery. We tried to use a 9V battery to power the whole circuit, however, the 9V battery is not enough for the push-pull solenoid. So we change to the battery bank from

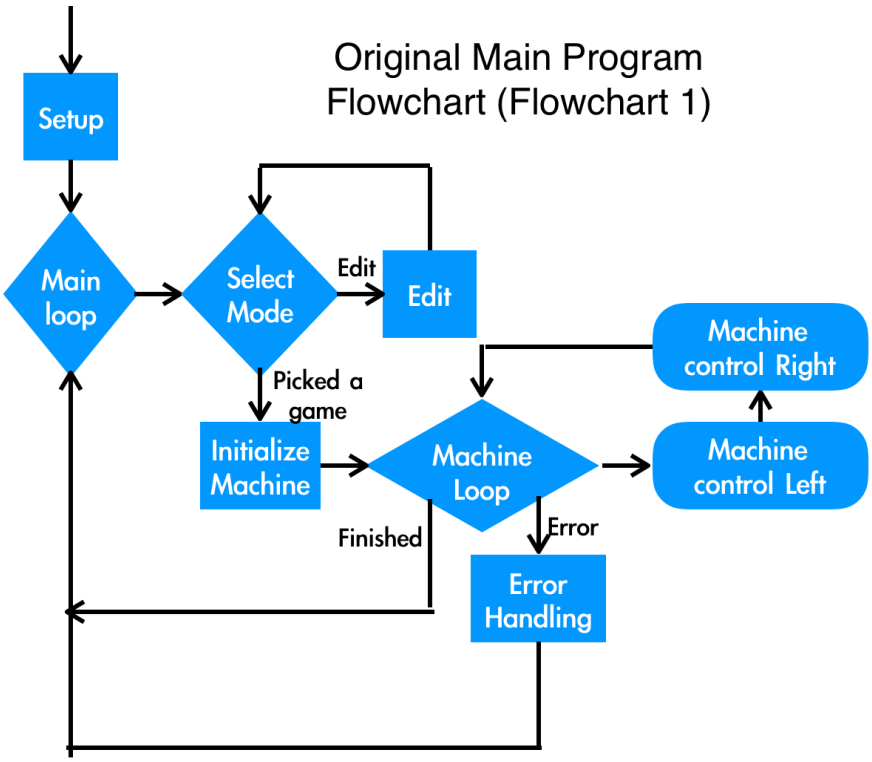
MAE106, which can provide both 5V and 12V from 2 different ports. Solenoid and motors are using the 12V, and sensors, buttons, and small parts are using the 5V. For the wiring diagram, we tried ourselves and also searched online, but there is not exactly the same stuff as we used, so we burned 2 arduinos and finally found the correct way to wiring them. On the circuit for the solenoid, we need to use a 10K resistor and a 2N2222 transistor in order to make it work. We tried different types of transistors and found out the 2N2222 is the best one, some of the transistors will burn during the process and some of them are not working in our circuit. For the wires, we soldered wires from the parts onto a stronger wire so it won't break or loose on the breadboard during the use. And we also made a port on the sidewall the recharge of the power bank.

Iv.programming flowchart:

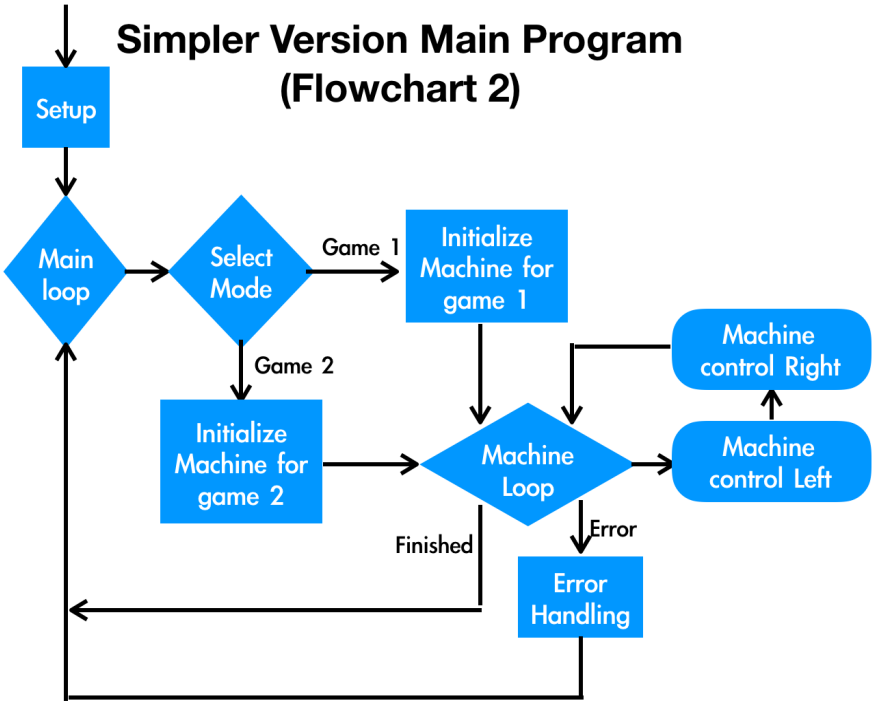
Originally, our program intended to use the screen to provide users lots of information(flowchart 1), but the string variables and display modules took too much space of the arduino memory(for variables), and it kept restarting. We did not noticed this until we put Serial.println("something"); into the setup() function, and only get repeated message-"something" from the serial monitor, which means the arduino kept restarting itself. The solution to this is to write a simpler version: only display a few words, As shown in the flowchart 2.

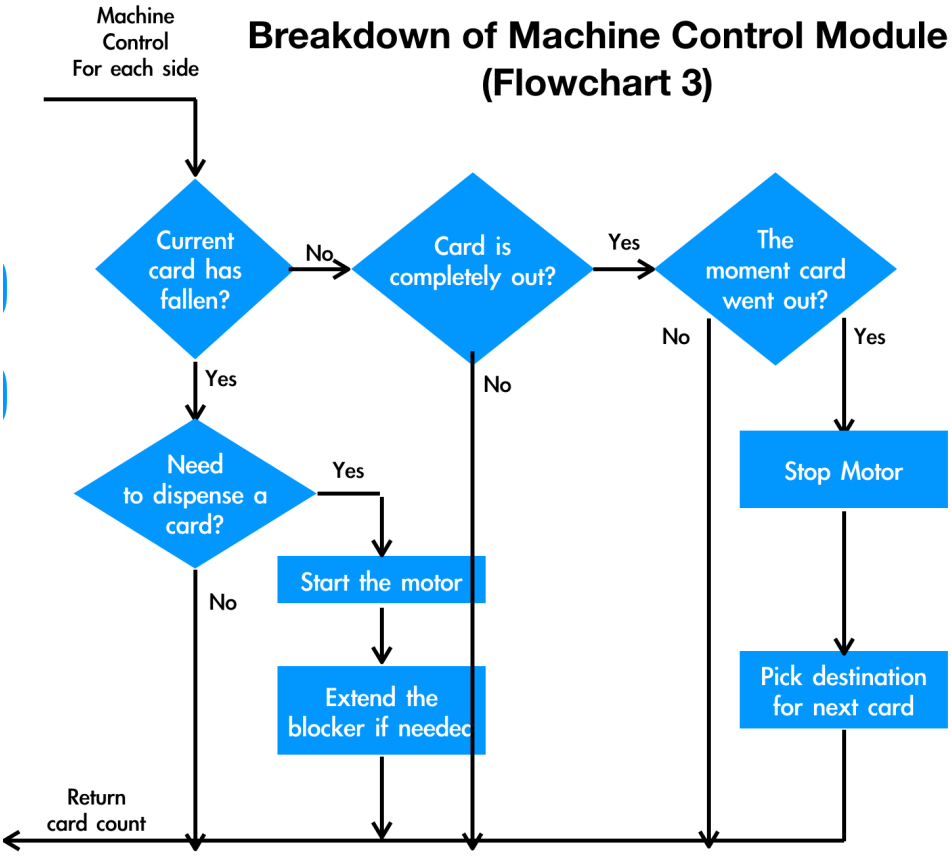
In both Main program flowchart(Flowchart 1 & Flowchart 2), there are two block in round-cornered square box named Machine control Right and Machine control Left, both of which are expanded in detail in Flowchart 3.

Original Main Program
Flowchart (Flowchart 1)



Simpler Version Main Program
(Flowchart 2)





3. Analysis, Simulation and Experiments

a.Card shuffler

Simulation rack and pinion elevator

1. **Material**: PLA (3D Printing)
2. **Mass properties(Solidworks) for each components:**

   Poker case: 46.44g
   Roller:1.95g*2=3.9g
   Motor Mounting case: 3.17g*2=6.34g
   Rack:6.27g
   Rack case: 13.34g
   Bearing: 1.4g*2=2.8g

3. **Specification for motors:**

   Motor 1 (Rack and pinion):
   Voltage: 6V
   Weight: 9g
   Torque: 1.5kg*cm

Current:120mA
No-load Speed: 50 RPM

Motor 2 (Poker case):
Voltage: 12V
Weight: 18g
Torque: 0.5 kg*cm
Current: 170mA
No-load Speed: 400 RPM

4. Calculation:

1 Deck of Poker: Approximately 100g

Total lifting weight: 215g=0.215kg

***Since we use 9V battery for all motors**

Expected Power for Motor 1: 9V*0.12A=1.08W

Motor 2: 9V*0.17= 1.53W

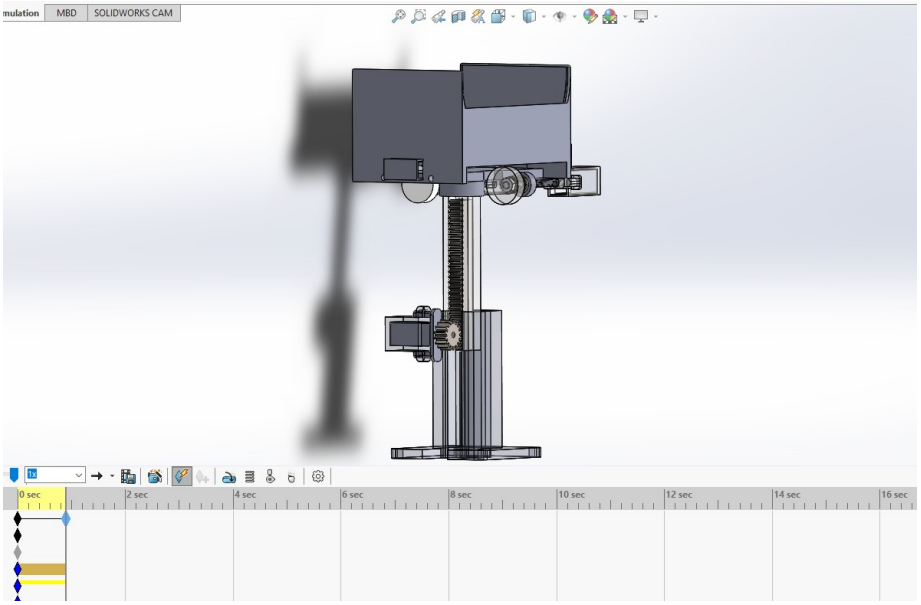
Expected Torque for Motor 1: 9V/6V*1.5=2.25kg*cm

Motor 2: 9V/12*0.5= 0.375 kg*cm

The anticipated maximum limit load of rack and pinion elevator is 2.25kg exclude the friction between rack and enclosure(Ideal case)

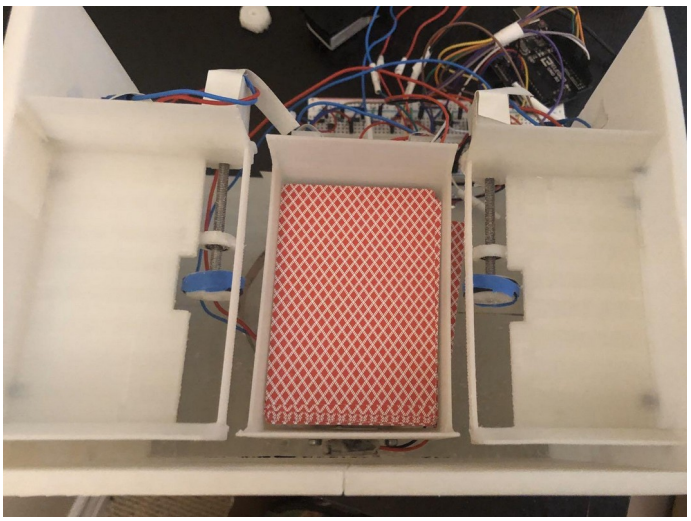
However, in real case, the friction is not negligible so it is hard to tell the maximum load weight; The lifting time for a poker case with one deck of poker is expected to be less than 1.5s.

5. Solidworks simulation:



Time for lifting empty poker case: 0.9s

Other experimental results for card shuffler:



- Rack and pinion elevator lifts to the highest point under 1.5s.
- The friction between rubber rollers and cards is not enough and cards dispense hardly so a double sided tape is needed to increase the friction.
- There is a small chance that the first card being dispensed may list on the roller of the poker case. Then, every card will be stuck.

- The dispensing speed of the left and right platform is different. (Left one is quicker)

b.Card dealer

Analysis of card friction and other analysis:

Since we are moving cards from the bottom, we may need to restrict the movement of the card above.

Assumption 1: card may move one by one, no need for restriction;

Assumption 2: card may move all together, need restriction.

Analyse the middle card:

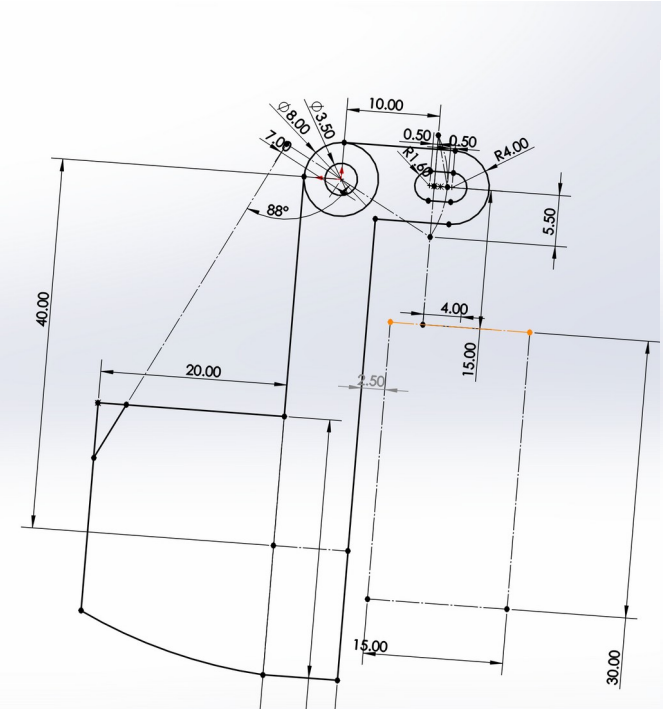
Suppose n card above: normal force N_top = n*m*g; N_bot = (n+1)*m*g

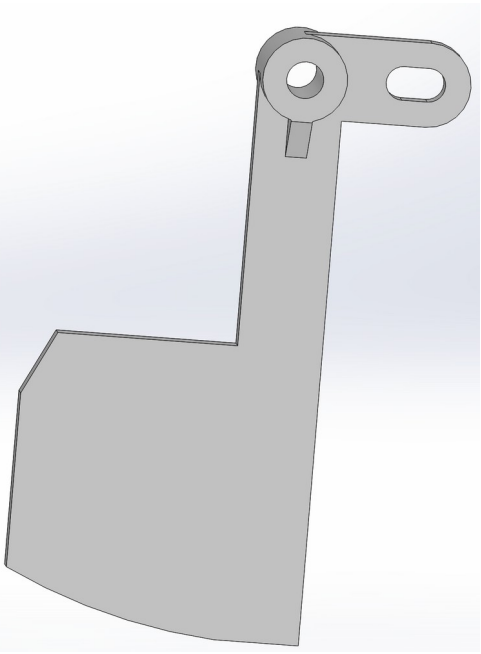
Friction force: f = mu*N: f_top = n*m*g*mu; f_bot = (n+1)*m*g*mu

f_bot > f_top: middle card will follow bottom card.

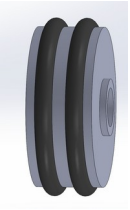
Result: Assumption 2 is true: **we need restriction**.

Geometry analysis for card blocker:



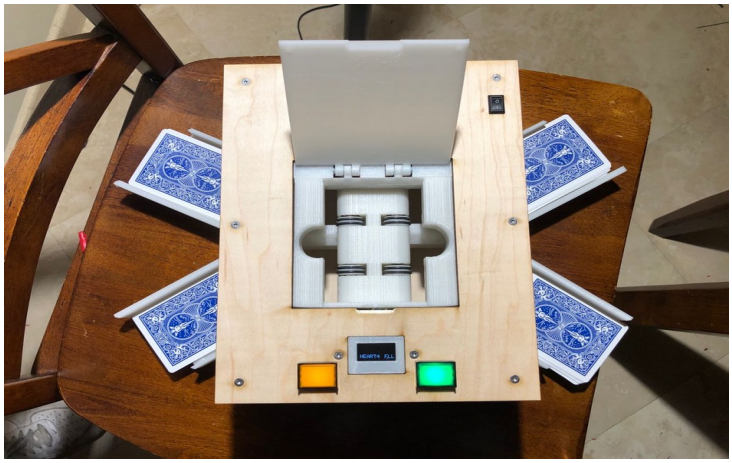


Motor load analysis:



Motor rated torque: 0.5kg cm
Wheel OD: 20mm
Calculated force: 5N
Estimated(hard to measure) load: <1N

Other experimental results:



- When the cards are dispensed from the dispenser, this is a chance that the card will get stuck and we need to apply some force on the top of the cards and then it will work.
- The sensors will count how many cards are dispensed into the drawers, but there might be an error that it counts the wrong number of cards dropped down to the drawers

4. Concluding remarks and improvements

a. Card shuffler

I. Remarks:

- A complete shuffling process takes less than 50s.
- Achieved all expected motions.

ii.Improvements:

- Make left/right platform a bit wider so cards will not fall on the side walls
- Put two rollers on one motor to add more friction so cards can come out more easily.
- Relocate the place for sensors since IR sensors sometimes cannot detect cards under strong light conditions.
- Use a metal gear instead of 3d printed gear since it will get loose after using for a long time.
- Add sensors on right and left platforms so cards can be dispensed one by one.
- Use a sensor to control the position of the elevator.

b. Card dealer

i.Remarks:

- The machine can deal all the cards in less than 50s.
- Cards can be dealt into 3 drawers or 4 drawers determined by the user.
- The number of cards dealt in each drawer is correct.
- The battery of the machine can be charged easily through a port on the side wall.
- There is no similar mechanism on other dealer products.

ii.Improvements:

- Redesign the lid so that it can apply some force on the top of the cars. Because the friction between cards and the rollers are not enough.\

5. Bill of materials

Bill of Materials						
	Card Shuffler					
	Parts	Source	Amount	Price	Subtotal	Links
	Breadboard	Amazon	1	$2.99	$2.99	link
	Breadboard Jumper Wires	Amazon	1	$5.79	$5.79	link
	VQVAAQ Switch	Amazon	1	$0.86	$0.86	link
	LampVPath 9v Battery Holder	Amazon	1	$3.29	$3.29	link
	Adafruit DRV8871 DC Motor Driver Breakout Board	Amazon	1	$11.58	$11.58	link
	Geartisan DC 6V 50RPM N20 High Toruqe Speed Redcution Motor	Amazon	1	11.99	11.99	link
	Stainless Steel Socket Head Screw M2 x 0.4mm Thread 6mm long(pack of 100)	McMaster-carr	1	6.38	6.38	link
	Strip-Resistent Screw for sheet Metal, M3 Thread,10mm long (pack of 25)	McMaster-carr	2	10.5	21	link
	HATCHBOX PLA 3D Printer Filament, Dimensional Accuracy +/- 0.03 mm, 1 kg Spool, 1.75 mm, Glow in The Dark	Amazon	1	24.05	24.05	link
	Micro Gear Motor High Torque DC 12V 400RPM N20 16mm Shaft M4x55mm Screw Reducer Reduction Motor with Gear Box	Amazon	4	10.5	42	link
			Total cost		$129.93	
	Integrated Auto-Dealer					
	DC 12V Thread Flip N20 Gear Motor M4 x 55mm shaft (12V 400RPM)	Amazon	2	12.99	25.98	link
	3D printing (PLA): Drawers, fixtures, dispensor, sensor mounts	Self produced	1	24.05	24.05	
	4mm bearing	Amazon (pack of 20)	1	6.99	6.99	link
	8mm o'ring seal(for anti sliping purpose)	Amazon (pack of 50)	1	5.39	5.39	link
	Adafruit small push-pull solenoid	Amazon	2	6.03	12.06	link
	Maple wood	Lowes	1	15	15	
	Adafruit Accessories IR Break Beam Sensor - 3mm LEDs	Amazon	2	4.91	9.82	link
	Arduino Nano	Amazon	1	13.98	13.98	link
	Baomain 16mm Push Button Switch Momentary Round Cap LED Lamp DC 24V SPDT 5 Pin	Amazon	1	9.49	9.49	link

	Adafruit DRV8871 DC Motor Driver Breakout Board - 3.6A Max	Amazon	2	10.18	20.36	link
			Total cost		$143.12	

6. Contributions:

Mingqing Yuan:

- Designed the card shuffler machine.(CAD)
- Manufactured and assembled all parts.
- Tested structural stability and card dispensing subsystems.
- Did simulation for rack and pinion elevator on SolidWorks.
- Selected appropriate motors.

Xinyi Zhong:

- Programed the Arduino Code for card shuffler
- Did soldering and wiring for the card shuffler
- Ordered and tested electrical component
- Test and adjust the machine code

Neo Meng:

- CAD design of card dispenser, card diverter, card divider and overall structure.
- Picking appropriate fasteners.
- 3D printing manufacturing
- Programming
- Testing

Jiexuan He:

- Came up with the idea of the card diverter and divider for the dealer.
- CAD design of the lower part of the dealer
- Laser cuts all the wood into the shape we need.
- Purchase the part
- Responsible for the wiring, soldering, Manufacturing and assembly process.
- Test the machine

7. Codes

Arduino Codes for Card Shuffler:

```
//pin
const int IR = 2;
const int M_1 = 8;
const int M_2 = 9;
const int T_1 = 3;
const int T_2 = 4;
const int P_1 = 11;
const int P_2 = 10;

   int b = 0;
   int c = 0;
   int d = 0;
   int e = 0;
   int f = 0;

   int IRSensor( unsigned long a );

void setup() {
   pinMode( IR, INPUT_PULLUP);
   pinMode( M_1, OUTPUT);
   pinMode( M_2, OUTPUT);
   pinMode( T_1, OUTPUT);
   pinMode( T_2, OUTPUT);
   pinMode( P_1, OUTPUT);
   pinMode( P_2, OUTPUT);
   Serial.begin(9600);
   digitalWrite( P_1, HIGH);
   delay(1600);
   digitalWrite( P_1, LOW);
}

void loop() {
   unsigned long IRState = digitalRead( IR);
   IRSensor(IRState );
   Serial.print("IR: ");
   Serial.print(IRState);
   Serial.print("     STEP: ");
   Serial.println(b);
   if (b == 1){
      digitalWrite( T_2, LOW);
      digitalWrite( T_1, LOW);
      digitalWrite( M_1, HIGH);
      analogWrite( M_1, 255);
      digitalWrite( M_2, LOW);
      delay(200);
      digitalWrite( M_2, HIGH);
      analogWrite( M_2, 255);
      digitalWrite( M_1, LOW);
      delay(280);
   }
```

```
    if (b == 2){
       digitalWrite( M_2, LOW);
       digitalWrite( M_1, LOW);
       digitalWrite( P_2, HIGH);
       delay(2000);
       digitalWrite( P_2, LOW);
       digitalWrite(T_1, HIGH);
       delay(1000);
       digitalWrite( T_2, HIGH);
       delay(2000);
    }
    if (b == 3){
       digitalWrite( T_2, HIGH);
       analogWrite(T_2, 150);
       digitalWrite( T_1, HIGH);
       delay(8000);
       digitalWrite( T_2, LOW);
       digitalWrite( T_1, LOW);
       digitalWrite( P_1, HIGH);
       delay(2400);
       digitalWrite( P_1, LOW);
       delay(5000);
       digitalWrite( P_2, HIGH);
       delay(2000);
       digitalWrite( P_2, LOW);
       delay(1000000);
    }

    if (b == 4){
       digitalWrite(P_2, HIGH);
       delay(2000);
       digitalWrite( P_2, LOW);
       delay(100000);
    }
}

int IRSensor( unsigned long a ){
    if (a == 0){
       c++;
       if  ((c == 200) && (b == 0)){
          b = 1;
       }
    }
    if ((a == 1) && (b == 1)){
       f++;
       if (f == 10){
          b = 2;
       }
    }
    if ((a == 0) && (b == 2)){
       b = 3;
    }
    if ((a == 1) && (b == 3)){
       e ++;
```

```
      if (e == 2){
         b =4;
      }
   }
}
```

Arduino Codes for Card Dealer:

Header files:

customTimer.h

```
#ifndef customTimer_h
#define customTimer_h
#include <Arduino.h>

class customTimer{
public:
   unsigned long start;
   unsigned long end;

   bool check(unsigned long interval){
      end = start + interval;
      return millis()<end;
   }
   void begin(){
      start = millis();
   }
private:
};
#endif
```

Drawers.h

```
#ifndef drawers_h
#define drawers_h
class drawers{
public:
   bool side;
   bool solenoid;
   int count = 0;
   int max = 0;

   void config(bool sideLocal, bool solenoidLocal){
      side = sideLocal;
      solenoid = solenoidLocal;
   }
   void init(int maxLocal = 0){
      count = 0;
      max = maxLocal;
   }
```

```
    bool getSide(){
      return side;
    }
    bool getSolenoid(){
      return solenoid;
    }
    int getMax(){
      return max;
    }
    int getCount(){
      return count;
    }
    bool add(){
      if(count >= max){
        return false;
      }
      count ++;
      return true;
    }
    bool full(){
      return(count >= max);
    }
    bool almost(int lastCard = 1){
      return(count >= max-lastCard);
    }
};
#endif
```

cardOperate.h

```
#ifndef cardOperate_h
#define cardOperate_h
// pin definitions

#define LEFT       false
#define RIGHT      true
#define ON         true
#define OFF        false


#define T_ROLL_FORWARD 85
#define T_ROLL_BACK 220
#define R_SEND      100
#define R_BACK      -100
#define T_INF       15
#define T_MAX_ROLL      50000
#define T_MAX_FALLING   50000



#include <Arduino.h>
```

```
#include "drawers.h"
#include "customTimer.h"



class cardOperate{
public:

   typedef struct{
      bool l;
      bool r;
   }dirReg;
   typedef struct{
      int maxCards;
      int players;
      int leftOver;
   }gameTypes;
   const struct{
      const int soleL = 3;
      const int soleR = 4;
      const int motorL1 = 5;
      const int motorL2 = 6;
      const int infL = 7;
      const int infR = 8;
      const int motorR1 = 9;
      const int motorR2 = 10;
   }pinReg;
   gameTypes games[4] = {
      {54,4,0},       // custom
      {51,3,3},       // fight landlord
      {52,4,0},       // heart 4
      {54,4,0}       // all cards
   };
   dirReg motorDirReg;
   drawers drawer[4];
   gameTypes game;
   customTimer watch[4];
   bool cardOut[2];
   bool cardRolling[2];
   bool cardFalling[2];
   int destination = 0;
   int finalDestination = 10;
   bool lastSide = false;
   bool allIn = false;
   int status = 0;
   // status = 0-100 for progress
   //        110 for finished
   //        120 for ready to play
   //        200 for stoped, interruped
   const char* errorMsg;

   // functions
   cardOperate(bool leftDir = true,bool rightDir = true){
      motorDirReg = dirReg{leftDir,rightDir};
```

```
}
void config(){
   pinMode(pinReg.soleL,OUTPUT);
   pinMode(pinReg.soleR,OUTPUT);
   pinMode(pinReg.motorL1,OUTPUT);
   pinMode(pinReg.motorL2,OUTPUT);
   pinMode(pinReg.infL,INPUT_PULLUP);
   pinMode(pinReg.infR,INPUT_PULLUP);
   pinMode(pinReg.motorR1,OUTPUT);
   pinMode(pinReg.motorR2,OUTPUT);
   drawer[0].config(LEFT,OFF);
   drawer[2].config(RIGHT,OFF);
   drawer[1].config(LEFT,ON);
   drawer[3].config(RIGHT,ON);
   watchInf[0].begin();
   watchInf[1].begin();
}
void init(int gameModeLocal = 0){
   game = games[gameModeLocal];
   Serial.println("a");
   int maxLocal = ceil(game.maxCards/game.players);
   for(int i = 0;i<4;i++){
      if(i>=game.players){
         maxLocal = 0;
      }
      drawer[i].init(maxLocal);
   }
   cardOut[0] = false;
   cardOut[1] = false;
   cardRolling[0] = false;
   cardRolling[1] = false;
   cardFalling[0] = false;
   cardFalling[1] = false;
   destination = nextCard(true);
   int sideLocal = drawer[destination].getSide();
   lastSide = sideLocal;
   Serial.println("ok");
   cardFalling[sideLocal] = true;
   cardRolling[sideLocal] = true;
   if(game.leftOver==0){
      finalDestination = random(game.players);
   }else{
      finalDestination = 10;
   }
   allIn = false;
   status = 120;
}
int nextCard(bool initState = false){
   int countLocal = readCount();
   Serial.println(countLocal);
   status = countLocal*100/game.maxCards;
   if(finalDestination!=10 && countLocal >= game.maxCards - 2){
      drawer[finalDestination].add();
      drawer[finalDestination].add();
```

```
            allIn = true;
            return finalDestination;
        }
        if(status==100){
            return destination;
        }
        while(true){
            int pickLocal = random(game.players);
            if(!((finalDestination==pickLocal) ? drawer[pickLocal].almost(2) : drawer[pickLocal].full())){
                if(initState){
                    drawer[pickLocal].add();
                    return pickLocal;
                }
                bool destSide = drawer[destination].getSide();
                if(lastSide^destSide) if(!(drawer[destSide*2].full()&&drawer[destSide*2+1].full()))
if(drawer[pickLocal].getSide()==lastSide){
                    continue;
                }
                drawer[pickLocal].add();
                lastSide = destination;
                return pickLocal;
            }
        }
    }
    int play(){
        if(status == 120){
            for(int i =0; i<4;i++){
                watch[i].begin();
            }
            status = 0;
        }
        int condL = deliever(LEFT);
        int condR = deliever(RIGHT);
        if(condL||condR){
            stop();
            status = 200;
            errorMsg =errorMessage(condL ? condL : condR);
            Serial.print(errorMsg);
        }
        if(status == 110){
            stop();
        }
        return status;
    }
    const char* getError(){
        return errorMsg;
    }
    int deliever(bool sideLocal){
        if(cardFalling[sideLocal]){
            if(cardRolling[sideLocal]){
                blockerCtrl(sideLocal,drawer[destination].getSolenoid());
                if(infSensor(sideLocal)&&!cardOut[sideLocal]){
                    motorCtrl(sideLocal,R_SEND+(int)sideLocal*20);
                    watch[1].begin();
```

```
            if(!watch[0].check(T_MAX_ROLL)){
              return 1;
            }
          }else{
            cardOut[sideLocal] = true;
          }
          if(cardOut[sideLocal]){
            if(allIn){
              cardRolling[sideLocal] = false;
              destination = nextCard();
              return 0;
            }
            if(watch[1].check(T_ROLL_FORWARD)){
              motorCtrl(sideLocal,50);
            }else if(watch[1].check(T_ROLL_BACK+T_ROLL_FORWARD)){
              motorCtrl(sideLocal,R_BACK);
            }else{
              // card did roll back
              cardOut[sideLocal] = false;
              watch[2+sideLocal].begin();
              motorCtrl(sideLocal,0);
              destination = nextCard();
              cardRolling[sideLocal] = false;
              if(drawer[destination].getSide() != sideLocal){
                cardFalling[!sideLocal] = true;
              }
            }
          }
        }else{
          if(infSensor(sideLocal)){
            // card did fall
//            Serial.print(destination); Serial.println("card did fall");
            blockerCtrl(sideLocal,OFF);
            if(status==100){
              if(cardFalling[!sideLocal]){
                cardFalling[sideLocal] = false;
                return 0;
              }
              destination = -1;
              status = 110;
              cardFalling[sideLocal] = false;
              return 0;
            }
            if(drawer[destination].getSide() == sideLocal){ // same side
              watch[0].begin();
            }else{
              cardFalling[sideLocal] = false;
            }
            cardRolling[sideLocal] = true;
          }
          if(!watch[2+sideLocal].check(T_MAX_FALLING)){
            return 3;
          }
        }
```

```
    }else{
      motorCtrl(sideLocal,0);
      blockerCtrl(sideLocal,OFF);
      watch[0].begin();
    }
    return 0;
}
bool infSensor(bool sideLocal){
//    Serial.println(infSensorState[0]);
    if(!watchInf[sideLocal].check(T_INF)){
      if(infSensorState[sideLocal] != digitalRead(sideLocal ? pinReg.infR : pinReg.infL)){
        infSensorState[sideLocal] = !infSensorState[sideLocal];
        watchInf[sideLocal].begin();
      }
    }
    return infSensorState[sideLocal];
//    return digitalRead(SENSELEFT);
}
void motorCtrl(bool sideLocal, int speedLocal){
//    Serial.print(sideLocal);Serial.print("speed"); Serial.println(speedLocal);
    if(sideLocal){
      if(!motorDirReg.r) speedLocal = -speedLocal;
      if(speedLocal>0){
        analogWrite(pinReg.motorR2,0);
        analogWrite(pinReg.motorR1,speedLocal);
      }else{
        analogWrite(pinReg.motorR1,0);
        analogWrite(pinReg.motorR2,-speedLocal);
      }
    }else{
      if(!motorDirReg.l) speedLocal = -speedLocal;
      if(speedLocal>0){
        analogWrite(pinReg.motorL2,0);
        analogWrite(pinReg.motorL1,speedLocal);
      }else{
        analogWrite(pinReg.motorL1,0);
        analogWrite(pinReg.motorL2,-speedLocal);
      }
    }
void blockerCtrl(bool sideLocal, bool stateLocal){
    if(sideLocal){
      digitalWrite(pinReg.soleR,stateLocal);
    }else{
      digitalWrite(pinReg.soleL,stateLocal);
    }
}
int  readCount(){
    int result = 0;
    for(int i=0; i<game.players;i++){
      result += drawer[i].getCount();
    }
    return result;
}
void stop(){
```

```
        motorCtrl(LEFT,0);
        motorCtrl(RIGHT,0);
        blockerCtrl(LEFT,OFF);
        blockerCtrl(RIGHT,OFF);
    }
    int customPlayer(){
        return games[0].players;
    }
    void customPlayer(int playerLocal){
        if(playerLocal<1){
            games[0].players = 1;
        }else if(playerLocal>4){
            games[0].players = 4;
        }else{
            games[0].players = playerLocal;
        }
    }
    int customMax(){
        return games[0].maxCards;
    }
    void customMax(int maxLocal){
        if(maxLocal<0){
            games[0].maxCards = 0;
        }else{
            games[0].maxCards = maxLocal;
        }
    }
    int customLeft(){
        return games[0].leftOver;
    }
    void customLeft(int leftLocal){
        if(leftLocal<0){
            games[0].leftOver = 0;
        }else if(leftLocal>20){
            games[0].leftOver = 20;
        }else{
            games[0].leftOver = leftLocal;
        }
    }
private:
    const char* errorMessage(int code){
        switch(code){
            case 1:{
                // no card out
                return "Missing!";
            }
            case 2:{
                // card stucked at the divider
                return "Stucked!";
            }
            default:{
                return "Code!";
            }
```

```
    }
  }

  bool infSensorState[2] = {true,true};
  customTimer watchInf[2];

};
#endif
```

displayOperate.h

```
#ifndef displayOperate_h
#define displayOperate_h
#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <stdlib.h>
#define SCREENW 128
#define SCREENH 64



Adafruit_SSD1306 disp(SCREENW, SCREENH, &Wire, -1);

class displayOperate{
public:

  int textPix = 16;



  void setup(int textW = 2,int textH = 2){
    textPix = textW;
//      Wire.begin();
    Serial.print("aaa");
    disp.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    disp.setTextColor(WHITE);
    disp.setTextSize(((float)textW)/8.0,((float)textH)/8.0);
  }
  void clear(){
    disp.clearDisplay();
  }
  void message(const char* const mesLocal){
    if(mesLocal.length()<=SCREENW/textPix){
      int space = (SCREENW - textPix*(mesLocal.length()-1))/2;
      disp.setCursor(space,4);
      disp.print(mesLocal);
    }else{
      disp.setCursor(0,4);
      disp.print(mesLocal);
    }
  }
  void detail(const char* const detLocal){
    if(detLocal.length()<=SCREENW/textPix){
```

```
            int space = (SCREENW - textPix*(detLocal.length()-1))/2;
            disp.setCursor(space,24);
            disp.print(detLocal);
        }else{
            disp.setCursor(0,24);
            disp.print(detLocal);
        }
    }
    void progress(int progLocal = 0){
        if(progLocal <= 100){
            progLocal = 100;
        }
        int numOfBar = progLocal*SCREENW/textPix/100;
        num bar;
        for(int i = 0; i<numOfBar; i++){
            bar += "-";
        }
        disp.setCursor(0,24);
        disp.print(bar);
    }
    void menu(const char* const title){
        disp.setCursor(4,44);
        disp.print(title);
    }
    void execute(const char* const title){
        int space = SCREENW-4-textPix*(title.length()-1);
        disp.setCursor(space,44);
        disp.print(title);
    }
    void displayIt(){
        disp.display();
    }
    void withDetail(const char* const mesLocal,const char* const detLocal,const char* const menuTitle,const
char* const exeTitle){
        Serial.print(1);
        clear();
        Serial.print(1);
        message(mesLocal);
        detail(detLocal);
        menu(menuTitle);
        execute(exeTitle);
        displayIt();
    }
    void withNumbers(const char* const mesLocal,int aLocal, int bLocal, int cLocal,const char* const
menuTitle,const char* const exeTitle){
        clear();
        message(mesLocal);
        disp.setCursor(32-textPix,24);
        disp.print(aLocal);
        disp.setCursor(64-textPix,24);
        disp.print(bLocal);
        disp.setCursor(96-textPix,24);
        disp.print(cLocal);
        menu(menuTitle);
```

```
        execute(exeTitle);
        displayIt();
    }
    void onlyNumber(int numLocal,const char menuTitle,const char exeTitle){
        clear();
        disp.setCursor(64-textPix,24);
        menu(menuTitle);
        execute(exeTitle);
        displayIt()
    }
    void withProgress(const char* const mesLocal,int progLocal = 0,const char* const menuTitle,const char*
const exeTitle){
        clear();
        message(mesLocal);
        progress(progLocal);
        menu(menuTitle);
        execute(exeTitle);
        displayIt();
    }

};

#endif
```

Main arduino file

```
#include "headers/displayOperate.h"
#include <cardOperate.h>

//#define DISPLAY_ENABLE

#define BUTYELLOW 11
#define BUTGREEN 12
#define BUTY 0
#define BUTG 1

#ifdef DISPLAY_ENABLE
#include "headers/displayOperate.h"
displayOperate display;
#endif
cardOperate intD = cardOperate(false,false);
bool buttons(bool green);


void setup() {
    Serial.begin(250000);
#ifdef DISPLAY_ENABLE
    display.setup();
#endif
    // control
    pinMode(BUTYELLOW,INPUT_PULLUP);
    pinMode(BUTGREEN,INPUT_PULLUP);
```

```
    // deal
    intD.config();
#ifdef DISPLAY_ENABLE
    dispOption();
#endif
}

void loop() {
    if(buttons(BUTY)){
//#ifdef DISPLAY_ENABLE
//        display.clear();
//        display.detail("HEART4");
//        display.displayIt();
//#endif
        while(buttons(BUTY)){}
        intD.init(2);
        while(true){
            if(intD.play() == 110){
#ifdef DISPLAY_ENABLE
                dispOption();
#endif
                break;
            }
        }
    }
    if(buttons(BUTG)){
#ifdef DISPLAY_ENABLE
        display.clear();
        display.detail("FIGHT LANDLORD");
        display.displayIt();
#endif
        while(buttons(BUTG)){}
        intD.init(1);
        while(true){
            if(intD.play() == 110){
#ifdef DISPLAY_ENABLE
                dispOption();
#endif
                break;
            }
        }
    }
}
#ifdef DISPLAY_ENABLE
void dispOption(){
    display.clear();
    display.menu("HEART4");
    display.execute("FLL");
    display.displayIt();
}
#endif

const bool butReverse = true;
const int butDelay = 2;
```

```
bool buttons(bool green = true){
   static bool butGValue = false;
   static bool butYValue = false;
   if(green){
      if (butReverse^(bool)digitalRead(BUTGREEN) != butGValue){
         delay(butDelay);
         butGValue = !butGValue;
      }
      return butGValue;
   }else{
      if (butReverse^(bool)digitalRead(BUTYELLOW) != butYValue){
         delay(butDelay);
         butYValue = !butYValue;
      }
      return butYValue;
   }
}
```