Jack of All Trades

Miya Higuchi, Mason Loyet, and Rachel Ratnam

Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—A system that facilitates remote gameplay with physical cards. For those who wish to play card games with friends and family in separate locations, the current alternatives available are online websites and video game software. Unlike these applications, our system aims to make card games such as Go Fish, Rummy, and Euchre possible to play remotely while retaining the authentic feeling of holding physical cards. By utilizing computer vision and machine learning, players are now able to manually place cards on the table and draw from a deck to progress a remote game.**

***Index Terms*— computer vision, LCD screen, machine learning, peripherals, Raspberry Pi, TCP socket, thermal printer, tty, YOLO algorithm**

I. Introduction

Playing cards is a fun activity to spend with family members and friends, but it is difficult to do when separated by great distances. The current alternatives for remote gameplay are purely virtual, consisting of online sites and applications that lose the authentic feeling of holding and placing cards. Inspired by the nostalgia of playing with physical cards, with our system, remote gameplay using physical cards is now possible.

As the main appeal of our system is its tangibility, our approach aims to emulate the in-person playing experience while preserving simple gameplay progression. Specifically, players can manually draw and place cards by hand as their "move" as opposed to having to strictly interact with software. In addition to the primary focus on physical cards, our system aims to have multiplayer backing, game logic for specific card games, and support for concurrent games. For a deck of cards to be referenced by players in separate locations, our system will also include card dealing and card recognition to ensure that inconsistencies such as card duplicates will not occur.

For the actual implementation, the physical components of our system can be organized into input, output, and dealing devices. The keyboard will help progress the game where user input is needed. For example, during a game of *Go Fish*, players will need to input to the system which card rank they are requesting. The server needs to be able to know what the request is, in order to relay the information to the opponent and progress the game state. The camera module input device will act as the "eyes" of our system. It will capture the cards the user "plays" and upload the information to the server, updating the gamestate. The output device is our LCD screen, which displays the current gamestate and items of action to the user. For instance, the screen may indicate the move the opponent has just made. Lastly, for the scope of this project, the dealing device will be a thermal printer instead of a mechanical sorting machine. The thermal printer, connected to the Raspberry Pi, will print out the cards as a means of dealing. Cards will be printed out on receipt paper and supplemented with cardstock.

II. Use-Case Requirements

The specifications for our system mainly aim to satisfy user experience during gameplay and mitigate any inconveniences that might arise due to the remote limitations.

One specification for our project includes the minimum playing space that needs to be allocated for card recognition. The user will want sufficient enough space to play cards but will not want to have to provide an unreasonable amount either. Most popular board games are between 15" by 15" (*Scrabble*) and 20" by 30" (*Risk*). For the scope of this project, the area will be approximately 18" by 24" of table top space, such that it is within the limits of convenience for the player. This space will account for the area of all our physical devices, as well as the area necessary for the camera to recognize cards being played.

Furthermore, since the system will be requiring connectivity to the server during gameplay, the user should be able to join the lobby as long as they have access to WiFi as well as an outlet to sufficiently power the devices. Once setup is completed, connecting and initializing a game between players must take no longer than 10 seconds. This metric is derived from the amount of time consumers are usually willing to spend for websites or video games to load.

Due to our system using a thermal printer to deal cards, , we want cards to be dealt at a rate of at least one card every two seconds. This is so the player doesn't have to be idle for too long during the beginning phases of the game, such as when players are being dealt their preliminary hands. Since the players will not have to wait for their opponent's card to be dealt with the same device, this frequency is more closely modeled to the speed of casual gameplay.

Additionally, for card legibility, the fonts used should be at least 16 pixels (size of 12 point font). Most printing media use and recommend a 12 point font, so this metric was decided in order for the cards to be readable for a broader range of players. Also, the physical card sizes must be at least 3" by 2". Standard playing cards used by casinos and professional settings are typically 3.5" by 2.5", but for our project, the cards will be 3" by 2" (slightly larger than standard mini sized cards).

For specific games that will be implemented, our minimum viable product will be to have a fully functional remote game of Go Fish.

III. Architecture and/or Principle of Operation

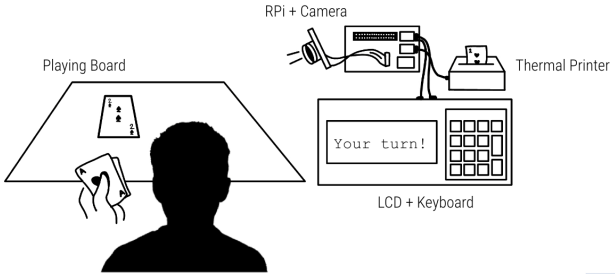

Fig. 1. Mockup illustration of the device setup

The physical set up of our system is illustrated in Figure 1. The player will be sitting adjacent to the devices which will be connected to the Raspberry Pi.

To initiate and join a game, participating players must join a game lobby. For a given game lobby there exists a server (which may be hosting several game lobbies concurrently) and a set of 2-5 devices. Each device system allows exactly one person to interface with the game, and can be located anywhere as long as it is connected to power and wifi. Devices do not communicate point-to-point with each other. Instead, each device system has a single TCP socket connection to the server over which all communication occurs. The Raspberry Pi serves as the central hub that connects to our peripheral devices, relaying the inputs and actions from the player to the server. Using the Raspberry Pi, information in the form of JSON objects will be sent to the server to be interpreted as commands and responses.

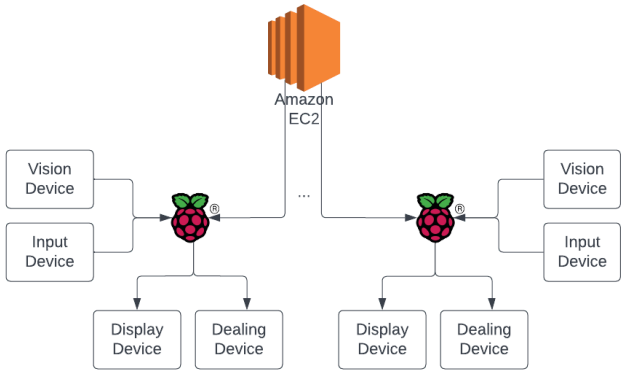

Fig. 3. System Block Diagram

The specific configurations of each device to the Raspberry Pi are shown in Figure 2. The keyboard will be connected to the Raspberry Pi via the USB port, LCD screen will connect to the GPIO pins, the printer is connected via UART wires, and the Raspberry Pi Camera module's ribbon connector will be secured to the Raspberry Pi Camera Serial Interface (CSI) port.

RaspianOS will take care of the device drivers necessary to communicate with the keyboard and camera. For the LCD screen and printer, they will be interfaced separately.

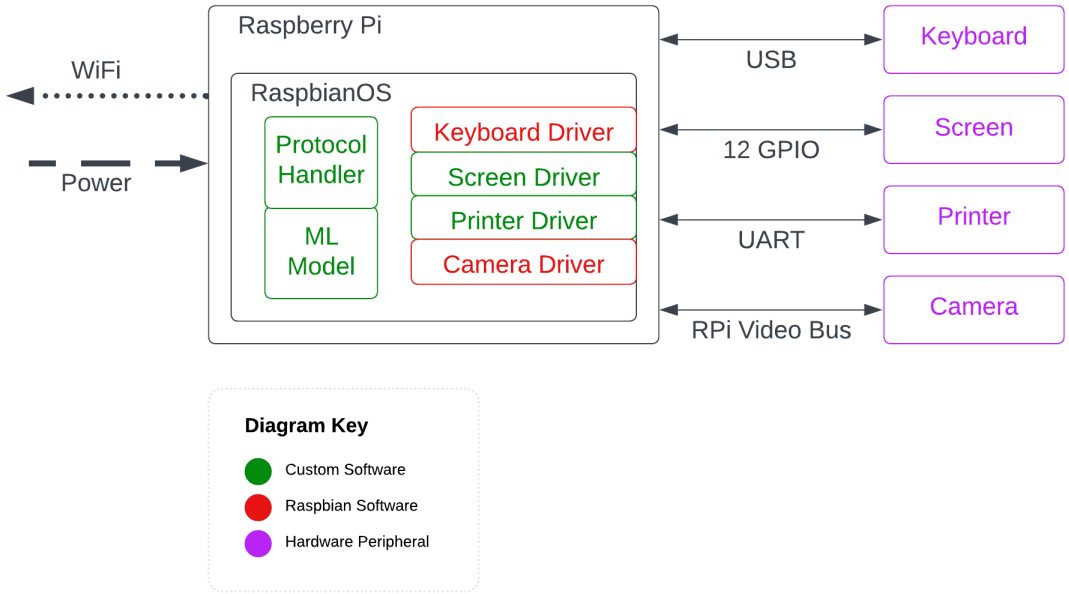

Fig. 2. Device Block Diagram

IV. Design Requirements

The most noticeable aspect of playing a card game is the cards. Our use case requirements state that we need to be able to play with physical cards, but in order to play with physical cards in an acceptable way, we need to set some quantitative requirements.

The cards need to be printed accurately. We expect for the printer to be able to flawlessly print an entire deck of cards such that both a human and the camera is able to detect every card. In order to meet this requirement, the design must be resistant to misprints. The suit and number will be at least double redundant in the bottom left and upper right corners.

Also, the cards need to be printed quickly. In a game of cards using a real deck and human dealer, the dealer will shuffle the deck a few times before dealing out the cards. If the shuffling takes about 5 seconds and 4 cards are dealt a second, it would take 10 seconds for a human dealer to deal 5 cards to 4 people. Since we do not have to shuffle the cards and we can deal all the cards in parallel, our design requirement is to deal 1 card every 2 seconds to match this real world benchmark.

When coupling a keyboard and a screen it is critically important that there is virtually no latency between typing and having the text appear on the screen in order for the system to feel responsive. Studies[3] show that there is a 40 ms window to show the result of an input, but there are conflicting studies, so we will require a 20 ms update latency for our LCD screen.

When playing a card game in person, you are visually able to see your opponents move almost instantly. In order to accurately emulate the gameplay experience, we must have the latency from card play to card reporting be sufficiently small. Human reaction time is about a quarter of a second[2], so the total latency between the time that a player plays a card and the card play being reported on all other devices must be less than 0.25 seconds.

This classification latency leads to the strictest subsystem requirements for almost all components. The allocation of time is as follows: first, we want the classification time to be around 25 ms. Then, the camera image capture time must be less than 100 ms. The network needs to also be less than 100 ms including encoding and decoding. Finally, as before, the LCD screen write latency should be less than 20 ms.

Another important aspect of playing cards is the amount of space where cards can be placed. For example, it would be very uncomfortable to play cards on an airplane tray table. We decided that a sufficient amount of space to hold the cards in a playing area with 5 people is 18 inches by 24 inches. This requirement means that our camera must be able to capture this region with enough accuracy to identify the cards. Since our receipts have precision of about 100 dpi, the image captured must be at least 2400x1800 pixels.

Finally, when not using the device it must be easy to store in a game cabinet. Our target shape is smaller than a shoebox (14 in x 10 in x 5 in) and lighter than 10lbs.

V. Design Trade Studies

In order to derive our final design decisions, we had to compare a lot of tradeoffs between various options and determine which one best satisfies our design requirements. Overall, we essentially had to choose the best ML algorithm, card dispensing device, camera module, and microcontroller.

A. Real Time Object Detection Algorithm

In terms of picking the best real time object detection algorithm, we were deciding between the Deformable Parts Model (DPM), Region-Based Convolutional Neural Network (R-CNN), and the You Only Look Once (YOLO) algorithms. In terms of picking the best algorithm, we first looked at what design requirements we needed to satisfy. Our goal for our system to be able to detect cards and be able to return its suit and rank was around 25 milliseconds. The DPM used a sliding window approach with multiple connected networks. However, because of these multiple networks it is slower and it uses static features causing it to also be less accurate. The second approach of R-CNN uses certain regions of the video to perform detections. However, this can lead to multiple detections of the same object and detecting the same card multiple times can lead to logic issues in a lot of our games. Additionally, each image can take more than 40 seconds to detect which does not align with our specifications for the design. The YOLO algorithm utilizes a single network, a grid, and feature recognition within each box of the grid which proves to be over 1000 times faster than the R-CNN and it tends to make approximately 50% of the errors that the optimized version of R-CNN (Fast R-CNN) makes. Due to this speed, we have decided to work with the YOLO algorithm for our project. The only setback for this algorithm is that it does not properly identify groups of very small times when they are close to larger items. However, this will not be an issue for us, since our camera will only be used on cards against a constant black background.

B. Card Dispensing Device

The next decision we had to make was how we wanted to actually deal the cards. When making this decision, we had two options. The first option was a mechatronic device that would have one set of standard playing cards in it. It would then contain a rotary device that would sort through the cards and then eject the card that is necessary at that moment. This first option has a lot of moving parts such as a rotating device that would be able to understand how much to rotate and how much to spin. It would then have to be able to eject one card. It would also have to be able to take in a card and place it in the rotating mechanism in the right order according to the card's rank and suit so that it can be properly found the next time around. The second option was to purchase a mini thermal printer. This printer would essentially serve the purpose of printing out disposable versions of the card as needed. So the shuffling of the cards would occur within the software and according to the game logic, certain cards will be assigned to a user. As a result of that, the printer would

proceed to print those cards out. To make it feel more like a card, the user can attach the printed receipt cards onto a piece of cardstock with a clip. At this point, we decided that both of these designs achieve our need to give cards to the user. Although, it more environmentally friendly to use the first option, due to the scope and time constraints of this project, it made more sense use the second option because no one on our team specializes in mechatronics or robotics so building this contraption would be out of scope and instead designing and coding cards to be printed from our thermal printer made more sense.

C. *Camera Module*

In terms of the camera we were going to use, the only specification we were looking for is easily integrable with our raspberry pi and computer vision model so that detection of cards was as efficient as possible. When looking for cameras that fit these specifications, it made the most sense to use one that was compatible with the raspberry pi since it has a camera driver. Therefore, we decided to use the raspberry pi camera module version 1 that works well with the raspberry pi. It also has a still resolution of 5 megapixels and sensor resolution of 2592 x 1944 pixels which was a high enough resolution and quality for our machine learning model to perform well. Another aspect of the camera module was its field of view because as mentioned by the design requirements, the camera will be propped up in a way such that it is overlooking the vision area, so the camera must have sufficient vertical and horizontal fields of view so that it is able to cover the entire area. With a horizontal field of view of 53.50 +/- 0.13 degrees and a vertical field of view of 41.41 +/- 0.11 degrees. Therefore, with these, it was sufficient to see the whole vision field and easily integrable with our chosen

D. *Microcontroller*

For the microcontroller of this project, it came down to two options. The first option was the STM32 microcontroller and the second was the Raspberry Pi 3. Both microcontrollers had their pros and cons. The STM32 was integrated with wireless wifi and it is a much smaller and cheaper device than the Raspberry Pi. However, the Raspberry Pi 3 has software that is meant to drive a lot of the components of our project. For example, we can utilize the camera driver and the keyboard driver provided by the Raspberry Pi and we can integrate it with an operating system that we develop. Therefore, due to these properties and advantages of the Raspberry Pi 3, we decided to go with that.

VI. System Implementation

As outlined in the architecture, the network topography forms a star with the server in at the hub and devices as peripherals. The server's software manages the connections to the peripheral devices and holds the consensus of the game state by authority. We emphasize this centralized state philosophy as much as possible by designing the devices to hold little to no state about the game. While connected to the server, they simply relay their I/O through messages.

The device itself runs the Raspberry Pi OS and attempts to follow Unix best practices to be as simple and infallible as possible. Specifically it interfaces with the camera, the receipt printer, the LCD screen and the keyboard through sysfs. The implementation of these file interfaces are explained in more detail in their respective sections.

A. *Server*

The server is an amazon EC2 instance running RHEL (Red Hat Enterprise Linux). We use RHEL as opposed to Amazon Linux because RHEL is more widely used, documented, and supported. On top of that, Amazon Linux could tie us to Amazon as our cloud provider since Amazon Linux only runs on EC2 instances. Our team also has experience using RHEL in the past.

The software running on the server will be written in Rust to allow for easier concurrency and greater static guarantees of memory safety. The architecture will be fairly simple. There will be a single acceptor thread that handles requests to connect to the server by either creating a new game lobby or adding the client to an existing lobby.

A lobby consists of a unique name supplied by the creator, a game type, and an expected number of players. The lobbies are associated with threads which will be taken from an idle thread pool by the acceptor thread. This model lends itself to the 3 states of threads depicted in Fig. 2. All threads begin idle before a lobby is attached. Then once the thread is given a lobby by the acceptor it is in the waiting state until the game is started. Then, once the expected number of players attach to the lobby and the game starts, the thread is put in the active state for as long as the game is being played. When the game ends, the lobby is disbanded and the thread returns to the idle state. If a game lobby is abandoned before it starts, the thread may return to the idle state without a game starting. This may also happen if the lobby sits unfilled for a long period of time. These considerations are important as to not exhaust our thread pool waiting for games that will never start.

Once a game starts, a game state object will be created to accompany the lobby object. Every game has a different game state object to reflect the rules of the game. While a game is being played, it will be in the game loop. At any given time

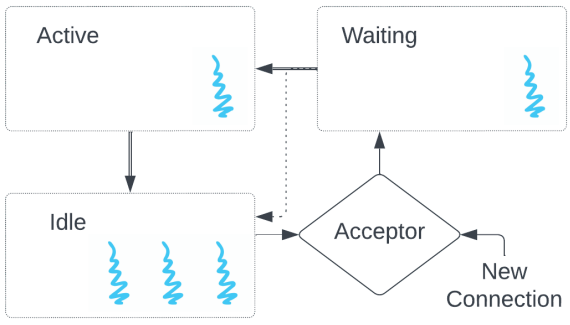

Fig. 4. Thread Pool Diagram

it is exactly one player's turn. This player is called the active player. A turn consists of two stages: the play stage where the active player does their move, and the reconciling stage where the active player's action is reconciled with all other players in the game.

During the play stage, the game state object will dictate what messages to send to the active player. The active player will then apply some input to the device which will be visible in the form of messages received on the server. All messages from the active player will be passed to the game state. The game state will indicate if the messages received have completed the active player's turn. For example, in Go Fish, the active player would be prompted to input a player and a suit to request. When the server receives messages that the player typed something into the keyboard, the server passes these messages to the game state to be verified. It may be the case that the game state needs to send more messages to the player if they give illegal or unexpected inputs.

After the play stage is complete, the reconciling stage begins. The game state will dictate what messages to send to all players. This time, all messages from all players will be sent to the game state object. Again, the game state object will indicate when the reconciling stage is complete. In the go fish example, this would include sending a message to the requested player to give up the cards of a certain suit. The player would place all such cards in the vision area. At the same time, the device can send a message to the active player to dispense the newly gained cards.

After the reconciling stage, the game may either be over, or play could continue with the next player's turn, starting at the play stage.

B. Server-Device Communication

The server-device communication is done point-to-point over TCP sockets. All messages are sent encoded as JSON objects. Encoding and decoding from objects is done by a Rust crate called serde. There is no device-device communication.

The protocol is designed to reflect the stateless nature of the devices. For example, lines could be buffered by the device, but are instead sent immediately to be buffered on the server side. Similarly, the server is responsible for tracking what cards were detected in the last scan, the device sends all cards detected.

The strings received in Line messages are exactly as expected except that the suit keys are encoded. The mapping is +, -, /, * to spade, club, heart, diamond. These keys will be repainted on the actual keyboard.

The LCD screen messages are fairly straightforward. Text strings are sent in the print messages. There are some embedded special characters in the string that will represent the suit characters. To be consistent, these special characters are the same as the characters sent by the keyboard. (+,-,/,*). There is also a command to clear the LCD screen. If text is sent that would overflow the LCD screen, the text will scroll down.

TABLE I. PROTOCOL MESSAGES

Source	**Message**	
	Name	***Description***
Server	`Clear`	Clear the LCD Screen
	`Deal(`*card*`)`	Dispense *card*
	`Print(`*s*`)`	Print *s* to LCD Screen
	`Scan`	Scan the playing area for cards
Device	`Line(`*s*`)`	The keyboard received the line *s*
	`Detect(`*cards*`)`	The camera detected *cards*

The Deal message works exactly like expected. Given a card to print, the device will send a message to the printer to dispense the card.

In order to make the device detect cards, a Scan request must be sent. Then, the device will take a snapshot of the board, run it through the detection model, and send the matched cards over the connection in a Detected message. Every scan will result in exactly one Detected message. It is undefined behavior to send two Scan messages before receiving a Detected message. Observing the time elapsed between a Scan message and Detected message is a tangible target metric for the overall card detection latency.

C. Device Supervisor

On the raspberry pi there will be a user space program called the device supervisor. This program establishes the network connection to the server and sends and receives all messages.

On the raspberry pi's boot, this process will start. It is responsible for running ensuring that all the peripherals exist and are initialized and any associated processes are started. In accordance with Unix principles, we will attempt as best as possible to have all peripherals represented as files. See the peripheral subsystem sections to see how this is done.

This program prompts the user to enter the lobby and game information before any connection to the server is established.

Once the server connection is established, it will communicate with the 4 peripherals in order to send and receive messages as described in the protocol.

D. Printer

The receipt printer has a very simple interface in terms of communication. It uses a form of ASCII UART with a baud rate of 19200. General-purpose computers typically have very good support for serial ASCII interfaces through tty devices and the Raspberry Pi is no exception. With some configuration of the Raspberry Pi OS, it is possible to connect the GPIO UART pins directly to tty files. In our case we connect the printer to the /dev/ttyAMA0 serial device.

This means that in software, sending messages to the printer simply consists of opening the tty file and appending text to it.

That being said, there is a more complicated protocol built on top of the ASCII UART channel. In this protocol, basic text is printed directly to the printer, but through the use of escape

sequences it is also possible to instruct the printer to modify text properties, print barcodes and QR codes, and even display generic bitmaps. We intend to use the generic bitmaps in order to put the suits on our cards.

E. Keyboard

Since we are using an off the shelf USB keyboard, we are able to use Raspberry Pi OS's USB keyboard drivers and configuration. Similar to the receipt printer, the keyboard will appear as a tty device in the file /dev/ttyUSB0.

In order to read continuously from the keyboard, we'll simply have a thread with the file open listening continuously and sending the lines that it reads back to the Device Supervisor for transmission.

F. Camera

The Raspberry Pi camera is the most complicated device by far, but has the best driver support. Raspberry Pi OS has a kernel module called bcm2835-v4l2 which allows us to set up several capture modes with a daemonized process. The capture mode that we will use is periodic capture to file. Once we start the daemon, whenever we want the up to date image, we just read from the file we specified in the process launch.

G. YOLO

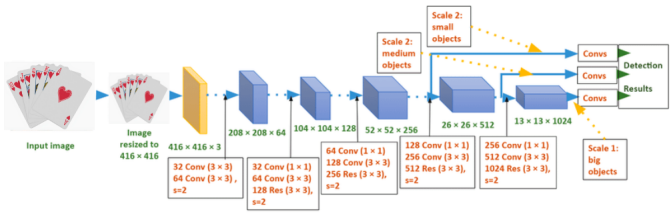

Fig. 5. YOLO Architecture

The YOLO algorithm will be applied on the real time video received from the Raspberry Pi camera module. The algorithm has a base network speed of 45 frames per second and when it is optimized, it can reach up to 150 frames per second. With real time streaming, the video can be processed with a latency of under 25 milliseconds. This algorithm takes the images from the video, resizes it to be smaller, and then splits it into an nxn grid which make up the bounding boxes. This algorithm then utilizes a single network to perform the actual detection. Within each box, our algorithm will determine if the center of the object is in the grid cell. If that is true, then that specific grid cell will work on detecting what the object is. The actual network will have 24 convolutional layers, 2 fully connected layers. a 1x1 reduction layer, followed by a 3x3 convolutional layer [1]. We will be utilizing linear activation functions within the layers and using the sum of squared errors to perform the computations within the layers.

H. LCD Screen

The LCD screen has a pretty complicated interface. It is a 11-pin GPIO bus. On top of 8 pins for data transmission, there is also a r/w pin and register select pin. The last pin is used for an operation-enable signal to tell the device to interpret the bus as a command.

The interface I have described is the interface to the ST7066 chip. Internally this chip interprets the 11-pin bus and actually lights up the pixels in the LCD screen. This chip has the same interface as a much more popular chip called the Hitachi HD44780, which has become the standard for LCD interface.

There are a number of open source kernel modules that work with the HD44780 interface. One which looks promising is called lcdi2c. This would allow us to operate the device through the file /sys/class/alphalcd/lcdi2c.

VII. Test, Verification and Validation

When testing our design implementation, we will be employing a mix of both unit and integration testing, verification. and validation. We chose to split up the testing as such because it is important to test each of the individual components as we go to make sure that they all work as expected, and we also need to make sure that when they are connected, all the various integrations perform as expected.

A. Unit Testing, Verification, Validation

The first unit that we will be testing is the thermal printer. As described by our design requirements, we will be testing to ensure that our printer can print out 3.25" x 2.25" cards with the accurate corresponding suit and number in a maximum of 1.5 seconds. We will have the code for each card predetermined, so when we send a certain card to the printer system, it should be able to output the exact card with the given specifications under the specific time restraints. The reason for these values is because the width of the printer is 2.25" and we need to fit the cardstock, so the length of 3.25" will be slightly shorter than that. The frequency of the thermal printer is approximately 2.35"/second, so it can print approximately one card every 1.5 seconds. The next test will focus on the camera with the connected computer vision ML model. Because the selected YOLO algorithm has a processing latency of under 25 milliseconds and a video detection of about 10 FPS for a non GPU device, we created a high estimate of approximately 35 milliseconds and hope that when a new card is placed in the vision area, the system should be able to know what the suit and rank of the card is under those 35 milliseconds. Next, the keyboard will also be tested to ensure that when the user enters in an input, it is properly received and buffer in under 10 milliseconds. This metric is the minimum value such that the human user does not experience any lag from their point of view which would be an inconvenience, so the testing will ensure that the latency is at peak performance so that users do not have to wait for their inputs to show up on the screen. Likewise, testing for the LCD screen will include measuring how long it takes for the screen to display text and special characters like suits. We hope to be able to display the text in under 0.1 milliseconds. This is because for the particular screen we have selected, the frequency of reading data from the RAM is approximately 37 microseconds and the frequency of writing data to the RAM is

also approximately 37 microseconds. Together, this will be approximately 74 microseconds, so an estimate of 1 millisecond will account for these 74 microseconds with a lag and also meet our design latency requirements of game state update speeds and requirements. Another unit test is the testing of the implementation of the game logic and rules for the three games we have selected to implement: Go Fish, Euchre, and Rummy. This testing will be more logic based and can contain user testing and the user of our test suites to ensure that all of the rules are properly implemented within the system and it understands all of the smaller protocols such as how many cards to pass out, how turns work, what inputs to take, and more. Finally, the last unit test will be in regards to EC2 and networking. We plan on implementing concurrency and logic tests for our EC2 instance and ensuring that we can send multiple signals to our EC2 from the various games without having any signal issues.

B. Integration Testing, Verification, Validation

Next, we will also be focusing on testing the integration of all of our individual devices. The first integration test will be in regards to the software device-level supervisor. Because of our use-case requirements of multiplayer support for up to 5 players per game and the ability to have concurrent games going on at the same time, it is crucial that our servers are able to properly interrupt signals from peripherals in a timely manner without dropping or disrupting any of them. This is important, because we need to make sure that we have all of the necessary and relevant information for each of the games, and this can lead to logic problems if any of the signals are dropped or taken from other games going on at the same time. The next integration test is with the keyboard and screen latency. As mentioned in the design requirements, due to the latency necessity of humans to process visual information with no lag, we are aiming for a latency of approximately 20 milliseconds. So for testing, we will ensure that keypresses performed by the user on the 10 key keyboard show up on the screen within the expected 20 milliseconds, which according to the latency requirements from our unit testing for the individual components is very reasonable because it accounts for the time in between to send the data from the keyboard to the LCD screen. Finally, the last integration test we will be performing is testing the network protocol between the server and device. The device supervisor should be able to send game state update messages to the server in a timely manner and the server can reconstruct a matching local game state. The reverse is also true here and the server can send commands to the device which are again serviced in a timely manner. This is to ensure that the communication between the devices is efficient and can keep up with the flow of the game.

VIII. Project Management

In order to manage this project efficiently, we have created a project schedule for the entire semester with goals, tasks, and deadlines so that we can stay accountable and ensure that the project is completed in a timely manner. Based on the skills of the three individual team members, we have also split up this project into various tasks and then assigned them to the appropriate member based on their specialties. Finally, we have also created a bill of materials with budget and risk mitigation plans to ensure that we are able to feasibly build this project and have a working end result at the end of the allocated time period.

A. Schedule

The projected schedule for our project's milestones and deadlines is displayed in Fig. 3. The overall project has essentially been split up into different tasks with three distinct paths allocated on the schedule for the three members. The schedule goes up until the middle of April when the project is due. Each of the 23 tasks has at least one person working on it and is organized in such a way that tasks that depend on others are scheduled so that all dependencies are met prior to starting it.

B. Team Member Responsibilities

As shown on the schedule, there are three main paths for the team members to take on. The first path focuses on the machine learning model with reading papers, comparing models, locally creating the computer vision, and writing the test driven development for the computer vision. Rachel will have the primary responsibility of handling the machine learning model and hooking up the computer vision to the device for this project because of her minor in machine learning and particular interest in the field. Her secondary responsibility will be helping the software track as necessary when the other members are running into problems or if she completes her parts early. The second path described in the schedule consists of more hardware components which will include defining a protocol, building a mock server, a CV debug dashboard, a peripheral dashboard, and performing test driven development on the keyboard, camera, LCD screen, and the thermal printer. This role will be the primary responsibility of Miya, who is concentrating in hardware in her degree. Her secondary responsibility will be helping Mason with the software path which is described next. The final path will include defining the protocol, building a mock device, writing test suites for Go Fish, Rummy, and Euchre, and performing test driven development on all three games. Mason's primary responsibility will be this due to his concentration in software and specialties in systems. His secondary responsibilities will include helping Miya with the hardware track as needed. All three members will then come together to work together and integrate all three of the games once the rest of the work is done and then will also work on optimizing the system as time permits.

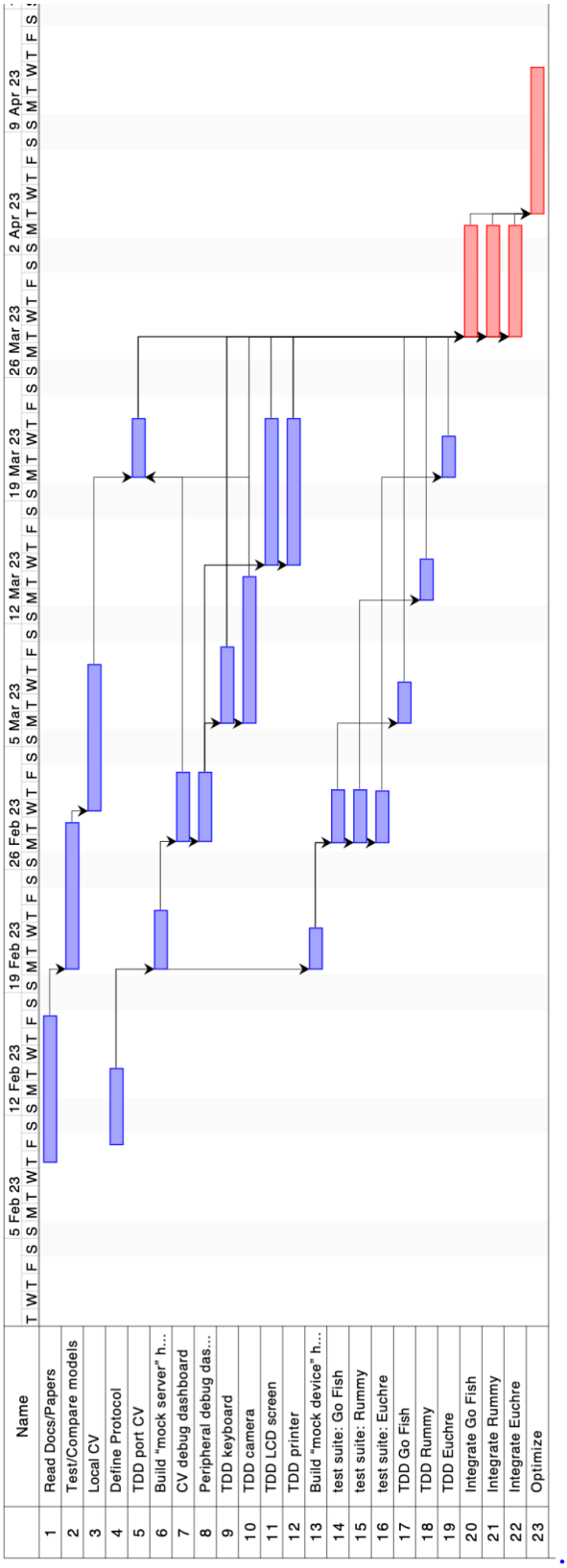

Fig. 6. Schedule with milestones and team responsibilities

C. *Bill of Materials and Budget*

The forecasted bill of materials and budget can be seen at the end of the report on Table II.

D. *Risk Mitigation Plans*

A potential risk we have here is that no one in the team has ever worked on a real time object detection. However, image detection with pictures has been performed before with simple computer vision. Although this is unknown, the YOLO algorithm is very well documented so should be simple enough to implement. However, if this does end up being a problem, a way to mitigate this risk is just actively taking individual screenshot images every couple of seconds, and then treating that as an image and feeding that into our network alternatively.

IX. Related Work

Our product is unique in many ways. One similar idea is that there are multiple online card game platforms. These offer a multitude of various games that people can play with their friends online but none offer a way to play with physical cards remotely. Additionally, there is a patent out there for a system that plays card games remotely that performs a very similar functionality as our but uses mechatronics instead of a thermal printer. This patent has been used multiple times for various card dispensers, blackjack and poker apparatuses, and more. However, our work differs from these because we focus on three games that have not been implemented yet: Rummy, GoFish, and Euchre.

X. Summary

Overall, our system allows for people to play cards with their friends and family remotely while still maintaining the feeling of playing physical cards. This will continue to bring people together even when separated by great distances. The thermal printer provides any card to the user and the design of the keyboard and LCD allows for the users to easily and efficiently input whatever they need to and witness it appear on the LCD screen. Additionally, the implementation of the various game logics, allows for the users to play a variety of games similar to how they would play them in real life. We will be facing multiple upcoming challenges in our implementation and in meeting the use-case and design requirements. Our next steps in the schedule include implementing the computer vision locally as well as performing test driven development on the various components such as the keyboard, camera, LCD screen, and printer. Additionally, we will need to be building the test suites for the various games. These will all be challenging, especially integrating the various pieces together. However, as long as our design is aligned with our implementation, this should not be a problem and soon people will be able to play cards remotely.

Glossary of Acronyms

YOLO – You Only Look Once

References

[1] Wang, Chien-Yao & Bochkovskiy, Alexey & Liao, Hong-yuan. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. 10.48550/arXiv.2207.02696.

[2] Jain A, Bansal R, Kumar A, Singh KD. (2015). A comparative study of visual and auditory reaction times on the basis of gender and physical activity levels of medical first year students. Int J Appl Basic Med Res. 2015 May-Aug;5(2):124-7. doi: 10.4103/2229-516X.157168. PMID: 26097821; PMCID: PMC4456887.

[3] Deber J, Jota R, Forlines C, Wigdor D. (2015). How Much Faster is Fast Enough?: User Perception of Latency & Latency Improvements in Direct and Indirect Touch. 978-1-4503-3145-6

Description	Model #	Manufacturer	Quantity	Cost @	Total
LCD MOD 160DIG 40X4 TRANSFLCT WH	NHD-0440WH-ATFH-JT#-ND	Newhaven Display Intl	1	$31.96	$31.96
RaspberryPi Camera	Module V1	RaspberryPi	2	$0	$0
USB Numeric Keypad Numpad	NK895	NOOX	1	$8.99	$8.99
128 Pieces Mini Transparent Plastic Clear Clips	B08RZ7NPT5	GALIREN	1	$7.92	$7.92
100 piece Cardstock	MaxGear-Printable-Business-Cards-100pc	MaxGear.LLC	1	$5.99	$5.99
Mini Thermal Receipt Printer	A2 Micro Pannel Thermal Printer	Cashino	1	$49.95	$49.95
RaspberryPi 3	A+	RaspberryPi	1	$0	$0
Uncut Matboard, 24" x 36" 4-ply matboard	Item # 10471916	Studio Décor®	1	$7.99	$7.99
Amazon EC2 Instance	Red Hat Enterprise Linux 8.0 (HVM)	Amazon Web Services	1	$0.156/hour	$314.50
				Grand Total:	$427.30

TABLE II. BILL OF MATERIALS