

Gaming Wizard

A Smart Table for Tabletop Gaming

Senior Design 1

Fall 2019



Group 30

Gabriel Holguin

Computer Engineering

Daniel Kalley

Computer Engineering

Erica Lindbeck

Electrical Engineering

Logan Taylor

Electrical Engineering

Table of Contents

1.0 Executive Summary	1
2.0 Problem Overview	2
3.0 Project Requirements	5
3.1 Marketing Requirements.....	5
3.2 Engineering Requirements.....	6
3.3 Standards.....	7
3.3.1 Hardware Standards.....	8
3.3.2 Standards for Software Development	9
3.3.3 App Accessibility Standards	12
3.4 Realistic Design Constraints	14
3.4.1 Economic and Time Constraints.....	15
3.4.2 Environmental, Social, and Political Constraints	17
3.4.3 Ethical, Health, and Safety Constraints	18
3.4.4 Manufacturability and Sustainability Constraints.....	20
4.0 Previous Projects.....	22
4.1 Multi-Touch Poker Table.....	22
4.2 Smart Table.....	24
4.3 Magic Frame: Turn Everything into a Touch Area	24
4.4 Similar Mobile Apps.....	25
5.0 Hardware Design	26
5.1 Multi-Touch Surface.....	26
5.1.1 Available Touch Detection Techniques.....	26
5.1.2 Touch Detection Scheme Selection	29
5.1.3 Touch and Display Surface Materials.....	29
5.1.4 Illumination.....	30
5.1.5 Infrared Camera	31
5.1.6 Display Method.....	32
5.1.7 Touch Surface Prototyping	34
5.2 Table	35
5.3 Microcontroller	39
5.3.1 ATmega328.....	39
5.3.2 MSP430FR6989.....	39
5.3.3 ATmega2560.....	40

5.3.4 Microcontroller Comparison.....	40
5.3.5 Uploading Bootloader to MCU.....	41
5.3.6 Communicating with the MCU.....	41
5.3.7 Standalone MCU Schematic.....	41
5.4 Cooling System.....	42
5.4.1 Temperature-Based Control System.....	43
5.4.2 Prototype of Temperature Control Subsystem.....	44
5.5 Timer.....	46
5.6 Brightness Adjustment.....	47
5.7 Special Effect Lighting.....	48
5.7.1 TLC5940.....	49
5.7.2 WS2812B.....	49
5.7.3 Led Effects Hardware and Software Design.....	49
5.7.4 Prototype of LED Effects System.....	50
5.8 Sound Design.....	53
5.9 Power Supply.....	53
5.9.1 Voltage Regulator.....	53
5.9.2 Voltage Regulator Design.....	54
5.10 PCB Design.....	55
5.11 Serial Communication Protocols.....	57
5.11.1 UART.....	57
5.11.2 I ² C.....	58
5.11.3 SPI.....	59
5.11.4 Summary.....	59
5.12 Wireless Communication Protocols.....	60
5.12.1 Bluetooth 5.0.....	60
5.12.2 Wi-Fi IEEE 802.11ac.....	61
5.12.3 Wireless Technology Comparison.....	63
6.0 Software Design.....	63
6.1 Software Overview.....	63
6.2 Object and Touch Detection.....	64
6.2.1 Framework Discussion.....	64
6.2.2 TUIO.....	66
6.2.3 Object Detector Class.....	67

6.3 Mobile App	69
6.3.1 Android OS Overview	73
6.3.2 Apple iOS Overview	74
6.3.3 Financial Costs	75
6.3.4 Developmental Tool Differences	75
6.3.5 OS Selection	76
6.3.6 App User Interface Framework	77
6.3.7 App User Interface Design	79
6.4 Game Software	80
6.4.1 Game Software Endpoints	80
6.4.2 Game Software Class Descriptions	81
6.4.3 Programming Language Selection	95
6.4.4 Multithreading	97
6.5 Wi-Fi Direct Interface	98
6.6 Windows API	100
6.6.1 Application with our Project	102
6.7 GitHub	105
7.0 Test Plan	106
7.1 Hardware Testing	106
7.1.1 IR Camera	106
7.1.2 IR LEDs	106
7.1.3 Surface Material	106
7.1.4 Power Supply	106
7.1.5 Light Sensor	107
7.1.6 Timer	107
7.1.7 Speakers	107
7.1.8 Accent Lights	107
7.1.9 Temperature Sensor and Fans	107
7.1.10 Wireless Communications	108
7.1.11 Table	108
7.2 Software Testing	108
7.2.1 Mobile App Testing	108
7.2.3 Communication of App to Game Software Testing	108
7.2.4 Object Detection Testing	109

7.2.5 Save and Load Game Data Testing.....	110
7.2.6 Display Testing	110
7.3 System Testing.....	110
7.3.1 Game Options	111
7.3.2 Player Character Options	111
7.3.3 Map Options.....	111
7.3.4 Game Master Options	112
7.3.5 Game Simulation	112
8.0 Budget.....	113
9.0 Milestones.....	115
Appendices.....	i
Appendix A: Copyright Permissions	i
Game Logo and Backgrounds.....	i
Figure 19	i
Figure 20.....	i
Figure 21	ii
Figure 22	ii
Figure 23	ii
Figure 30.....	iii
Figure 31	iii
Figure 32.....	iv
Works Cited	v

1.0 Executive Summary

Tabletop gaming has experienced a surge in popularity thanks to the rise of several popular web series featuring gameplay and more widespread references in traditional media. With this increase in popularity there is a corresponding increase in demand for new materials to game with, both in terms of game content such as new options for player characters, and gaming accessories such as dice and figurines. We plan to meet some of this demand with an adaptable system that will enhance or simplify existing game mechanics and provide a variety of new features that could be used to improve immersion in a wide variety of roleplaying games.

The system begins with a 38" x 30" x 36" table on which games can be played. The table itself will display the game in simulation of traditional mediums of gameplay, and the surface of the table will facilitate interaction with the imaginary world of the game through touch inputs. This functionality will be implemented using a rear diffused illumination touch screen made of clear acrylic, embedded in the top of the table, with a diffusive material attached. In comparison to many previously developed smart tables, this method will allow users to generate touch inputs using objects such as figures of characters, in addition to their fingers. The touch screen is illuminated from below in the visual spectrum by a projector, allowing any necessary game maps and features to be displayed. Additionally, the surface is illuminated by an array of infrared LEDs to generate a heat map of touch inputs which will be translated into coordinates by blob-detecting software and game software subsystems.

Additionally, several bookkeeping elements of gameplay, such as rolling dice and calculating distances on maps, will be taken care of automatically by the game software hosted by a player's pc, and a smart phone app which will turn players' phones into character controllers using a Wi-Fi Direct connection. Thus, time spent on the entertaining aspects of the game rather than performing tedious calculations can be maximized. To simplify performing these calculations across many games, the player app will track relevant information relating to each player's character such as current location, health, movement speed, and abilities, and save them between sessions. Similar apps can already be used to track many of these features, but our app will be able to track additional pieces of information and integrate them more seamlessly into gameplay.

To enhance immersion in roleplaying tabletop games, we add special effect functionality. Speakers embedded in the table will play sound effects for specific actions, and LED lighting around the edges of the table will light to denote the use of magic or other special abilities. If a character's choice of action causes changes in their environment, the map displayed on the table will show the affected area.

This report details the process of designing the Gaming Wizard system, accounting for budgetary and time restrictions as the project is funded solely by group members and intended to be assembled in under 5 months. Special consideration is given to guaranteeing that the system is compatible with a wide variety of current and future games, as well as incorporating standards to interface with the smart phones and computers that must be used to run the Gaming Wizard software. During the design stage we ensure that no components will be forced outside their specified operating ranges during intended operations, checking our power supply for compatibility and adding fans to keep the interior of the table cool. During the assembly and testing stages, we ensure that all electrical components are safely handled and securely attached to the table.

2.0 Problem Overview

In an open-ended tabletop role-playing game, each player controls at least one character and cooperates with other players in their party to complete a series of adventures known as a campaign. Traditionally, the physical representation of game settings consists of a gridded board and figures representing the landscape and characters, respectively. Paper character sheets are used to keep track of character abilities, and dice are used to determine the outcomes of events in the game. During gameplay, keeping track of the locations of characters and monsters on the board with respect to each other is vital to various mechanics such as combat, using magic, and interacting with the environment. However, problems arise such as remembering character locations between game sessions or when a piece falls over. Further, counting grid squares to determine distances each turn often causes delays in gameplay and can be a matter of contention when paths do not lie along grid lines.

Another issue that arises with tabletop gaming is the large amount of information that needs to be tracked. This includes things like character attributes, skills, equipment, and the amount and type of dice needed to be rolled for an interaction. These values evolve over the course of the game which normally requires a large amount of erasing and rewriting. Given that an average game involves four players and a game master, this information is tedious to track and slows down gameplay.

Our project seeks to solve these issues by using a “smart” game board that can recognize multiple character locations, represented by physical figures on the game board, track that information, and display the relevant information both on the board and in a phone application. Ideally when a character is moved on the board the application software will recognize this and ask for a confirmation. The application will save and remember character locations even after being closed which will greatly reduce the time it takes to reset a board between meetings.

Additionally, the application will track character information such as attributes and combat statistics. This will create a paperless game that removes the need for constant erasing and rewriting. Combining this with the location information will also greatly reduce the time it takes to perform combat interactions, as the board can automatically display the range of an ability or spell once it is selected by the user. Potentially, dice rolls will also be simulated in the software, removing the problems of needing a large surface to roll dice on and losing dice.

The proposed smart table will host the game from a user’s laptop and communicate with players’ phones using Wi-Fi. From their phones, players and game masters (GMs) will use a menu system to create characters, view and modify their stats, and trigger events on the table’s surface corresponding to character movements and abilities. Further, a character’s statistics will be saved on the corresponding player’s phone, to reuse the character on different maps that may be loaded onto the board for different scenarios. All character and digital token locations on the board will be saved to the GM’s phone or computer when the table is shut down, allowing the group to recreate the game board quickly for subsequent game sessions. The GM will have access to additional options such as map selection and enemy creation that should not be accessible to players. The current map along with additional relevant graphical information will be projected onto the tables surface from the interior of the table. The structure of this phone application/computer program/table system is given in Figure 1.

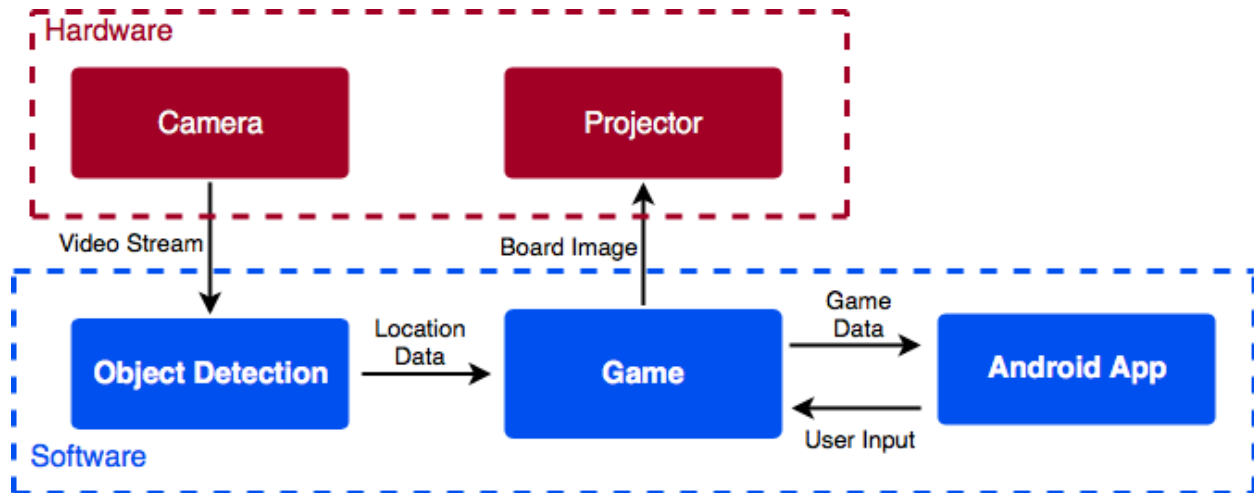


Figure 1: General Software Block Diagram

In order to track characters, figures on the table’s touch surface will be located by using Rear Diffused Illumination (Rear DI). In Rear DI, in addition to the image being displayed, infrared light illuminates the table’s surface from below and diffused for even coverage of the surface. The translucent material of the surface allows some light to be transmitted through the surface while some is reflected downwards. When an object is placed on or hovers closely above the table’s surface, the transmitted infrared light is reflected downwards by the object and creates a region of higher intensity infrared light, which can be detected by a camera under the surface. The generated heat map is used to determine object and touch locations. This method was selected based on location accuracy requirements, cost efficiency, and the need for object detection in addition to detection of fingers and specially designed styluses. Touch input will also be used to select locations for ability usage (e.g. the center of a spells areas of effect) and movement of digital tokens representing enemies or allied non-player characters.

The table will rely on a standard electrical outlet for power in order to remain turned on for lengthy game sessions. An automatic cooling system will trigger when the inside of the table, which will contain most of the electrical components, exceeds safe operating temperatures. This system will include of at least one fan and an associated temperature sensor. To ensure visibility, manual controls will be available to set the baseline brightness of the table’s image. Additionally, a light sensor will automatically adjust the brightness of the table when possible as the table’s environment brightens or dims.

To improve the quality of gameplay, there will be several additional features to the table. Indicator lights on the table will be used to denote player turn and/or ability effects. These lights will be able to emit various colors, with each player character corresponding to a unique color. A player’s color will also be used to mark their character’s location for the purposes or recreating a map layout at the start of a session. Further, speakers inside the table will allow the game master to play ambient music or special effects, controlled by the app on their phone. Finally, a programmable timer will be included for timed skill challenges and limiting decision-making time in combat. The incorporation of the many proposed features of the table is detailed in Figure 2.

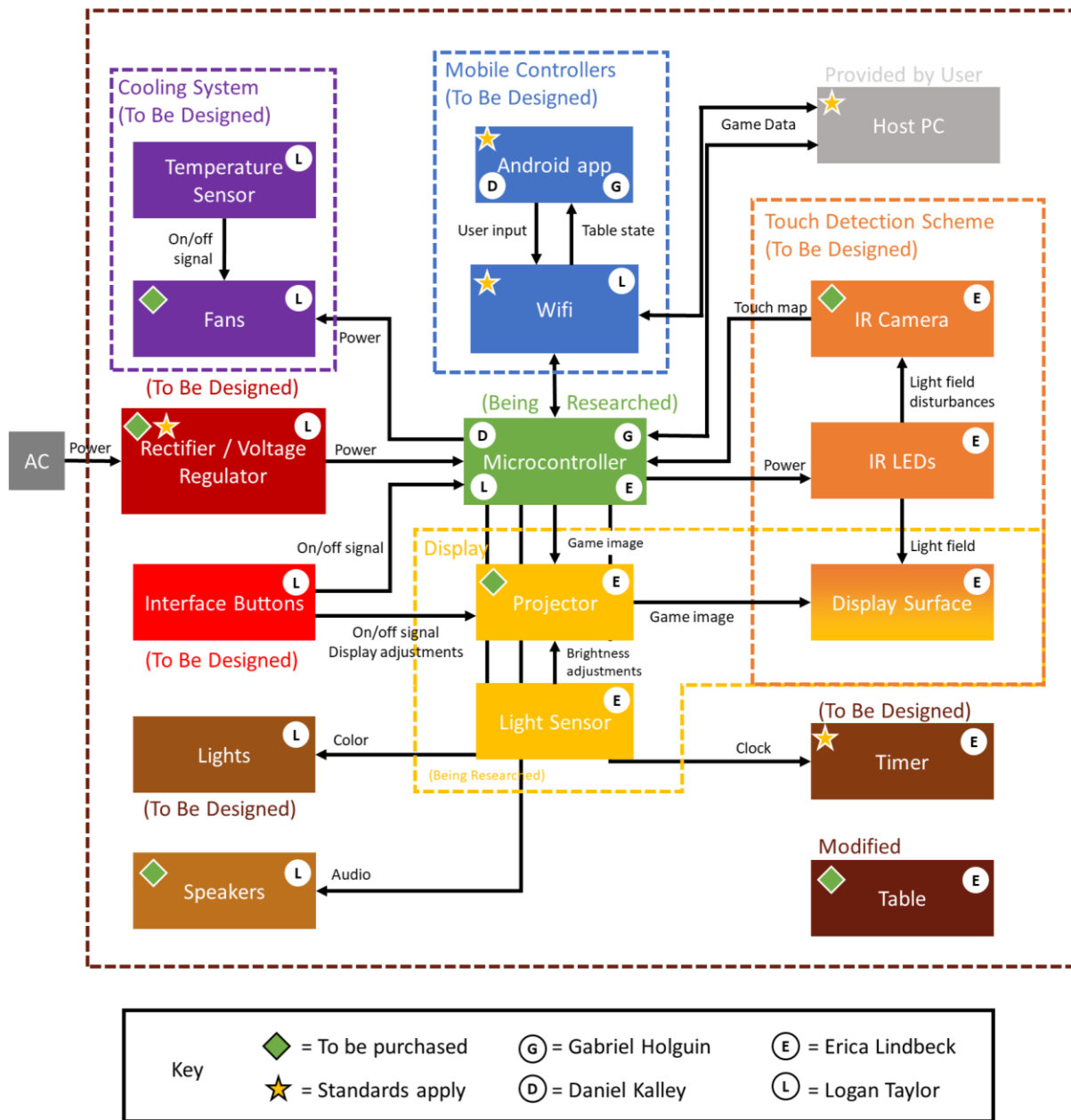


Figure 2: Hardware Block Diagram

3.0 Project Requirements

3.1 Marketing Requirements

As we design a smart table for tabletop gaming, we must consider what the most desirable characteristics of such a table would be, given a typical game. As a game usually includes several players and is run by a single game master (GM), support for several simultaneous users is necessary. During an encounter or while exploring a specific location, a map consisting of a top-top view of the environment, divided into squares which represent a 5' square in the environment with a 1" square on the map. This map should be sufficiently large to show a reasonable portion of the environment and contain all of the relevant characters. Setting up the map, placing characters, and looking up character details should be quick processes to avoid delays in gameplay. As each player's character is represented using a miniature figurine of the character with a 1" diameter base, the figure must be accurately located on the map to determine which objects, enemies, and allies are within range of the character's abilities (such as magic spells and weapons), as well as which locations (i.e. grid squares) the character can move to on their turn. A player should be able to easily tell where objects are on the map and distinguish features of the environment such as doors, levers, and walls in order to make decisions about which actions to take on their turn. Finally, the table must be affordable for a dedicated table group and so that the benefits it confers are worth the price. These desirable features are organized in Table 1 as they are the general characteristics which the table must have.

Table 1: Marketing Requirements

Desired Features	Reasoning
Supports Normal Game Groups	Most games are balanced to be played by 4 players with a game master. A larger or smaller group is possible, but the table should be able to support groups of typical size.
Easy to Use	If the table is significantly more complicated than traditional paper/dry erase maps and figurines, there is little to be gained by using it. Setting up a map and placing characters should be comparably quick, and interfaces should allow easy access to desired character and ability information.
Supports Full Length Games	A typical game session is 3-6 hours, though it may last longer. A map is usually not required for the entire session, so the table may not be necessary for the whole session but should be available for the entire duration, just ot be on the safe side.
Map Details Are Easily Distinguished	Map features such as landscape, locations of digital tokens, and areas of effect for character abilities should be easily distinguishable for effective gameplay
Character Locations and Touch Inputs Are Accurate	Locations of characters and choice of target when triggering an ability such as a spell should be accurate to the constraints of the game, which is played on a grid.
Low Cost	The table should be low enough cost that the benefits it brings to gameplay justify the expense, and maintenance (e.g. replacing projector bulbs) is not excessive.

3.2 Engineering Requirements

Engineering requirements of the table which will ensure that the table meets marketing requirements are given in Table 2. These requirements are chosen in compliance with the relevant standards given in Table 3, based on our chosen implementation using a Rear DI touch screen and a phone application. Some of these standards are detailed further in the following section. We will likely encounter more standards as we finalize our design and begin its implementation, but for now we believe our design will either satisfy standards or be modifiable to meet them.

Table 2: Engineering Constraints

Aspect	Constraint	Reasoning
Touch Surface Dimensions	20"-36" per side	Large enough play area without causing significant detection issues; 20" square game mats are the most common commercially available custom mapmaking product
Table Height	1.5'- 3.5'	Players are expected to sit around the table during gameplay
Maneuverability	Can be safely moved by 2 people	The table will have to be moved for demonstrations and in home without special equipment
Device Lifetime	≥ 3 years	Desirable lifetime given expected cost of development/replacement
Simultaneous Touches Detected	≥ 6	Support 4 player tokens + 2 touch points for user input
Simultaneous Mobile Controllers	≥ 5	Support at least 4 players and a game master
Mobile Controller Range	≥ 10'	Avoid disconnecting walking around a room
Mobile Controller Input Delay	≤ 1 s	Acceptable delay considering inherent wireless communication delays over a stable connection
Touch Input Delay	≤ 0.2 s	Acceptable delay for touch input detection to interact with the table in near-real time
Operating Temperature Inside Table	≤ 32°C	Safe operating temperature for most electronic devices/components
Time to Cool from Startup (assuming ambient temperature is within operating range, internal temperature at most 10°C higher than allowable)	15 minutes	Acceptable delay assuming ambient room temperature is within operating range, long enough for fans to significantly cool the interior of the table
Average Projector Bulb Change Time	≤ 6 minutes	Ease of replacement is necessary as projector bulb is an expected point of failure in the long term
Continuous Operation Time	≥ 6 hours	Enough for most game sessions
Display Resolution	≥ 720 x 720 ≥ 1080 x 1080 ideal	Possible with available projectors and provides reasonable image quality at expected display dimensions
Object Size for Detection	≥ 0.5" diameter	Must detect fingers and 1" diameter figure bases
Object Location Accuracy	≤ 0.5" from true location	Must be able to accurately place track objects/touches to 1" grid squares
Object Removal/Placement Detection	≥ 95%	Must reliably detect when a figure is moved for character tracking
Locations Tracked and Saved on Exit	≥ 20	Able to store location data for the next game session for 4 player characters and a reasonable number of virtual allies/enemies
Average Time to Set Up New Map	≤ 2 minutes	Acceptable set up time based on average time needed using a traditional map and figures

Table 3: Considered Standards for Engineering Requirements

Standard	Application
IPC-2221B	Generic Standard on Printed Board Design
NASA-STD-8739.3	Soldered Electrical Connections
IEEE 802.11-2016	IEEE Standard for Information technology--Telecommunications and information exchange between systems Local and metropolitan area networks. (Wi-Fi Standard)
IEEE 3007.3-2012	IEEE Recommended Practice for Electrical Safety in Industrial and Commercial Power Systems
CUI Power Supply Safety Standards	Standard for power supplies
Android Design Guidelines	Regulates Android applications
Google Java Style Guide	Non-mandatory standard for program formatting
IEEE 29119	Software testing

The relationships between a selected subset of engineering requirements and the marketing requirements are detailed in the House of Quality in Figure 3.

		Simultaneous touches	Simultaneous controllers	Image Resolution	Screen Size	Continuous Operating Time	Touch Input Delay	Average Time to Set Up New Map
		+	+	+	+	+	-	-
Supported group size	+	↑↑	↑↑		↑			
Ease of use	+				↑		↑	↑↑
Supported game duration	+					↑↑		
Detail distinguishability	+			↑↑	↓			↑
Touch input accuracy	+				↓		↑	↑
Cost	-			↓	↓		↓	
		≥ 6	> 5	≥ 1080x1080	2'-3' per side	≥ 6 hours	≤ 0.2 s	≤ 2 minutes

Figure 3: House of Quality for Selected Engineering Constraints

3.3 Standards

Standards define the characteristics of a product, process, or service. They simplify product development, reduce unnecessary duplication, lower costs, increase productivity, promote safety, and permit interchangeability, compatibility, and interoperability. Standard development

organizations such as IEEs facilitate the development and maintenance of official standards but some companies such as NASA choose to also follow their own standards.

3.3.1 Hardware Standards

Various standards are used as a guide for designing and testing the hardware of the device. Standards were selected based on relevance to design decisions such as safety, wireless technology, and soldering techniques. To reduce costs for the project most of the standards chosen are freely available.

3.3.1.1 CUI Power Supply Safety Standards

Devices used within the smart table must be powered with various voltages which presents a level of danger especially when dealing with the alternating current coming from the mains. Standards can be used to protect against fire and electric shock. The CUI power supply safety standards [1] aim to identify the major standards that relate to power supply safety including: IEC 60950-1 (safety of information technology equipment), IEC 62368-1 (audio/video, information and communication technology equipment), IEC 60601-1 (safety of medical electrical equipment), and IEC 61010-1 (safety of measurement, control and laboratory equipment).

Within the document three classes of equipment are defined based on how their power supplies are isolated from dangerous ac mains voltages. Class I devices achieve electric shock protection through basic insulation and protective earth grounding and have conductive parts with a hazardous voltage connected to a protective earth conductor in case of insulation failure. Class II devices use double or reinforced insulation and do not require a ground. Class III devices operate from a safety extra low voltage supply circuit where extra low voltage is defined as a voltage in a secondary circuit not exceeding 42.4Vac or 60Vdc which are the levels when a voltage is considered hazardous. The power supply, although connected from the wall outlet, only provides 12Vdc so this device is considered class III. The only hazardous voltage is coming from the wall but is protected through the use of insulation around the wires so this does not pose a problem.

3.3.1.2 IEEE 802.11-2016 (Wi-Fi)

IEEE 802.11 [2] is part of the IEEE 802 set of LAN protocols and specifies the set of media access control (MAC) and physical layer (PHY) protocols for implementing wireless local area network (WLAN) Wi-Fi computer communication. IEEE 802.11-2016 is the current version currently in publication which supersedes previous releases. It is a revision based on IEEE 802.11-2012 and incorporates 5 amendments (11ae, 11aa, 11ad, 11ac, 11af). The abstract of the standard states that the revision includes correction and clarifications to IEEE Std 802.11 for wireless local area networks as well as enhancements to the existing medium access control and physical layer functions. This project will be utilizing Wi-Fi direct which is called a direct link in the standard. The definition of a direct link is a bidirectional link from one quality-of-service (QoS) station to another QoS station operating in the same infrastructure QoS basic service set that does not pass through a QoS access point. Once a direct link has been set up, all frames between the two QoS STAs are exchanged directly.

3.3.1.3 NASA-STD-8739.3 Soldered Electrical Connections

It should first be stated that on October 17, 2011 NASA-STD 8739.3 [3] was replaced by J-STD-001E requirements for soldered electrical and electronic assemblies which is a joint industry standard. However, J-STD-001E is not free to access and the old NASA standard is adequate for

the scope of this project. There is no requirement to follow a specific standard for soldering, and it is simply used to gain a better understanding of actual soldering techniques used in industry.

For this project a large number of IR and RGB LEDs will need to be soldered to conductive wires and surface mount parts will need to be soldered to the PCB. Understanding the appearance of an adequate solder connection will be important to ensure strong connections so devices work as intended. Figures in the document show minimum and maximum acceptable amounts of solder for various connections as well as improper wire stripping and how solder connections look when insufficient or excessive heat is used. There is also a list of definitions that are useful when trying to understand the different terms used in soldering.

It is stated multiple times throughout the document to not use pressurized air to cool solder joints. The solder should be left to cool naturally at room temperature. Cooling the joint too quickly can cause it to become fragile or reduce the integrity of the connection. It is also stated that there should be no motion between conductors when solder is applied. This means that when soldering wire to LEDs they should be fixed in place. When it comes to the geometry of a solder joint the fillet should be smooth and concave. About using the correct amount of heat, it is stated that a cold connection will exhibit poor wetting and have a grayish and porous appearance. An overheated connection will have a rough surface and will be dull, chalky, grainy, porous, or pitted. These descriptions will be useful when inspecting the appearance of our solder joints.

3.3.2 Standards for Software Development

Though not as strictly adhered to as hardware standards where the misuse of a standard can cost the functionality of the project, there are some standards for the development of mobile applications that make readability and peer reviewing easier. Android OS has a few developmental standards for mobile app development, C++ has some of its own syntax and style standards, and Wi-Fi Direct has its own standards.

3.3.2.1 Android Mobile Application Standards

The two general purpose standards that Android OS developers have to deal with relate to UI design and quality of app. In the case of UI design, it is mostly up to the user's preference, but there are a few guidelines to follow that will help with human interactivity with the app. Style, layout, usability, components, and animations are the five main UI designing basics. For style, color must indicate which elements are interactive and the most important elements should stand out. All text must be legible on the selected background and a general color theme should be followed. For layout, the UI must be predictable in a way that humans can use with consistency. The UI must give some kind of feedback when an input action is performed by the user. In the case of usability, the developer must have a specific target audience in mind but also be user friendly enough with a developer-defined minimum level of technological understanding. Components deal with the layout of the UI in order for the app to be ergonomic on a smart phone and having related navigation bars is equally important. The bottom navigation is also a staple of any good Android OS application. Animations must show spatial and hierarchical relationships between elements and must keep focus on what's important. Animations should not bombard the user with stimulus that will confuse them.

Mobile application quality standards are more important than UI standards when it comes to publishing your app and Table 4 below shows us some of the most important standards apps should follow.

Table 4: Android Application Quality Standards

Area	Description
Standard design	The app must not redefine pre-built internal functions such as the back button
Navigation	Pressing the Home button at any point while the app is running must return the user to the Home screen
Permissions	The app should not request higher access to the user's personal data than the absolute minimum needed to run the app
Installation	If the app is larger than 10MB, it should support installation on a SD card and function the same as if it were installed on phone memory
Audio	While the screen is off or another app is running, audio must not be playing unless that is a feature of the app such as for a music player
App states	While the app is in the background, any service such as Bluetooth or Wi-Fi must halt unless it is understood as a core feature of the app
	When the user opens the app while the app is in the background, it should restore the user to the same point that it was left at before
Stability	No freezing, crashing, or interruptions of the same magnitude should occur when using the app
Performance	If more than two seconds have gone by while loading, some kind of feedback must be indicated to determine the status of still loading
Data	All data that is private must be stored on the app's internal storage and be inaccessible to the developer
Networking	SSL is used for all network traffic and the app must declare a network security configuration beforehand
Cryptography	No custom algorithms are allowed and platform-provided cryptography should be used instead
Content Policy	The app must adhere to Google Play Developer Content Policy [4] when dealing with copyright and other intellectual properties
App Details Page	High quality image must be used and the image must not contain any explicit graphics or resemble an advertisement
Testing	Follow Core Suite [5] testing methods for additional testing methods

Since we currently do not have any plans to commercialize our app, we wouldn't need to strictly adhere to these standards, but it is a good idea to use best practices any time for development. These standards can be our guidelines when developing our app so we can have the option of marketing our app in the future.

3.3.2.2 C++ Language Standards

Since similarly to Android app development there are no ISO developed standards for C++, we must look at what are considered standards and best practices by engineers. We will be taking our standards and styles from the Google C++ Style Guide [6]. This gives us a guide as to best practices for implementations into larger projects that we can pull from. Google uses C++ as their main programming language for open-source projects and as such has developed a style for cooperation between multiple engineers.

The reasons these standards exist in the first place is to succeed in a few goals laid out by Google. Firstly, all of the style rules should be enough of a significant net benefit when comparing the style with and without the rule. They must be useful enough to make thousands of engineers adhere to it without being too much of a hassle. Since many people after the initial writer of the code will work on, build on, or just reference the work, all code must be optimized for the reader instead of convenient for the writer. Because of this, it is important to be consistent when writing code and not change styles or syntax midway as the reader can get confused. Consistency is also important for trying to adhere to some general use syntax that the C++ community uses for problems that are common or some idioms.

Since C++ can be complex as a programming language, it is important to avoid dangerous constructs that can risk compromising program correctness. The average C++ programmer would find some of these tricky constructs hard to maintain since the initial programmer might not be there to explain what is going on. All of these rules are mostly in place for cooperation as no one person can do all the work in the world. This also applies to our group as we have multiple programmers working on the same project, it is important to keep consistent syntax between members. Some of the style and syntax choices are specified in their document are shown in

Table 5 and Table 6 below.

Table 5: C++ Coding Standards

Category	Standard
Header Files	Every .cc file should also have an associated .h file, with the exception of very small files that only have a main
	Headers should be self-contained meaning they should be able to compile on their own.
	To prevent multiple inclusions, all header files should have #define guards and should be able to guarantee uniqueness by basing the guard on the full path name
	#include any headers you may need instead of using forward declarations
	For ordering headers the order is related header -> C system headers -> C++ standard library headers -> other libraries' headers -> your project's headers. Each has a blank line in between the categories
Scoping	Initialize all variables in the declaration and only place variables in the narrowest scope possible
	Using static or global variables should be avoided if possible
Classes	Specify if the class is copyable, move-only, or neither
	Use classes for everything except for passive objects that carry data; then use structs.
	When using inheritance, make the class public
	Make constants public and all other classes' data members private
Functions	Declarations should be in order of access level. public -> protected -> private.
	The output of a function must have a return value or be provided through output parameters
	Functions should have a small focus to do only a few tasks per function call
	If a parameter is passed by lvalue then it must be labeled with a const before its name
	Function overloading is only used if the reader will be able to figure out the function's happenings without having to question which overload is occurring
Exceptions make the flow of programs difficult to evaluate so it's best not to use them	

Table 6: C++ Code Formatting Standards

Category	Standard
Formatting	The line length of code should not exceed 80 characters. 80 characters is roughly the right size to have the editor open on one side of the screen and have something else on the other
	Tabs should produce 2 spaces since tabs are arbitrary and can be different per computer
	Brackets should be consistently used either next to functions and such or on the line after them, not both types
Naming Conventions	Give variables names that have clear meanings to help future readers
	File names should be all lowercase with no spaces; underscores are fine
	Type names start with a capital letter
	Variable names are all lowercase with no spaces
	Constants should start with a “k” and every word in them should start with an uppercase letter
	Function names have a capital letter for each name
Comments	The code should be written in such a way with naming convention that comments should be sparse throughout the code
	Keep consistency and only use // or /* */ throughout code. It’s better not to use both
	It is better to not literally describe what code does unless the code is nonobvious
	Always have file comments at the beginning of a file that describes its contents and licensing

Even if some of these standards are not followed, consistency is key when writing good readable code.

3.3.2.3 Wi-Fi Direct Standards

There are not many standards or best practices for Wi-Fi Direct [7] as it is a fairly new technology, but there are some common operations that must be performed for the connection to function correctly. The setup for the direct connection must be made with either a push-button or PIN-based method. The security used for the connection is WPA2, which may be used with Android OS. To use Wi-Fi direct, the connection to a hotspot network need not be built in.

3.3.3 App Accessibility Standards

The Web Accessibility Initiative (WAI) has produced guidelines for ensuring that web content and mobile applications are accessible to individuals with disabilities such as visual impairment, dyslexia, and epilepsy. These are the Web Content Accessibility Guidelines (WCAG) [8]. Some of these guidelines cannot be reasonably accommodated by our mobile application, but those which are relevant will be taken into consideration. The guidelines are either A, AA, or AAA quality, with A being the minimum requirements and AAA the strictest. As this app is intended for casual games and not an essential interface for any other services, and we do not have plans to commercialize our product, we are able to disregard the stricter AA and AAA requirements

without significant concern, although in a few cases when their features would in general be beneficial to our users they may still be considered. In some cases, we may disregard the A level requirements as well, as this is effectively a prototype design with a niche intended user base, and the A level requirements cannot be easily accommodated in our time frame. For example, many guidelines specify that some aspect of a page can be programmatically determined, that is, that other programs can extract that information from our app (e.g. that the app is in English). However, we do not expect a user to need to extract this information as it typically is only useful to individuals who are so visually impaired that the entire gaming table would be useless to them, and thus they would not be in our user base. In other cases, this information is extractable by default when using any standard development environment. Either way, we may not be able to make a conformance claim for our end product but should be able to reasonably accommodate most of our expected users.

3.3.3.1 Perceivability

There are several relevant guidelines regarding the perceivability of information in our application, detailed in Table 7.

Table 7: Perceivability Guidelines

Aspect	Requirement	Accommodation
Colors	Color should not be the only visual means of conveying a piece of information	All information will be conveyed through explicit wording, although color-coding may also be used
Contrast	Text should have a contrast ratio of at least 4.5:1 for small and medium sized text, and at least 3:1 for large text	All text shall be black and all backgrounds sufficiently light
Images of Text	Images of text should not be used in place of text unless the text is part of a logo of brand name	The only image of text will be in the logo on the splash screen or in user uploaded images beyond our control

Other perceivability guidelines ensure that sound is not disruptive to users and that any images or sounds which contain essential information have corresponding transcriptions to allow those with audio or visual processing difficulties to have an alternative in a medium they can understand. However, any images (aside from our logo and background textures) and sounds which will be used in the app and game will be uploaded by users, and we may reasonably assume that a user will not upload an image or sound effect that they cannot distinguish.

3.3.3.2 Operability

There are several relevant guidelines regarding the operability of our application, detailed in Table 8.

Table 8: Navigation Guidelines

Aspect	Requirement	Accommodation
Titles	Each screen has a title describing its topic or purpose	All menus will have a top bar labeling the current menu
Link Purpose	The purpose of any link or button can be determined from the label text alone, except where the purpose is ambiguous or general	All buttons should have straightforward labels (max 5 words) which are understandable by experienced tabletop players
Focus Visible	Any keyboard interface has a mode of operation where the keyboard focus indicator is visible.	Any keyboard interface will either expand to fill the screen or highlight the typing area

Other operability guidelines are related to keyboard usage and reducing the risk of inducing seizures with flashing lights. However, we will not be developing our own keyboard, so the keyboard interface will be determined by the user’s phone’s settings and thus we will not have any control that would allow us to meet these guidelines. Further, we will have no flashing lights or images in our app, and any changes in display will be prompted by the user’s touch input, so there should be no concern about the timing of flashes. The only exception would be the user changing menus too quickly, but we can prevent this with debounce intervals and the assumption that if a user knows they are prone to seizures it is reasonable to assume they would not cause flashing by tapping quickly anyway.

3.3.3.3 Understandability

There are several relevant guidelines which ensure information in our application and its functions are understandable to users, detailed in Table 9.

Table 9: Understandability Guidelines

Aspect	Requirement	Accommodation
Abbreviations	A mechanism is available for identifying the expanded form or meaning of abbreviations is available	The expansion of any abbreviation used in a label/button will be provided when the label/button is pressed
Labels / Instructions	Labels or instructions are provided when content requires user input	Any input will be labelled in accordance with the standard labels of tabletop games when available, and instructions provided for table-specific inputs
Error Identification	If an input error is automatically detected, the item that is in error is identified and the error is described to the user in text.	When an invalid input is given, such as an alphabetic character in a context where only numeric input is reasonable, an informative error box will be displayed
Help	Context-sensitive help is available	When possible, an informative pop-up corresponding to the currently considered action will be displayed

Other understandability guidelines are related to ensuring the layout of the app does not change significantly in an unpredictable fashion that would disorient the users. As any menu will either expand or replace the screen only when the labeled button to access the menu is pressed, and a button to return to the previous menu will be available, there should be no such issues.

3.4 Realistic Design Constraints

There are multiple different realistic design constraints which we had to be aware of and take into account during each stage of our project. These design constraints are different from the standards that we have described before. This project is considered our prototype for our design so if we did not include all standards for the implementation than we would could still make a functioning project. Those standards are meant for cooperation between different engineering teams and for implementation, it meant easier compatibility with hardware components. While the hardware component standards would be a pain to ignore as compatibility would be up to our team to get

working correctly, we could still manage. Software standards on the other hand can still be mostly worked around if necessary since we do not plan to make this product commercially available.

Standards make cooperation easier between other groups, but constraints are absolutes. Constraints must be taken into account and while there can be some wiggle room in some cases, failing to take these into account will mean the failure of a project. While failing to use these realistic design constraints will mean failure, the understanding of these constraints can help with design criteria, quality of project, and decision making. Classifying certain criteria can help us to understand which constraints apply to each level of the engineering requirements. These constraints can be broken down into a few general topics that will be covered in the following sections such as:

- Economic
- Time Management
- Environmental
- Social
- Political
- Ethical
- Health
- Safety
- Manufacturability
- Sustainability

Generally, design constraints are issued by the customer, by a development organization, or are external regulations such as laws. They are mostly created to meet certain criteria or to make sure the product meets specific requirements that will allow it to go commercial. Since we are making our own project, we must act the part of the customer when thinking of realistic design constraints. There are the constraints that do not change, such as safety or ethical, whether a solution is home-built or company developed. Constraints limit the solution for a problem therefore before making allowances and changing a solution to fit a constraint it is imperative to make sure that the constraint needs is followed during the design process. The addition of even one more constraint can drastically change a solution and require many more hours of creative and difficult engineering to adhere to. Thus it is important to classify the design constraints into the categories mentioned above to make sure that they are necessary and applicable to our current project.

3.4.1 Economic and Time Constraints

With no financial sponsorship for our project, all financial costs and burdens fall upon our team of four people. When we discussed what our project budget should be, we decided that it should ideally cost under \$600, and absolutely not cost more than \$1000 for designing, testing, and final implementation. Since we are making this project ourselves, the designing phase should not constitute more than \$100 of our budget. This includes having to cut specific materials such as the acrylic or other unforeseen manipulations of materials that require specific tools that we do not have access to. Testing is estimated to cost around \$200 where we would test more specific parts to see if they work with our project. We can also purchase some redundant inexpensive parts since shipping would cost more if a part were to break or malfunction on us. The shipping cost for most technologies used is a where a large part of our funds went towards. We projected that the final project, if commercially made available, could be made with the budget of around \$450 with the acquisition of cheaper parts. Mass production companies can acquire cheaper parts, compared to

what we have to work with, because they purchase items in bulk. Since they can acquire items through bulk purchase, much of the shipping costs that we can pay would be cut down significantly.

Since we have no sponsor, we need to try and get cheaper products either through deals or accepting slightly worse quality products to use in our project. The projector is definitely our biggest overall expenditure thus we must try to minimize the expense while not compromising on the performance of the projector. We will secure a projector by mainly looking at deals during black Friday and cyber Monday. By comparing projectors that will meet required specifications for our project we can diminish the economic burden placed on us. There are many different projectors that we have looked at as stated in this document, but since this is the most expensive overall part of the project, if the projector fails to meet specifications and performance results, the expenditure would significantly increase as another projector will need to be acquired.

Subsequently, since customers do not have to cover the financial cost of designing and testing, we assume the final product will be cheaper to mass produce. Our expenditures for designing the project are based around materials and tools that we realistically do not have cheap access to. The acrylic must be cut with a specific tool to accurately and safely change the dimensions in a way that the acrylic will not be damaged. Acrylic is not delicate but can crack easily if an amateur attempted to cut a piece thus we must expend some funds for professionals to cut the pieces. While this seems like an unnecessary expenditure, the cost for cracking an acrylic glass can severely negatively impact our financial situation. A company that will mass produce these tables will already have pre-made acrylic glass to fit each table, or have a cheaper way to cut the glass through a company contract.

The financial cost of testing would also not be pushed onto the customers as testing should be a large expense at the development phase and then less burdensome when trying to update a product. Most of our testing budget must go into testing to see if certain products work with the overall design we are envisioning. Items like LEDs, speakers, IR Cameras, microcontrollers, and cooling fans all fall into this category. While we did do extensive research on products to ensure performance and cost analysis, compatibility may be a problem as we cannot always foresee how each product will interact with each other in a real world test. While these products do have recommended compatibility charts, even then problems can occur with trying to connect to the software.

The time given to our group to design, build, test, and finalize this project was two contiguous semesters of college. For us, the time schedule began around late August 2019 and will have to be completed by April 2020. The project must be fully functional and meet engineering specifications described at that time. The table must be able to track individual character pieces through object detection, and the screen must be a touch display and be able to accurately display the game data within certain specification described in this document. The timeline of goals and milestones has been created to help us follow through with the time constraint given and be able to deliver a working product.

While it does seem as though we have 8 months from August 2019 through April 2020 to get the product meeting engineering specifications, the time we have to work on it is not so easily defined.

A company that would be working on a project such as this would likely dedicate personnel to focus on development of their one project and nothing else, but we do not have the time to do so. Each of us in our Senior Design group has other obligations that we must attend to apart from working on this project. From work, to other classes, or other responsibilities, we all have duties that take up time from working solely on our project.

3.4.2 Environmental, Social, and Political Constraints

Apart from the financial and time constraints placed on us during this project, there are many factors such as environmental, social, and political constraints that we must contend with. Since our smart gaming table is planned for use by people who normally play board games inside, we are designing and prototyping our table to be safely used indoors. As we are building this prototype in Florida, the conditions outside are less than favorable for electronics. The humidity is high and can damage electronics that are not specifically designed to resist such extreme conditions. The heat is also a problem as we do not want our electronics running too hot since they could potentially be damaged. Since we will be testing this product indoors, with air-conditioning, we expect the ambient temperature to reach no higher than 85° Fahrenheit (29° Celsius). Since we are indoors, the humidity will also be expected to be no higher than 20%. Florida sometimes reaches outside temperatures upwards of 95° F (35°C) and humidity levels around 74%. Although the prototype can be modified at a later time to protect against the harsh Florida climate, it would also require more resources to produce.

Our smart gaming table's electronics will not be weather protected since we must design those parts to work outside by ourselves. The printed circuit board especially is at risk for the humidity and high temperatures of Florida. It will be inside the table ideally, but will have connections and microcontroller exposed to the open air. As described in this document, we will have a cooling system with fans inside of the table that will keep components at a safe temperature. While the fans will help minimize the risk of overheating, the humidity levels can make it so water can accrue inside the fan and drip onto sensitive components. Our speakers' output can become muddled due to the humidity and the blob detection may become compromised due to dripping water on the acrylic sheet. While being in an air-conditioned building does remove some of the constraints of prototyping in Florida, if the climate outside is severely harsh then indoor building conditions will also be affected.

Social constraints for our current project mainly deal with consumer wants and copyright laws. As social constraints can be based on cultural preferences, whether a product will actually be used by anyone must be taken into consideration. Even if a new and innovative product solves a problem better, if no one takes an interest in it then it will become a failure. For our project, we must look at certain problems that people who play tabletop games face and if they would be willing to spend around \$500 on our product. One of the main problems that new players and sometimes veterans face is trying to keep track of all the information that goes into the game, from their own character stats, abilities, and items to visualization of the game world. The information of individual players and their attributes will be easily tracked and managed thanks to our companion app, severely lessening the load of mentally tracking such data. The app will also help lower the learning curve for newer players for a more welcoming experience. We believe the price is something a group of players can split to lessen individual expense since the table will presumably be used every week for a gaming session.

Since we are designing our own circuit board layout and are not claiming the electrical components to be manufactured by us, we safely do not violate any copyright laws on that end. Our object detection software has been labeled to be open source and is able to be used with individual projects. With the exception of free use UI libraries, all other software will be written by our group. The main constraint for copyright comes from Wizards of the Coast, the company who owns D&D, as we must make sure that our implementation follows their fair use statement. D&D uses the Open Game License (OGL) for how to deal with individual modification of their proprietary game. OGL is a public copyright license that tabletop games can use so that individuals can modify and redistribute their game mechanics. The OGL copyright for D&D encompasses the free use of the material referenced here: [9]. This document is their System Reference Document (SRD) that is used for the OGL allowed to individuals. We must not use any game information that is not in the SRD, even if we do not plan to commercialize our current prototype, it is better to follow these constraints to be protected under copyright laws.

As we are not designing anything that is used or potentially going to be used by the military or government, the political constraints for our smart gaming table are minimal. We do not predict that our table will have any impact on any foreign government either. The only use for this smart gaming table should be entertainment and thus there should be no political constraints associated with the project.

3.4.3 Ethical, Health, and Safety Constraints

Our prototype must adhere to ethical, health, and safety constraints that could be problems when dealing with a prototype. Since our prototype is our initial design and some of our group members' first time working with soldering electronics, we must be careful when dealing with these components. For this project, the main constraints in this section would be dealing with the storing and collection of player data, the ignorance of electrical safety, and the dangers of open circuit equipment being used by people other than our group as a consumer might not know the safe way to use any open electrical equipment.

Since we plan to make our game able to be saved and picked up again at a later time, for a game that is played in for multiple play sessions, we must ensure that we cannot see any personal user data. Since there are many ways for personal data to be stolen or even misused by the creators of an application, it is important to keep to the highest levels of ethical standards when dealing with personal data. A database created by developers should not have access to high-risk personal information such as credit card numbers or passwords. All sensitive data should be encrypted and sent over secure network lines so people cannot steal the information while in transit. If a developer is using a database, sensitive data must also be stored as encrypted on the database so anyone with access to the database cannot see the sensitive data.

Our group is planning to use Wi-Fi Direct in our connection to our mobile application which will not require the use of a web database. All storage of the game, assets, and player information will be directly on the user's personal computer. The game executable will create folders for individual player information that can be reused next game session. Using this method means that we do not need for players to create accounts as they will be able to select their in-game character from a list

stored on the main computer. Since no sensitive personal data is being sent through Wi-Fi direct then there is no need to encrypt data access from our mobile application to the host computer.

Our principal health constraint for consumer use in this project is how we deal with our ignorance when dealing with new electrical components that we have never worked with before. The equipment that we are using is not high voltage or high current as we will be using an already made wall adapter to supply our power. The wall adapter helps us circumvent the dangerous engineering requirement of working with potentially deadly levels of electricity. By using a preexisting wall adapter, we also make sure that any users of our product do not seriously harm themselves if we were to make an adapter. Any electronics must also be protected or concealed enough to the point where people cannot accidentally bump their legs underneath the table on an exposed wire.

Constraints on substance safety must also be followed to try and minimize the risk to ourselves. While the exposure of wires and electrical equipment is a safety hazard for consumers, we must also protect ourselves when dealing with the equipment necessary for our project. As a step for compliance with safety constraints we must use materials that are in compliance with RoHS standards. RoHS stands for Restriction of Hazardous Substances and there are plenty of electronics bound by them. The RoHS bans the excessive use of certain materials that can cause serious harm towards humans. The main elements that the RoHS considers dangerous substances that might be used in our project and their maximum levels are described in Table 10 below where ppm is parts per million.

Table 10: RoHS Restricted Substances

Element	Maximum levels
Cadmium (Cd)	< 100 ppm
Lead (Pb)	< 1000 ppm
Mercury (Hg)	< 1000 ppm
Hexavalent Chromium (Cr (VI))	< 1000 ppm
Polybrominated Biphenyls (PBB)	< 1000 ppm
Polybrominated Diphenyl Ethers (PBDE)	< 1000 ppm
Bis(2-Ethylhexyl) phthalate (DEHP)	< 1000 ppm
Benzyl butyl phthalate (BBP)	< 1000 ppm
Dibutyl phthalate (DBP)	< 1000 ppm
Diisobutyl phthalate (DIBP)	< 1000 ppm

While we may not be exposed to relatively large quantities of these substances, the smaller amounts can still be harmful if misused. Cadmium is mainly exposed to manufacturing and construction workers, but can also be encountered by people who recycle electronics. While we will not handle damaged electronics parts, we must throw away old or failed parts in appropriate recycling containers to ensure cadmium is not grouped with regular trash which would endanger workers. Electrical engineers often use lead to solder electrical equipment and must be careful with skin contact and contact with open sores. Lead poisoning can lead to anemia and kidney damage if in high enough dosage and can lead to dizziness and weakness for lower exposure levels. Soldering can already be dangerous without the mentally addling effects of lead that we must look out for. Low levels of mercury can be found in equipment we are using such as switches or

batteries. Similar to cadmium, Cr (VI) is usually exposed to people during welding but can show up when electronics get to unsafe hot temperatures. PBB and PBDE are common in electrical products that need to be flame-retardant. The last four elements, DEHP, BBP, DBP, and DIBP, are all used as insulation plasticizers that make polymers more flexible and increase thermoplasticity.

Apart from elements and substances that can be harmful when developing our prototype we must consider the harm that the electrical components can bring. Speakers will be able to play music and sound effects chosen by the lead player of a game, but we must put safe limits on sound output or make sure that the speakers cannot produce high decibel (dB) levels that cause permanent hearing loss. The safe noise level to minimize hearing loss over 8 hours is 85dB; 85 dB is roughly equivalent to the sound of a boiler room. Hearing loss significantly increases as sound levels go upwards of 85dB, so to be safe we must constrain our sound level output to never go above 80dB.

The electronics should not be able to run to high enough temperatures as the components and people can come to harm thus we are going to use a cooling fan to reduce the temperature. Cooling fans for electrical components run at around 4000 rpm (rotations per minute). Considering the plastic fan material and the speed, the possibility of permanent injury with this type of fan is low, but safety precautions must still be taken into account. Hair and other electrical components such as wires can get caught in fan which could jam the device or simply break it outright. Our table is made out of wood, but temperatures of our electronics should be nowhere near enough to burn or blacken the wood as most wood start burning at around 300° C, so handling of the table will be safe as long as we keep in mind the weight and size when transporting it.

While working with light sources and infrared technology, it is important to consider the effect that these technologies have with the human body. Most projectors give out a light with a power output of around 6kW (kilowatts), which can easily damage the eyes of someone looking at the projector. Looking directly at the device is not a major concern, but accidentally pointing the projector at someone else can cause serious harm. While working on our table, we must also be careful when looking from top to bottom if we remove the acrylic for whatever reason. After researching IR light, we found that since IR light produces lower energy photons than visible light, it is not harmful to humans with the IR LEDs we are using.

3.4.4 Manufacturability and Sustainability Constraints

Manufacturability deals with designing a product in such a way that parts can easily be made or acquired and assembled into the final product. The manufacturability constraints are mainly adhered for products that are mass-produced, but it is also important when trying to reproduce other people's work and simplicity of design. For constraints dealing with our group, manufacturability means that we must think about our part acquisition and assembly of our table. The processes that manufacturing consists of after designing is finished are:

- Parts Receiving and Processing
- Subassembly
- Final Assembly
- Inspections and Testing
- Packaging and Shipping

The company must buy or make all of the required parts which are then cataloged and made ready for initial assemblies. To make this process easier, we must try and minimize the amount of parts required for our project which will help reduce this step. The reduction of parts will also help lower prices as less parts means that a manufacturer can reduce shipping costs of bulk purchases. The next step is subassembly wherein the initial parts are assembled into smaller completed parts such as a microcontroller connecting to a cooling fan. This assembly is not the final product, but it is also easier to test some functions at this step before final assembly. During final assembly, all of the smaller assembled pieces are fitted and connected together to form the final product. The final assembly in a production line will be similar to our own final prototype, so making our final prototype easier to be assembled by using easily manageable parts will allow for an easier assembly process in the future.

Testing all of the functions of the product is the next step which allows checking for any issues made with assembly or defective parts. Inspections make sure the products are adhering to health, safety, and other constraints previously mentioned. Packaging and shipping must also be taken into consideration when building any device nowadays. Most products bought by consumers are shipped to homes all over the world, so the product must be robust enough to survive the trip. This means utilizing parts that are sturdy enough to survive some jostling or encasing the more delicate components into protective cases that shield them through shipping and can be left on during normal use. A table would have problems during shipping as the legs of the table are easily in danger of being damaged so packaging must fill the shipping box with soft material and handle with care. To make shipping easier, we must make sure all of our components will not fall off or out of the table.

Sustainability constraints address the need to use components that meet the needs of the device, but can still be easily found currently and in the future. If the resources are extremely scarce or if parts of the product are damaged and the entire device must be replaced then the product does not have good sustainability. Some of the parts that we are using such as our IR camera, butchered from a PlayStation Eye, may not be found in the future. Even though these exact parts may not be found, our smart gaming table's design will still work if another IR camera is used. The cost of older electrical devices is always decreasing as technology gets better so finding an IR camera in the future will actually be easier than it is currently. Companies can also manufacture their own projector or since the table does not use all of the functions of a regular projector, they can make a device specific for the needs of the table.

Our smart table will have all replaceable parts that will be easily interchanged if a part malfunctions. The LEDs, fans, projector, and speakers are all easily acquired although some parts are not inexpensive. The only product that will be difficult for the consumer to acquire in the future will be our own printed circuit board, but since the documentation of the circuit board is including with our document, the board can be printed again. Since these parts are easy to acquire for students like us then companies will have an easier time obtaining these products with a bulk purchase for selling replacement parts.

4.0 Previous Projects

There are a few similar smart gaming tables that have been done before that we can learn techniques and mistakes from. While their game is not the same, much of the technology, or at least their concepts, are similar enough to what we have available to use with our project. There has been much progress with technology after the time these products were built so we should have an easier time acquiring better technology for better prices than what these smart tables used. The technology that has most advanced in recent years for this type of smart table project would be the projector and the app development environment for phones. The cost of projectors has significantly been reduced ever since mini-projectors and home projectors have become more popular and affordable to the general public. Apps are also easier than ever to develop with tutorials and extended libraries from community users and from the standard libraries themselves.

We didn't want to focus too much on smart tables from companies with much more resources than we could possibly come up with for this project, so our research for similar products are built by small groups only. Two of these similar smart gaming projects, Multi-Touch Poker Table and Smart Table, are from UCF students that were in Senior Design like our group is now. We thought that we had the most to learn from fellow students in similar situations to us. They may have had more personal funding or prior knowledge, so it won't be a perfect comparison to what we are trying to make presently. The other project is from a DIY enthusiast that had some prior experience with hardware and software called Magic Frame: Turn Everything into a Touch Area.

4.1 Multi-Touch Poker Table

When researching techniques for multi-touch sensing we came across the Multi-Touch Poker Table [10] by UCF Alumnus Christopher Herod, Nathaniel Boucher, and Raeginald Timones done for Senior Design of Fall 2009. Their object was to create a low-cost smart gaming table to be able to play Texas Hold'em Poker on with a companion app to go along with it. Although not exactly what we are trying to design and create, since our project will have object detection for character figures, we figured it was worth taking a look at their design and implementation for a smart table. Similar to their project, we are planning to have a touch screen display, but we are also trying to create object detection for ours with open source software for blob detection. While the idea of only having digital tokens for our game characters did come up for discussion, we believed that with more modern technology that we could make it work. For our smart table, we will be making an Android OS companion app similar to their iOS app for individual players. Table 11 compares what the Multi-Touch Poker Table team used for their design in comparison to the technology we are using in our own smart table. While just comparing parts is not ideal, we can get a sense of the starting point of each smart gaming table and their designs.

Table 11: Smart Table Feature Comparison

Feature	Multi-Touch Poker Table	Smart Gaming Table (ours)
Multi-Touch Display	Allows for multiple finger touches at once and dragging digital tokens on screen	Will also allow these same features with the addition of object detection and tracking of character game figures on top of the screen
Projector	Minimum screen resolution of 800x600 with adaptive light output depending on ambience in room	Minimum resolution of 720x720 since our game boards will be square. Light sensor to automatically adjust the brightness of the screen or of the projector's output
IR Camera	Minimum resolution of 640x480 with a frame rate of at least 30FPS	Same resolution but minimum FPS of 60
Computer	Their project has a dedicated PC built inside of their table that runs everything for the game	Our plan is to have one of our home PCs separate from the table since the object detection will most likely take more processing power than we can afford for dedicated hardware with our budget
Table	Cooling system and phone charging station with USB 2.0 inside	Ours will also have dedicated fans for the microprocessor and components for the table. It will also include speakers, LEDs, timers, and buttons for interactions and in-game immersion.
Wireless Device Communication	IEEE 802.11g standard Wi-Fi to connect to iPhones and have input and output data from them	Wi-Fi Direct communication with Android OS device for communication with the game

While our idea for having a smart gaming table which uses a touch screen is nothing novel, as seen with this previous project, the idea of having physical object interactions combined with the touch screen will pose a new challenge. Even with open-source software for the object detection, we are still the ones setting up the hardware and trying to make sense of what the objects detected mean for our game. While we could try for a projector with better resolution, since in comparison to the resolution of the Multi-Touch Poker Table there is not much of a difference, there are expenses and over-engineering issues to consider. The cost of projectors jumps up significantly with each significant increase in image resolution. We decided that since our game is less reliant on image quality and 720 x 720 is already high quality, we can save money by getting a cheaper product than what was available 10 years ago.

There is no need for a better IR camera for this level of project since it already does the job it's meant to for a cheap price and getting a better one will not necessarily have any effect on the final product. The Multi-Touch Poker Table group also decided to have a dedicated PC built into their table which made it seem much more elegant but increases component cost for prototyping. If we were to find out that the PC provided had a problem running the software, we could easily try one of our other group members' PCs instead of swapping components in the table. If this were a product to be mass produced or even produced by demand then having PC components be part of the table would be required, but since the smart table we are trying to build would be our prototype, we decided that a separate computer would be easier to test with.

4.2 Smart Table

The Smart Table [11] is another smart gaming table designed for Senior Design by UCF Alumnus Chris Rodrigue, Phillip Murphy, Jonathan Lundstrom, and Ryan Mulvaney in Spring 2017. This project demonstrates some of the functional aspects of the software that we want to have available in our gaming table. This project is more of a utility or smart feature add-on to an already existing product. These type of products for general use can be compared to fridges with touch screens or Wi-Fi thermometers. They take an existing product that has little to know interaction with other electronics around it and give it options for interactivity and smart interactions between devices. While their main focus was not building up the smart gaming table with one game in mind, the techniques used can help us understand our own project better. Key interactions that we can learn from them are how they got sensors and LEDs to work with and affect the output display image.

The Smart Table group had a number of features including timekeeping, environmental sensing, current weather status, and some simpler games like Snake. They used Bluetooth for their communication option for input and selection of what to display on screen. Their project works similarly to a 16x2 LED display that people often use with DIY projects. The screen of their table is composed of a grid type pattern of square blocks that has a RGB LED backboard to it to display the image. While the technique of LED might be different to the projector idea we have for our project, we also need to create individual squares for our game. Similarly each of our squares will represent an area for the game matrix to function on. These blocks of ours will have more functionality than display like it did for the Smart Table project as it will also determine the characters options at that block.

4.3 Magic Frame: Turn Everything into a Touch Area

This DIY project to turn any flat surface into a touch screen was done by user Jean Perardel at hackaday.io. [12] While not entirely reliable, it is a way of touch screen that we could get to work with object detection in a grid as well. The surface that he used is a flat screen TV that would be easier to use and develop for than a projector and creating our own touch screen with acrylic. The technology used for this project is based on having IR LEDs on one side of the table emit light and having light triangulation with IR receptors on the other side of the table. The IR receptors check for the light from the LEDs and if it cannot see any light then an object is blocking the path. With enough of these IR LEDs and receptors you can triangulate the location of an object.

The problem and the ultimate reason that we decided to seek alternative solutions to this is the problem with light triangulation as reliable method of object detection. In tabletop games multiple characters are on the screen at the same time, so detection of specific characters is an issue with this method. There can be multiple characters physically on the screen that could block the position of another character if they are close by. Similarly, for our grid game, each character needs to stay

in their individual grid block, so we need to keep a record of which character is in each block. This would be difficult to program as the squares are quite small and the light triangulation might mistake an object as being in the wrong square leading to incorrect game data.

4.4 Similar Mobile Apps

The market for tabletop companion mobile applications is widespread and influential to the design and implementation of our own application. These companion applications are usually only for storing your personal character information or to help lighten the load for remembering people and places in the game. The Game Master can store his planned fights or attributes of the monsters for the fight in some of these apps or create their own with specialized monster creators. On iOS and Android OS some basic apps to help keep track of such information are available, such as Fight Club 5th Edition and Fifth Edition Character Sheet respectively. There are plenty of other examples since tabletop games are becoming increasingly popular, but these emphasize what most of the companion apps are capable of. Table 12 provides a comparison of these apps' features with our intended app's functionality.

Table 12: Mobile App Feature Comparison

	Features of Current Companion Apps	Features Our Mobile App Will Have
Character Sheet	All Companion apps have a dedicated page or two that details a player's information such as attributes, skills, and abilities	We will also have a tab solely for this character sheet as it is the information players most often use in game
Attacks	More advanced companion applications bring in data of weapons and skills from reference books	We will have a template for attacks where players specify range, attribute, die roll, and radius if applicable
Inventory	Basic applications just have a tab with an expanding list as to what is in a player's inventory	Our mobile app will have predefined places to put a few items such as money and then a template to write down other inventory items.
Enemy Characters	Some apps have a compendium to look up certain monsters created by others	We can create templates for monsters so we can have more customizable gameplay
Game Master Settings	The applications are mostly made for players and as such Game Master UI settings are limited to compendiums and note taking	We will have a different UI for the Game Master so they can control the flow of the game and select whose turn it is

Because of the plethora of abilities, weapons, creatures, and character features available in tabletop games, as well as the continual expansions of many games, we believe it is better to have a template for users to input data rather than forming a hard-coded database from existing sources. We also have these templates because we must make certain structured data available to the smart gaming table for tracking game characters. Having a template allows more flexibility in what people want the game to be like. The Game Master will be given a different user interface than players so they can control multiple monsters on the smart gaming table at once. Most apps that are available are commonly informational, but we desire to develop a true companion app that will enhance and simplify the experience.

5.0 Hardware Design

5.1 Multi-Touch Surface

An essential component of the table is a multi-touch surface capable of accurately detecting and tracking simultaneous touch inputs provided by fingertips and small objects. The inputs resulting from objects will be used to monitor the positions of characters in the game, while the fingertip inputs will direct the character's actions. Although a resistive touchscreen may be able to meet the detection requirements, and a capacitive touchscreen could detect objects if specialized contact surfaces were created to adhere to the bases of the objects, both methods leave the sensing apparatus more vulnerable to excessive impacts and require more advanced manufacturing techniques than an infrared projection-based touchscreen. Therefore, we have chosen to focus on light-based techniques for our design.

5.1.1 Available Touch Detection Techniques

Upon review of available multi-touch technology, only three methods supported object detection in addition to fingertip detection, and thus were considered for use in the table. Rear Diffused Illumination (Rear DI), Diffused Surface Illumination (DSI), and Light Triangulation (LT) are detailed in the subsequent sections.

5.1.1.1 Rear Diffused Illumination (Rear DI)

Rear Diffused Illumination (Rear DI) [13] uses a clear material (glass or acrylic) with a diffusive material adhered to either the top or bottom as the touch surface. The surface is embedded as the top of a closed box, and infrared illuminators, in the form of LEDs or similar emitters, project light onto the surface from below, illuminating the surface as evenly as possible. The diffusive material allows partial transmission of the IR light, while the rest is reflected downwards. The touch of a user's finger, or an object placed on the surface, reflects the otherwise transmitted light downwards as well, creating a bright spot known as a "blob" of infrared light, which can be seen up by an IR camera at the bottom of the box. Tracking software can then process the blob-filled image to create a map of touch input locations for use by the rest of the game software. This process is illustrated from a side view in Figure 4.

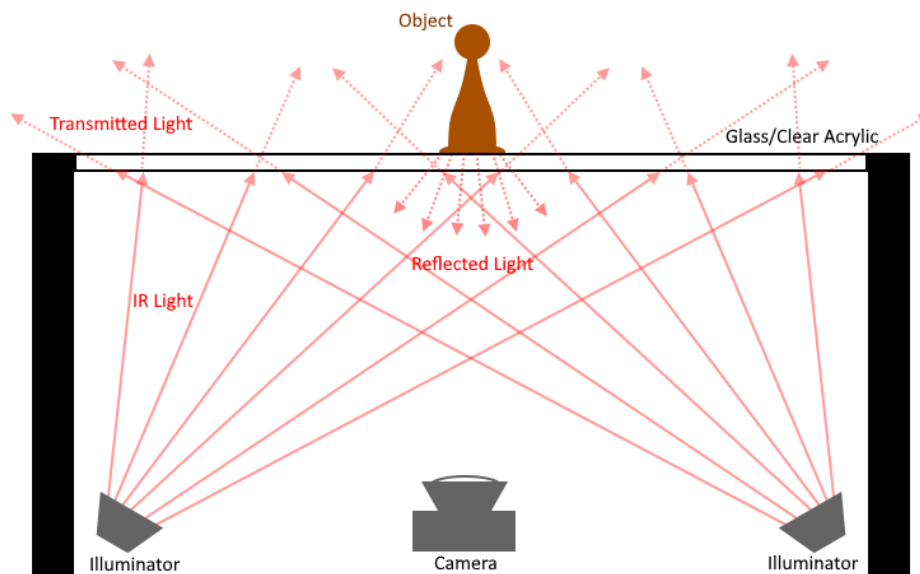


Figure 4: Rear Diffused Illumination Technique

Rear DI's main advantages lie in the low price and availability of the required materials, and the simplicity of the illumination method. Additionally, there is well-tested open-source software available for blob detection and tracking that would simplify the development of the touch/object tracking functionality. Disadvantages lie in the need for a closed-box projection system, difficulties in achieving even illumination and thus equal detection rates across the surface, the possibility of hot spots (false inputs), and the constant low-level reflection decreasing the contrast of blobs, making detection difficult.

Although a related process in the form of Front DI, in which illuminators are placed above the screen and inputs are detected by the shadows cast under the touchscreen by an object's presence, is also capable of multi-touch object detection, the inherent misdetections and dead zones caused by hands reaching across the table to move objects or touch specific locations render it impractical for our purposes.

5.1.1.2 Diffused Surface Illumination (DSI)

In Diffused Surface Illumination (DSI) [14], similarly to Rear DI, a specialized acrylic surface with a diffusive material adhered to either the top or bottom as the touch surface. This special acrylic is embedded with minute reflective particles. Then, when the acrylic is illuminated from an edge by IR LEDs, IR light is internally reflected by the particles and interfaces of the acrylic with air, diffusing evenly through the material out to a distance determined by the quality of the acrylic used (at most about 1.5 feet from an illuminated edge). When a finger or object is placed on the top of the surface, IR light is deflected downwards. As in Rear DI, this creates a visual blob which can be detected by an IR camera below the screen. This process is illustrated from a side view in Figure 5.

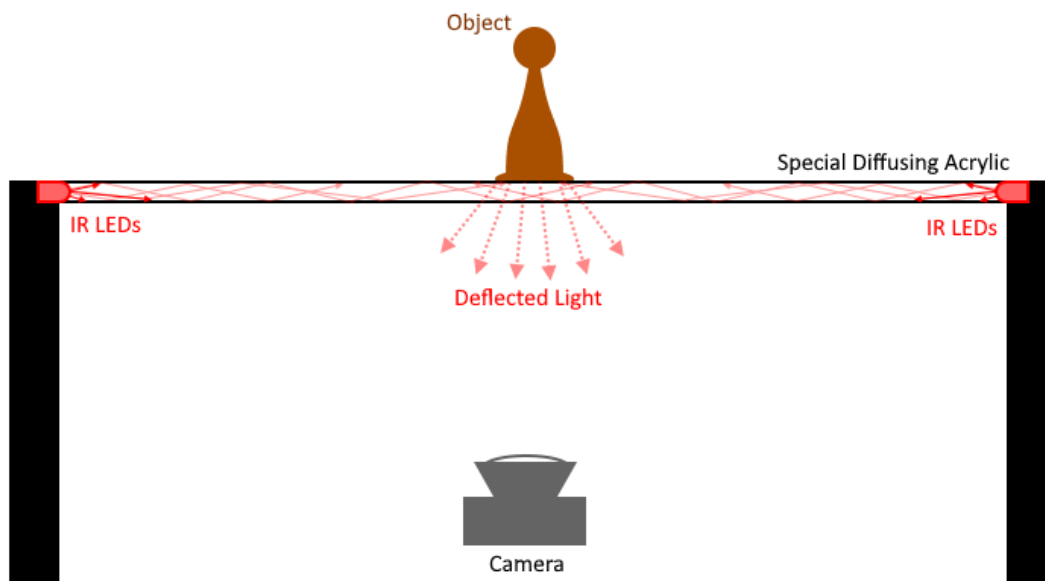


Figure 5: Diffused Surface Illumination Technique

DSI's main advantage is the uniform illumination of the surface, which makes tuning the detection software simpler and more reliable. Further, the same open-source software available for blob detection for Rear DI can also be used for DSI. Disadvantages lie in a similar need for a closed-box projection system, the low level of deflected light resulting in low-contrast blobs, the limited

size based on diffusion distance of the acrylic, and the high expense of the specialized acrylic. Depending on the diffusion distance required, and thus the quality of the acrylic, the cost can be between 5 and 40 times that of a glass or clear acrylic sheet of the same size, which can easily increase the cost of the surface by hundreds of dollars given the 4 – 9 square feet required.

5.1.1.3 Light Triangulation (LT)

In contrast to the previous two techniques, light triangulation [12] has no requirements for the material of the touch surface, as all hardware other than the visual display is necessarily placed above the screen. A strip of IR LEDs is placed just above the edge of the active area of the surface and shine across to a strip of sensors on the opposite side of the table. Each LED is lit briefly in sequence, and the sensors which can detect each LED are recorded. Any object or finger on the table will block the light between only a few combinations of LEDs and sensors and knowing those combinations we can extrapolate a region in which the blocking object must lie based on the geometry of the array. With high enough numbers of LEDs and sensors, high resolutions can be achieved. This process is illustrated from a top-down view in Figure 6.

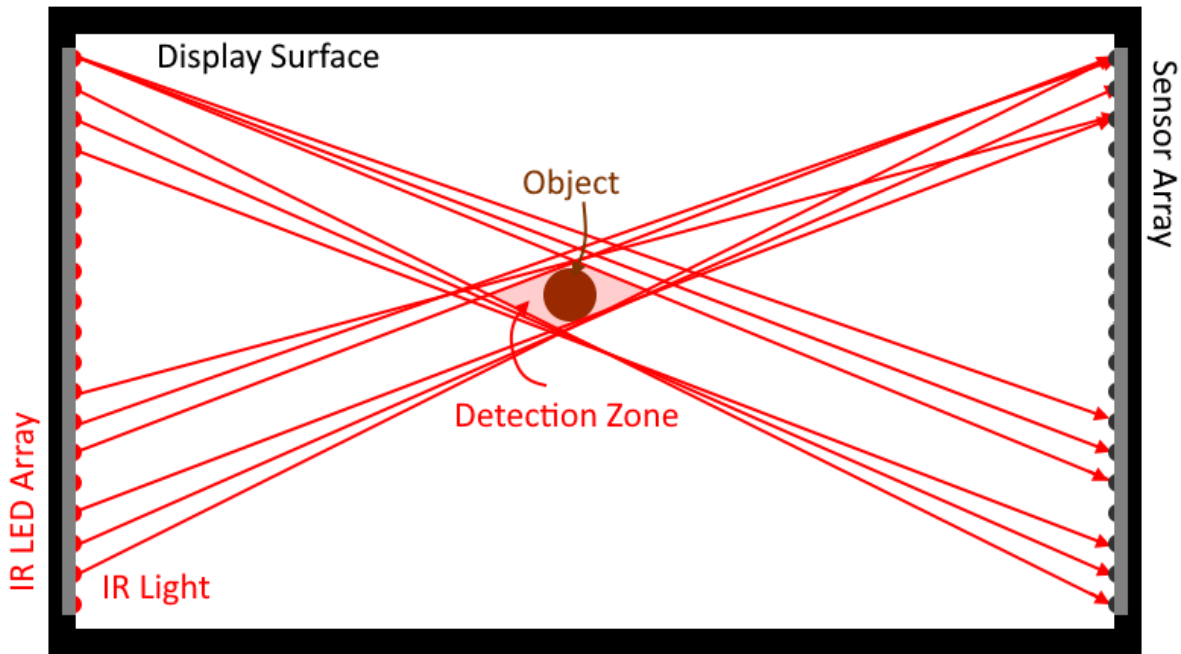


Figure 6: Light Triangulation Technique

LT's main advantage lies in the lack of limitations of visual display options, which would allow the use of a cheap television a projector, and no closed box requirement. However, there are many disadvantages to the technique. Locations near the edge of the screen may not have high detection accuracy, or be complete dead zones, unless the LED and sensor arrays are extended beyond the limits of the active touch-surface area and/or modified to unusual geometries. Objects which lie close together in groups may become indistinguishable as they block the same LED-sensor pairs. Additionally, the standard game figures which are intended to be used with the table typically have short, flat bases, and unusual shapes, as they represent humanoids, animals, and other creatures, so that they may not properly block all LED-sensor pairs that would be required for accurate

detection. The sensor array would be more complicated and expensive when compared to a single IR camera as used in Rear DI and DSI. Further, the technique is not widespread and although some pre-existing documentation and software from previous projects using the technique exist, there is very little compared to the plethora of information on Rear DI and DSI. Finally, the large polygons of the generated detection areas are guaranteed to provide a high level of inaccuracy compared to the diffused illumination techniques, in which the blobs typically line up closely with the edges of the contacting object/fingertip.

5.1.2 Touch Detection Scheme Selection

To determine which multi-touch detection scheme would best suit our needs, a weighted decision matrix was used, seen in Table 13. The matrix ranked the possible methods in several categories and weighed these rankings by a chosen importance level of the category. Rear DI had the lowest score, as well as the most first-place and fewest third-place rankings and was thus selected as our multi-touch detection scheme.

Table 13: Multi-Touch Technique Decision Matrix

Category	Importance	Rear DI	DSI	LT
Detection Accuracy	3	2	1	3
Display Surface Options	1	2	3	1
Cost	3	1	3	2
Ease of Implementation / Available Resources	2	1	2	3
Total		15	19	22

5.1.3 Touch and Display Surface Materials

Once Rear DI had been selected, it was necessary to choose materials for the touch and display surface. For our clear material, we could choose between glass and clear acrylic. The clear acrylics we could acquire were lighter and stronger than glass, at roughly a 50% increase in price, as shown in Table 14. The improvements in structural integrity were judged to be worth the price, especially considering the lack of experience we have with table manufacturing, and a clear acrylic surface was selected. OPTIX Acrylic was selected for its reputation and minimal thickness, as we intend to place the diffusive material under the acrylic to protect it, and a thick layer means an increased distance between the touched surface and the diffusive surface, which may lead to increased blurring or distortion of touch and object placement inputs.

Table 14: Touch and Display Surface Material Options

Material	Cost	Source	Status
Gardner Glass Products 30-in x 36-in Clear Glass	\$18.48	Lowe's	
30". x 36" x .094" Clear Glass	\$17.58	Home Depot	
OPTIX 36" x 30" x 0.093" Acrylic Sheet	\$32.78	Home Depot	
OPTIX 30" x 36" x 0.08" Clear Acrylic Sheet	\$26.49	Lowe's	Ordered

For a diffusive surface, some purpose-made rear-projection materials were considered, as well as drafting paper, which has been shown to be effective in previous projects by hobbyists and senior design groups. Considered options are detailed in Table 15.

Table 15: Diffusive Material Options

Material	Cost	Source	Status
30" x 42" x 0.003" Drafting Film Matte, 2-Sided	\$8.16	Grafix	Acquired
24" x 36" x 0.005" Drafting Film Matte, 2-Sided	\$6.78	Grafix	
Grey Rosco RP Screen with Finishing	\$29.50/ft ²	Full Compass	
36" x 36" Digiline Contrast Projection Film	\$52.36	IFOHA	
Carls Gray Rear Projection Film	\$64.95	Amazon – Carls Place	

Purpose-fabricated rear-projection materials were found to be exceedingly expensive, too large for our purposes and not easily trimmed, or both. Concerns about possible tears or wrinkles in drafting film were mitigated by the ability to fortify the material with clear silicone, which was cheaply available. Comparatively, drafting film was a negligible expense and if after testing/assembly drafting film proved insufficient for our rear projection needs, it would not be significantly more expensive to order a professional material later than if we had started development with such a material. Thus drafting film was selected as our initial diffusive material.

5.1.4 Illumination

To provide the necessary infrared illumination for the Rear DI technique, several pre-built IR illuminators and IR LEDs were considered. The most promising options are detailed in Table 16.

Table 16: IR Illumination Options

IR Source	Cost	Source	Status
EMITTER IR 850NM 100MA RADIAL	\$0.39 - \$0.50	Digi-Key	Tested
EMITTER IR 940NM 65MA 0805	\$0.1575 - \$0.49	Digi-Key	Selected
EMITTER IR 850NM 65MA 0603	\$0.522 - \$0.67	Digi-Key	Tested
JC Infrared Illuminator Silver	\$11.99	Amazon - JCHENG SECURITY	
850nm 6 LEDs 90 Degree Wide Angle IR Illuminator	\$21.99	Univivi	

Pre-built illuminators were generally intended for use with night-vision security cameras and had power supply interfaces that would be difficult to integrate into our design. Additionally, pre-built illuminators provided intense light that may have blinded our close-range camera by illuminating the entire surface so brightly that blobs could not be detected. IR LEDs and emitters on the other hand provided reduced illumination per unit, but greater flexibility in placement, which was a great benefit given the number of other components that required specific placements when installed in the table. The ability to adapt the layout to achieve uniform illumination was . Eventually several

LEDs and emitters were selected for testing based on cost, illumination angle, and brightness, and preliminary testing was carried out with our IR camera. Based on this testing, the 940 nm emitter was selected.

5.1.5 Infrared Camera

To record the blobs and thus detect touch inputs to the table, a camera which viewed only the infrared spectrum was required. Cameras built for infrared imaging were prohibitively expensive, so we considered standard web cameras in which there was no IR-blocking filter or the IR-blocking filter could be removed, and a visible light filter could be added. There was no purpose-made filter for most cameras, but a large filter for another camera may be modified to fit or a piece of exposed film negative or floppy disk can be used.

To ensure the accuracy of blob detection, we had to determine a sufficient image resolution. Since we expect the objects and fingertips which we will be detecting to be at least 1/4 inch in diameter and located on a surface that is at most 36 inches on a side, for detection within 1/2 inch we need at least 144 pixels in a dimension. For reliability, we triple that number for a minimum of 432 pixels x 432 pixels. Given standard resolutions, we considered webcams that provide resolutions of at least 640 x 480.

To minimize touch detection times, we wish to maximize the frame rate of the camera at thus the detection rate. Further, if the frame rate is too small, not only will there be noticeable delays in inputs at a single location, certain functions which rely on moving finger inputs to determine the direction of characters' actions may be impossible to implement. As these gestures may take as little as 1/4 of a second, and we wish to track at least 7 frames for gesture inputs, a minimum of 28 frames per second is required. Given standard frame rates, we considered webcams that provided at least 30 frames per second.

Since we plan to use a user's PC to assist in running the game, a USB interface and PC-compatible driver were the best options for the webcam. To minimize touch detection times, we required that the interface be USB 2.0 or newer. Cameras meeting these requirements are given in Table 17.

Table 17: Camera Options

IR Source	Cost	Source	Status
SainSmart 5MP 1080P Webcam Camera NoIR	\$19.99	Amazon - SainSmart	
PlayStation Eye	\$8.70	Amazon - PlayStation	Selected
Makerfocus Raspberry Pi Night Vision Webcam	\$23.99	Amazon - MakerFocus	
Cimkiz USB Webcam	\$9.99	Amazon - CimKiz	
Kodak 35mm Color Negative Film	\$3.49	B & H	
3-1/2" Diskettes x 10	\$18.99 \$0.00	Amazon - Imation Owned	Selected
ZoMei 55MM IR 760 Glass Infrared Filter	\$23.99	Amazon - ZoMei	

The PlayStation Eye camera was selected based on price and because its use in previous projects means there is ample reference material for conversion from a visible light camera to an infrared camera, as well as a reliable driver for Windows PCs for and additional \$3. Further, the camera

can record at a resolution of 640 x 480 at a rate of 60 frames per second by default, and the Windows driver we acquired allows us to select the desired frame rate. Once the camera was acquired, it was discovered that the infrared spectrum was not strongly filtered out by the hardware, as shown in Figure 7(a), so a visible light filter could be added externally rather than risking damage by opening the camera and attempting to replace the existing weak filter. When facing a bright light in the form of an incandescent lamp, sunlight, or LED flashlight, the camera still picks up some object outlines, as shown in Figure 7(d). As these sources are known to produce some levels of infrared radiation, it is unclear to what degree this is the result of infrared radiation produced by the light sources as opposed to imperfect filtering, as some bright lights can be seen through the filter by eye. Floppy disk material was chosen as the visible light-blocking filter based on price and filter testing.



Figure 7: Infrared Camera Images

5.1.6 Display Method

To display images onto the touchscreen for the game, using the rear DI method, a projector of some description was necessary. As the table is restricted in height, either a short throw projector or a regular projector combined with a mirror to increase the throw distance was required. New short throw projectors are several hundred or over a thousand dollars more expensive than regular projectors, so the mirror technique was chosen if a new projector was to be acquired. Because the mirror method involves projecting the image at an angle, significant keystone correction was necessary for undistorted image reproduction. Originally, a high resolution of 1080p was desired to ensure the details of the game could be distinguished. After some examination of available standard projectors, some 720p options were also considered as the image would remain acceptably clear at 1/3 the cost, which could mean hundreds of dollars of savings and a 25% or

more reduction in our overall budget as the projector was expected to be the most expensive component.

Despite the increased cost, short throw projectors were considered as the implementation would be simpler and more reliable than the mirror method, as well as providing more flexibility in the dimensions of the table. Any cosmetic damage to the projector was irrelevant as the projector would be enclosed in the box of the table and not visible. Minor discoloration in the image could be tolerated and guarantees on remaining lamp lifetime were enough for our purposes. These were the only expected reductions in product quality relevant to our requirements. Some reputable vendors of refurbished projectors were identified, and candidate refurbished projectors selected which had comparable prices to a new standard projector.

In addition to image quality and throw ratio considerations, control methods were also taken into account, as we wish to be able to easily turn the projector on and off and adjust the display brightness once the projector has been installed in the table. Every projector considered came with an infrared remote which we could modify or use to design a control system. Our top options are detailed in Table 18. The final selection was made in November in the hopes of acquiring an exceptional projector cheaply during Black Friday/Cyber Monday sales, as there was little concern about the compatibility of the projector with the rest of the design, and it was not particularly necessary to test other components at the early stages.

Table 18: Projector Options

Projector	Resolution	Throw Ratio	Keystone	Cost	Source
VANKYO Performance V600	1920 x 1080	1.5	$\pm 15^\circ$	\$249.99	Amazon - Vankyo
YABER Native 1080P LED Projector	1920 x 1080	1.3	$\pm 50^\circ$	\$239.99	Amazon - YABER
Crenova Native 1080p LED Projector	1920 x 1080	1.0	$\pm 15^\circ$	\$249.99	Amazon - Crenova
BenQ MP780ST DLP Short-Throw Projector	1280 x 800	0.5	$\pm 40^\circ$	\$189.56	eBay - voltarea
BenQ MX810ST MX713ST DLP Short-Throw Projector	1024 x 768	0.6	$\pm 30^\circ$	\$178.76	eBay - voltarea
Plain Square Mirror by ArtMinds				\$6.49	Michaels

As no particularly strong contenders went on sale in November, we decided to go with a used short-throw projector. The used projectors were cheaper than comparable new projectors and organizing the hardware inside the table is simpler when using a short throw projector, as the mirror method poses challenges in terms of placing items out of the path of the projected light. Additionally, most projectors in our price range have only auto-keystoning options which may struggle to adjust the display accurately when facing an angled mirror. After narrowing down the field to used short-throw projectors and identifying a couple of reputable dealers, two BenQ projectors emerged as the top contenders, the BenQ MP780ST and the BenQ MX810ST. The prices of both are comparable, so the choice comes down purely to projector specifications. The projectors are similarly sized and have the same brightness rating. They also both have some serial control ports that we may be able to use to adjust the projector brightness and other settings from the outside of the table. The MP780ST has a slightly higher resolution, greater keystoning range,

and a shorter throw ratio. In contrast, the MX810ST supports smaller minimum image dimensions, has a longer lamp life, and a greater contrast ratio. The minimum image dimensions for the MP780ST are slightly larger than we need, so that displaying the image in the projector-supported size range would require scaling our table so large it could not easily be sat around or maneuvered, while adjusting to a smaller size may result in an unfocused image. The keystoneing and throw ratio of both projectors are enough for our intended design, so overall the BenQ MX810ST DLP Short-Throw Projector was a better choice and therefore selected. The dealer we order from guarantees that the projector lamp has at most 1190 hours of use, so at least 2310 usable hours remain. We estimate that the table will be used on average 4.5 hours per game, and at most 3 games per week. With 52 weeks in a year, the used projector lamp should then last $2310/(4.5*3*52) = 3.29$ years, which is sufficient for our intended minimum device lifetime of 3 years. Table 19 provides the detailed specifications of the MX810ST projector.

Table 19: Projector Specifications

Projector			
Brightness	2500 ANSI Lumens	Keystone	Auto/Manual, $\pm 30^\circ$
Native Resolution	1024 x 768	Aspect Ratio	4:3
Contrast Ratio	4600:1	Dimensions	11.4" x 5.03" x 9.9"
Display	DLP	Power Supply	100 to 240 VAC, 50/60 Hz
Lamp Life	3500 hours	Throw Ratio	0.6

5.1.7 Touch Surface Prototyping

Hardware prototyping began with the touch detection scheme. The infrared camera was the first element of the multi-touch scheme to be developed in order to determine its sensitivity to IR light and to have a method of validating the IR illumination scheme. The array of IR LEDs for illumination of the touch surface was then designed to try to achieve as uniform an illumination as possible and thus make the job of the touch-detection software as easy as possible. The table surface was then assembled, with the diffusive surface attached to the clear acrylic sheet and then placed temporarily in a closed box to begin testing. At this point adjustments to the LED array and diffusive material could be adjusted to optimize both the display clarity and touch input detection accuracy.

Several configurations of the LED array were tested to determine which provided the best illumination of the touch surface. The configurations chosen for testing were the square, offset square, ring, and cross configurations detailed in Figure 8, in addition to a configuration consisting of two lines along the length of the table. The illumination of the touch surface by each configuration was recorded using the infrared camera and the results compared. During final assembly, some modifications may be made to the configuration as some area of the table will be occupied by other components, especially the projector and camera. As a result of the testing, and after a comparison of the impact on illumination caused by the removal of a section of LEDs, the offset square configuration was selected.

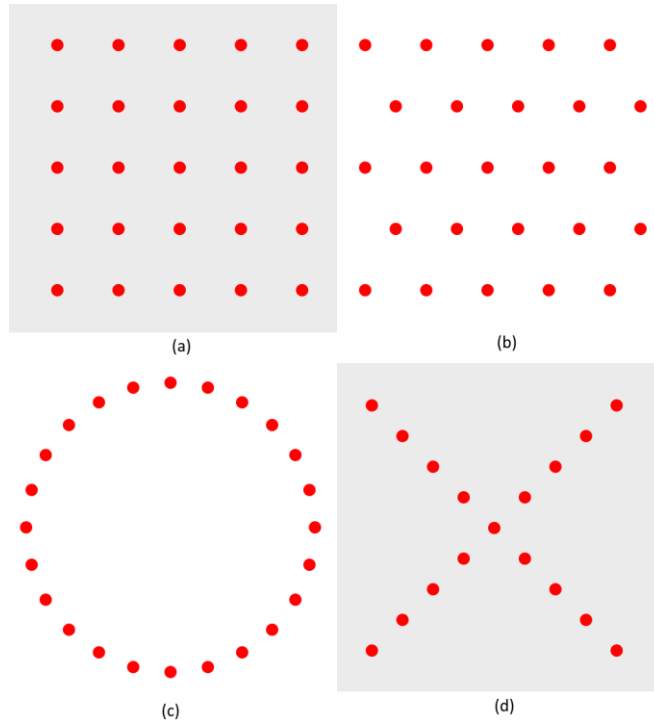


Figure 8: Infrared LED Configurations
 (a) Square (b) Offset Square (c) Ring (d) Cross

5.2 Table

The physical structure supporting the rest of the hardware must be chosen to provide enough space for all components and a reasonable throw distance for the projector, without becoming too tall or bulky for users to sit around during gameplay. The unique needs of the project left three main options for the table: commission a custom table, purchase a standard pre-built table and modify it, or build a table from scratch. Commissioning a table would guarantee the best results, as none of the group members have significant experience with woodworking. However, this option is also the most expensive, and thus was discarded early on.

When searching for pre-built structures to modify, we look for as many characteristics matching those we desire as possible. Glass-top tables, cabinets, and wheeled tables are especially desirable. Glass-top tables may need no replacement of the top surface, or at the very least be easy to replace with an appropriate acrylic. Cabinets fulfill the requirement of an accessible closed box without modification, come in appropriate sizes, and are easily ordered to custom specifications, although custom ordering is quite expensive. Wheeled tables will provide maneuverability while remaining stable. In all cases the material and structure of the table should be such that modifications can be safely made, such as wood or particleboard, in the areas requiring modification. After searching half a dozen retailers and online marketplaces, no glass-top tables, cabinets, or wheeled tables were found which would provide significant benefits over building from scratch. However, a shelving unit already in the possession of one of the group members was found to be a good base for a modified design.

The shelving unit consists of a metal frame perforated with holes along the length of each strut. These perforations provide connection points for shelving support structures. Each original shelf

is composed of particle board, easily removed and replaced with more appropriate materials. The surface area of each shelf is 2 feet by 4 feet, and the total height of the shelving unit is 3 feet. Each shelf is held in place by a supporting metal framework underneath its edges, which can be left in place at the top of the unit to hold our touch surface and surrounding table top, as well as at the bottom of the unit to support a ground-level shelf which will hold most of the table's electrical hardware.

The framework for each shelf's support is 3/8 inch deep, and thus we can use an acrylic sheet up to 3/8 inch thick while maintaining a closed box structure for the touch surface. As our intended less than 1/10 inch thick, this depth is sufficient. As the projector produces an image with a 4:3 aspect ratio, a 24 inches tall image will be 32 inches wide. After trimming the acrylic to this size, 16 inches of top surface remains uncovered. We address this deficiency by adding two 8 inch by 24 inch wood panels, one on either side of the acrylic, to create a small table area on which users may place cups, snacks, dice, and other small items. Any slight gaps left between the surface of the table and the frame will be filled with foam to prevent the acrylic and wood panels from sliding and sustaining damage. The boundaries between surface materials and the frame will be covered by thin wooden trim for aesthetic purposes and to ensure a light-proof seal. A hinged door will be added to one side to allow easy access to the interior of the table.

However, the shelf-based design has several issues. Attaching wheels would be difficult as they could not be bolted to or through the metal frame, and would therefore have to be attached closer to the center of the table, resulting in reduced stability and greater strain on the joints between the frame and the bottom shelf holding the electrical hardware, and a failure at this joints would cause potentially irreparable damage to the wiring of the table. Additionally, the dimensions of the shelf are not ideal, and the shortest dimension is slightly shorter than we would like and limits the touch surface area, while the longer dimension is much longer than we would like and forces us to purchase extra material and may cause issues when illuminating the surface from the interior. Overall, a table built from scratch is preferred.

Due to the lack of experience among the group, and the need for an enclosed space for rear DI, the from-scratch table design is simplistic but functional and can be seen in Figure 9.

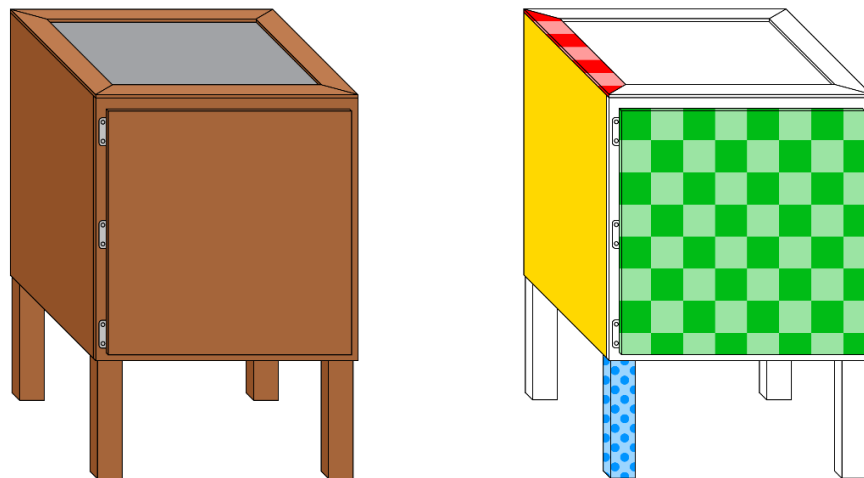


Figure 9: From-Scratch Table Design: Colorized (left) and Significant Components Highlighted (right)

Each side panel of the table (solid yellow) is made of plywood to provide a stable structure for embedding fans, buttons, etc. and to support the top of the table. The bottom of the table (not pictured at the angle shown) is made of the same plywood as the side panels and will bear most of the weight of the electronic hardware such as the projector and microcontroller. The legs of the table (dotted blue) are used to provide a stable structure and enough depth to securely fasten the other components together. A door (checkered green) of thin plywood is placed on one side of the table to allow easy access to internal electronic hardware for the purposes of installation, troubleshooting, and maintenance. The door attached on one side by hinges and held closed by a Velcro closure when not in use. Craft foam may be placed around the edges of the door to prevent impact damage when the door swings shut and to block light from leaking into the box when the door is closed. A frame (striped red) at the top of the table is added for aesthetic reasons, covering the tops of the legs to create a rectangular touch surface. The frame will also provide support for the acrylic which will be affixed to the top of the table. The acrylic will be adhered to the underside of the frame and if necessary some support beams will be added underneath the acrylic, hidden by the frame. Modifications in the form of handles on the sides and wheels on the bottom are acquired to be added after construction if the smooth box design proves difficult to maneuver safely. The exact dimensions of the table were not determined until after the acquisition of the projector, to ensure the visual display would completely cover the touch surface and the necessary throw distance could be guaranteed. Once this was done, the design of Figure 9 was scaled and constructed. The major dimensions are given in Figure 10.

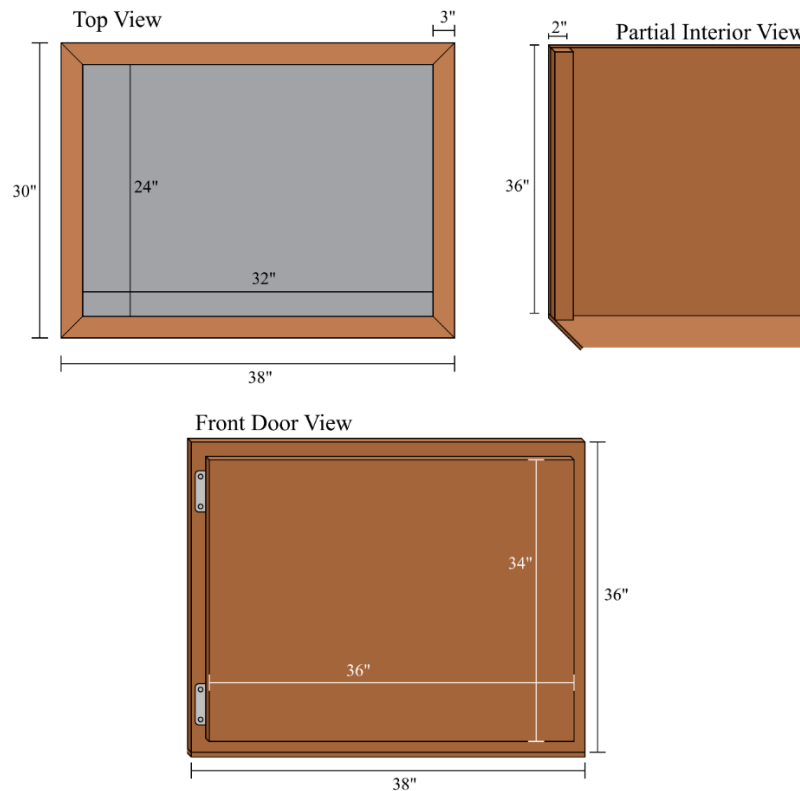


Figure 10: Major Table Dimensions

From the projector’s specifications, we know the native image resolution has a 4:3 aspect ratio. 24 inches for the smaller dimension is selected to meet minimum display size requirements and to make sure the table can fit through most standard doors. A 24 inch by 32 inch image will have a 40 inch diagonal. From the projector’s user manual, we determine that a throw distance of 494 mm, or just under 19.5 inches is required. Further, the projector is 9.9 inches long. Allowing 6 inches to plug cables into at the rear of the projector without damage, the closed box portion of the table must be at least 35.4 inches tall, which we round up to 36 inches for simplicity. As we expect to add a few inches to the table’s height by attaching wheels beneath the table, we do not lengthen the legs past the bottom of the enclosed chamber. This further simplifies construction and prevents the table from becoming too tall to reasonably sit around. The legs are chosen to be 2 inches square, on recommendation from hardware professionals with woodworking experience. On similar advice, 1/4-inch-thick plywood is chosen for the sides, bottom and top frame of the table, as well as the door for ease of ordering. The frame of the surface is then chosen to be 3 inches wide, to cover both the legs and the siding. One inch is left on each side of the door for strong hinges and a fastener to hold the door closed. One inch of overlap is left between the door and the table siding to block light and leave room for foam padding as necessary. The table is assembled using a combination of screws and wood glue. A summary of the materials and tools used in construction is given in Table 20. Tools for construction such as drills, paintbrushes, and clamps were either already owned by team members or available in UCF labs and thus effectively free to use, so they are not detailed here.

Table 20: Table Materials

Part	Unit Cost	Number	Total Cost	Source
Sandeply 1/4" x 4' x 8' Plywood	\$22.92	3	\$68.76	Home Depot
2" x 4" x 12' Lumber	\$6.92	1	\$6.92	Lowe's
Wood glue	\$5.98	1	\$5.98	Lowe's
Screws	\$1.08	1	\$1.08	Lowe's
Hinges (2 pk)	\$3.49	1	\$3.49	Ace Hardware
Caster Wheels (x2)	\$4.99	2	\$9.98	Ace Hardware
Handles	\$3.59	2	\$7.18	Ace Hardware
Paint	\$10.28	1	\$10.28	Lowe's
Total			\$113.67	

The plywood chosen is selected for its low cost and its smooth surface, which has already been prepared for painting. The ease of painting is an important consideration as we have no experience staining wood, and thus to improve the appearance of the table, wooden surfaces of the table may either be painted or shrouded in decorative fabric. Paint will maintain the shape of the surfaces of the table but will be more time consuming as we must allow drying time, and there are greater health and safety restrictions concerning paint fumes than there are for fastening fabric. Fabric will require less effort and could allow more elaborate decorative patterns but must be attached securely and stretched out to cling to the table’s shape. It would also be susceptible to tearing and add a layer of insulation that would force the cooling system to work harder to maintain a low

temperature within the desired operating range. Ultimately, paint is a better choice for our project, although a protective cloth cover could be made for the table's touch surface.

5.3 Microcontroller

A microcontroller unit (MCU) is a small computer contained within an integrated circuit chip. These chips generally contain a central processing unit (CPU), memory, and programmable input/output peripherals. When many people think of a microcontroller, they think of something like an Arduino board. These do contain microcontrollers, specifically the ATmega328 for the Arduino Uno, but also contain pin headers and other accessories for easy powering and programming. These are known as development boards and are primarily used for prototyping

This project will use a custom printed circuit board (PCB) containing an MCU. The MCU will be used for reading a temperature sensor, controlling fans, LED effects, controlling an LCD display, and potentially playing sound effects on a speaker. The main advantage of using the MCU for these things is speed. It is also useful to keep these simpler functions separate while the single board computer is used for more complicated things like object detection and controlling what is displayed on the projector. The follow sections detail the process of selecting an MCU for this project.

Various microcontroller units were considered based on factors such as number of inputs/outputs, functionality, and cost. Other factors such as familiarity and online resources such as tutorials and example code were also considered. The goal is to find an MCU with good enough performance, memory, and enough I/O lines for the planned features while keeping costs down.

5.3.1 ATmega328

The ATmega328 is a single-chip microcontroller created by Atmel (currently owned by Microchip Technology). It has an 8-bit reduced instruction set computer (RISC) architecture and achieves one million instructions per second per megahertz. Some features of the ATmega328 include 32KB ISP flash memory with read-while-write capabilities, 1KB EEPROM, 2KB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible timer/counters with compare modes, internal and external interrupts, serial programmable USART, a byte-oriented 2-wire serial interface, SPI serial port, 6-channel 10-bit A/D converter (8-channels in TQFP and QFN/MLF packages), programmable watchdog timer with internal oscillator, and five software selectable power saving modes. The device operates between 1.8 and 5.5 volts.

This microcontroller is a good potential candidate for this project. One benefit of using this microcontroller is there are a large amount of resources and tutorials available due to this device being used in the Arduino Uno development board. 32KB of flash memory is also plenty considering the simple operations the microcontroller will be performing. A 16MHz crystal will need to be used as an external clock due to the internal RC oscillator being unreliable but this is a very cheap device and not a problem. A potential problem using this microcontroller is the number of I/O lines. The device will have multiple outputs operating at the same time including a display for a timer that will take a large number of lines to control various segments.

5.3.2 MSP430FR6989

The MSP430FR6989 is an ultra-low-power (ULP) device with a Ferroelectric Random-Access Memory (FRAM) platform that combines embedded FRAM and holistic ULP system architecture to increase performance and consume less power than other MCUs. Features of this device include

a wide supply voltage range (3.6 to 1.8V), optimized ULP modes, ULP FRAM, intelligent digital peripherals, high-performance analog, multifunction input/output ports, code security and encryption, enhanced serial communication, and a flexible clock system.

The main benefit of using this device is familiarity due to using it in UCF’s Embedded Systems course. In that course multiple features of the device were tested including adjusting the clock settings, timing interrupts, using UART, I²C, and SPI, using analog-to-digital conversion, and using an LCD pixel display. All of this was programed at the register level in Code Composer Studio and implemented using the MSP430FR6989 Launchpad Development Kit. The development kit is similar to Arduino in that it combines the MCU with multiple peripherals and pin headers that allow for quick prototyping. Even though there are less online resources for this MCU compared to the chips used in the Arduino boards it is still a strong candidate for this project due to the large pin count and familiarity.

5.3.3 ATmega2560

The ATmega2560 is a high-performance, low-power Microchip 8-bit AVR RISC-based microcontroller that combines 256KB ISP flash memory, 8KB SRAM, 4KB EEPROM, 86 general purpose I/O lines, 32 general purpose working registers, real time counter, six flexible timer/counters with compare modes, PWM, 4 USARTs, byte oriented 2-wire serial interface, 16-channel 10-bit A/D converter, and a JTAG interface for on-chip debugging. The device achieves a throughput of 16 MIPS at 16 MHz and operates between 4.5-5.5 volts.

This MCU is similar to the ATmega328 but has an increased number of I/O ports and more memory. Like the ATmega328 this device has a large number of helpful tutorials and resources due to it being used in the Arduino Mega development board. Another benefit is the ability to use the Arduino board to prototype and test various peripherals before committing to a PCB. The increased number of I/O lines remedies the problems faced when using the ATmega328 so this the best candidate out of the three MCUs considered.

5.3.4 Microcontroller Comparison

A comparative summary of the MCU parameters is given in Table 20 below.

Table 20: Microcontroller Comparison

	ATmega328P	MSP430FR6989	ATmega2560
CPU type	8-bit AVR	16-bit ULP	8-bit AVR
Performance	20 MIPS at 20 MHz	16 MIPS at 16 MHz	16 MIPS at 20 MHz
Flash memory	32 KB	128 KB	256 KB
SRAM	2 KB	2 KB	8 KB
EEPROM	1 KB	0 KB	4 KB
Pin count	28	100	100
Maximum operating frequency	20 MHz	16 MHz	20 MHz
Maximum I/O pins	23	83	86
External interrupts	2		11
Cost	\$2	\$8	\$12

5.3.5 Uploading Bootloader to MCU

When a chip is received from the manufacturer it is blank and is unable to run programs. To solve this problem a bootloader is required. This is the first program that runs when an MCU is reset and allows the MCU to receive new programs through some means of communication. To simplify programming the MCU on the PCB the bootloader from the Arduino Mega will be used. This allows us to reuse the same programs that were used when prototyping on the development board. A guide was found that shows how to use SPI to burn the Arduino Mega bootloader from an Arduino Uno to the target chip [reference]. Table x shows which pins are used to upload the bootloader and their function. The pins in a given row are connected. The VCC and GND pins must also be connected. To implement this a 6-pin header will be created to be used as an In-Circuit Serial Programming Interface (ICSP).

5.3.6 Communicating with the MCU

Once the bootloader has been burned there needs to be a way to upload software that will be run by the MCU. For the Arduino Mega a second MCU called the ATmega16U2 is used specifically for USB-to-serial conversion. If it wasn't for this chip the ATmega2560 would not be able to recognize the USB input. Implementing this in our design would require many more parts, increase complexity, and increase the cost of manufacturing the PCB. Instead an already built USB-to-serial chip module will be used. The chip will allow the MCU to interface with a pc through UART. A module based on FTDI's FT232RL chip was chosen. The module contains all of the required pins discussed in the next section and LEDs that indicate when data is being transmitted.

The two most common protocols for serial communication with MCUs are transistor-transistor logic (TTL), also known as UART, which uses +5V and 0V for its voltage levels, and RS-232 which uses $\pm 12V$ for its voltage levels and is primarily used for long distance communication. It was previously decided that UART would be used to communicate with the MCU. The ATmega2560 has UART built in so this will be easy to implement. Multiple connections will be required including: receive (RX), transmit (TX), and Data Terminal Ready (DTR) along with the 5V and GND connections. The DTR pin is connected to reset of the MCU and when connecting the USB-to-serial chip will set the DTR pin to low. A capacitor is placed in series with the reset line so the reset pin will return to 5V. The receive pin of the USB-to-serial module is connected to the transmit pin of the MCU and the transmit pin of the module is connected to the receive pin of the MCU.

5.3.7 Standalone MCU Schematic

Once it had been determined how the MCU will be programmed and how the PC will communicate with the MCU a standalone MCU schematic could be created. This schematic shown in Figure 11 includes:

- The ATmega2560 MCU
- A 16 MHz crystal oscillator
- A 6-pin header for burning the bootloader
- A reset button pulled up to VCC
- Two 5-pin headers for UART communication
- Decoupling capacitors
- A 20-pin female header for digital pins
- A 5-pin female header for analog pins

- A power indicator LED

These subsystems are labeled in the schematic and arranged so that they are in close proximity to relevant pins. The 16 MHz crystal is needed because the internal oscillator of the ATmega2560 is not reliable and only runs at 8 MHz. The pin header for the boot loader allows for an easy temporary connection. Two headers are used for communication. The first one will be dedicated for uploading programs and the second one will be used for receiving UART commands during gameplay. Decoupling capacitors are required to reduce noise from the chip and traces. The digital pin header will be used for the 7-segment LED timer and the analog pin header will be used for any ADC that may be required. Finally, an LED will be connected from VCC to GND and will act as a power indicator. This design will be used as a foundation and will be updated based on which pins are necessary for the peripherals being used.

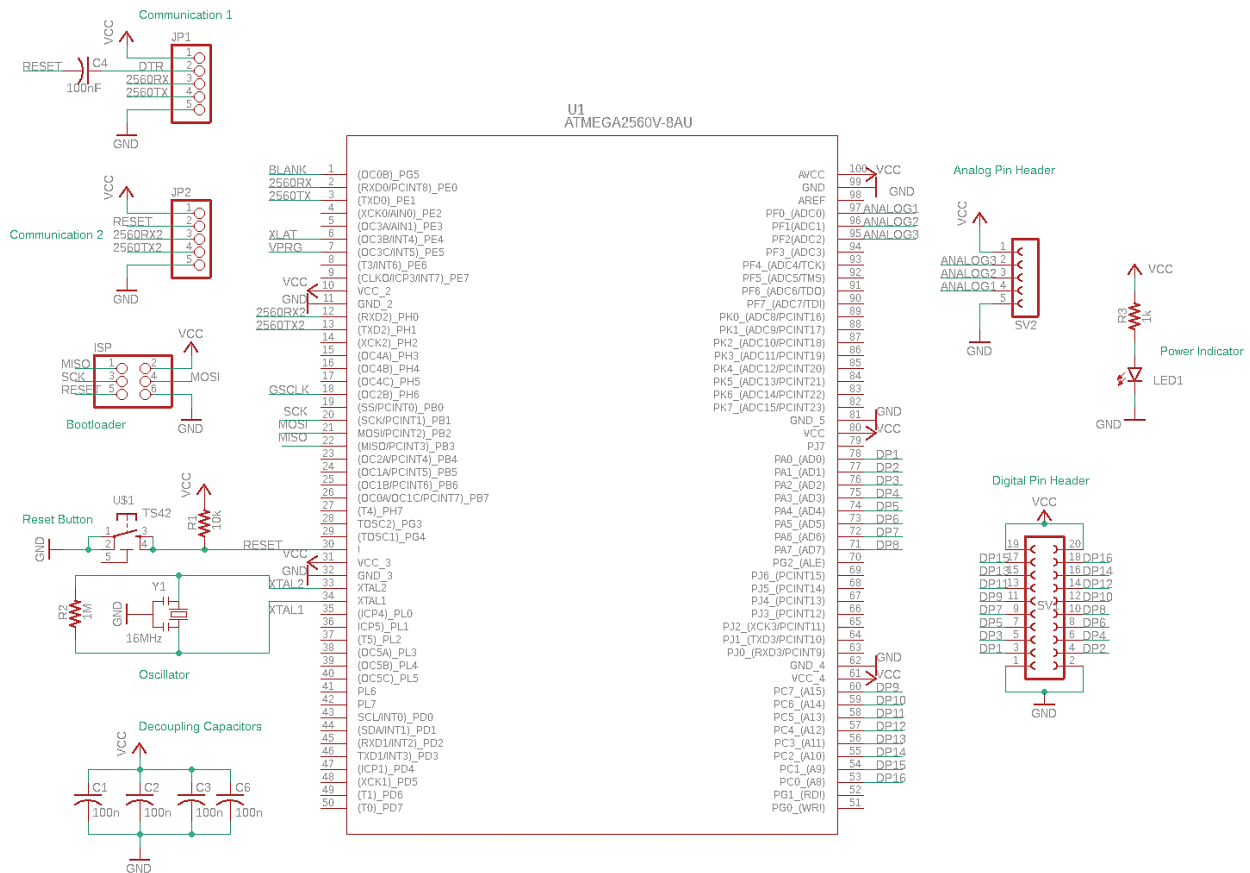


Figure 11: Microcontroller Schematic

5.4 Cooling System

A serious concern for this project is the heat generated by the various devices, especially the projector. This problem is magnified by the fact the components will be enclosed within the table. To reduce this heat two fans will be used, one for intake and one for outtake. The fans will be attached to opposite sides of the table so air will flow through the table and cool the enclosure. It was decided that a 12V fan would be used instead of a 5V fan to make sure enough air is flowing to reduce the temperature. Many of the lower voltage fans are used for computer cases which have less air volume so these fans will likely make little difference in a large enclosure. The chosen fan

has a current draw of 0.25A and runs at a speed of 1600rpm. It has an estimated air flow of 73CFM which is acceptable for this size of enclosure. The size of the fan is 120x120x25mm (4.72x4.72x4.13in) which will easily fit within the side of the table. If the cooling is not sufficient another set of fans could be added. The fans only cost \$6 so extras can be purchased.

5.4.1 Temperature-Based Control System

Controlling the fans so they only run while the enclosure is above a certain temperature will get rid of unnecessary power usage. In order to do this, there needs to be a way to detect the temperature of the environment. This is where the thermistor comes into play. Thermistors are temperature dependent variable resistors and are a cheap way of detecting changes in temperature. The most common type of thermistor is the Negative Temperature Coefficient (NTC) thermistor. For an NTC thermistor, increasing the temperature decreases resistance, and decreasing the temperature increases resistance. A chart can be used to determine which temperature corresponds to each resistance value. These charts are given in the data sheet for the thermistor. There is also a tolerance, usually 1-10%, that corresponds to the possible range of measured resistances for a specific temperature.

In order to control the fans using the thermistor the devices will need to interface with the MCU in some way. Two problems arise from this: the MCU detects changes in voltage, not resistance, and the MCU cannot supply enough current to run the fans. The solution to the first problem is to use a voltage divider containing the thermistor and another known resistor value. The voltage divider is then connected to the MCU as an analog input. As the temperature increases, the thermistor's resistance decreases, and the voltage at the input will increase. The ATmega2560 uses 10-bit analog to digital conversion. This means the input value will be somewhere between 0 and 1023 where 0 represents 0 volts and 1023 represents 5 volts.

The value of the ADC changes linearly with the input voltage so by using the voltage divider equation and the resistance of the thermistor given in the datasheet you can determine what voltage turns on the fan. For example, it is determined from the datasheet that the resistance of the thermistor at 30 degrees Celsius is 8k. If a 10k resistor is used in the voltage divider for the second resistor, and 5V is used for voltage source, the voltage of the divider at this temperature will be about 2.78V. Dividing this by 5 volts and multiplying it by 1024 gives the ADC value corresponding to that voltage which in this case is 569.

The solution to the second problem, the lack of current supplied by the MCU, is the use of a MOSFET. A MOSFET has three terminals, gate, drain, and source, and by supplying a high enough voltage to the gate, current is able to flow through the device. Using these properties, the gate can be connected to a digital output on the MCU and once the voltage at the analog input reaches the desired level specified in the firmware, the output will become high which turns on the MOSFET. By connecting the 12V power source, the fan, and the MOSFET drain and source in series the fan will run when the MOSFET turns on.

Using the MCU to control the MOSFET also allows the use of Pulse Width Modulation (PWM). This allows you to send repeated pulses instead of a constant current to the gate of the MOSFET which affects the speed of the fan because it is being run intermittently based on the duty cycle. The higher the duty cycle the faster the fan will run up to a max of 100% which corresponds to the output being always on. Therefore, the MCU can be programmed to have the fan run at different speeds based on temperature by controlling a PWM output. PWM on the ATmega2560 is based

on an 8-bit value that ranges from 0 to 255. An example of different duty cycles and their corresponding PWM values is shown in Figure 12.

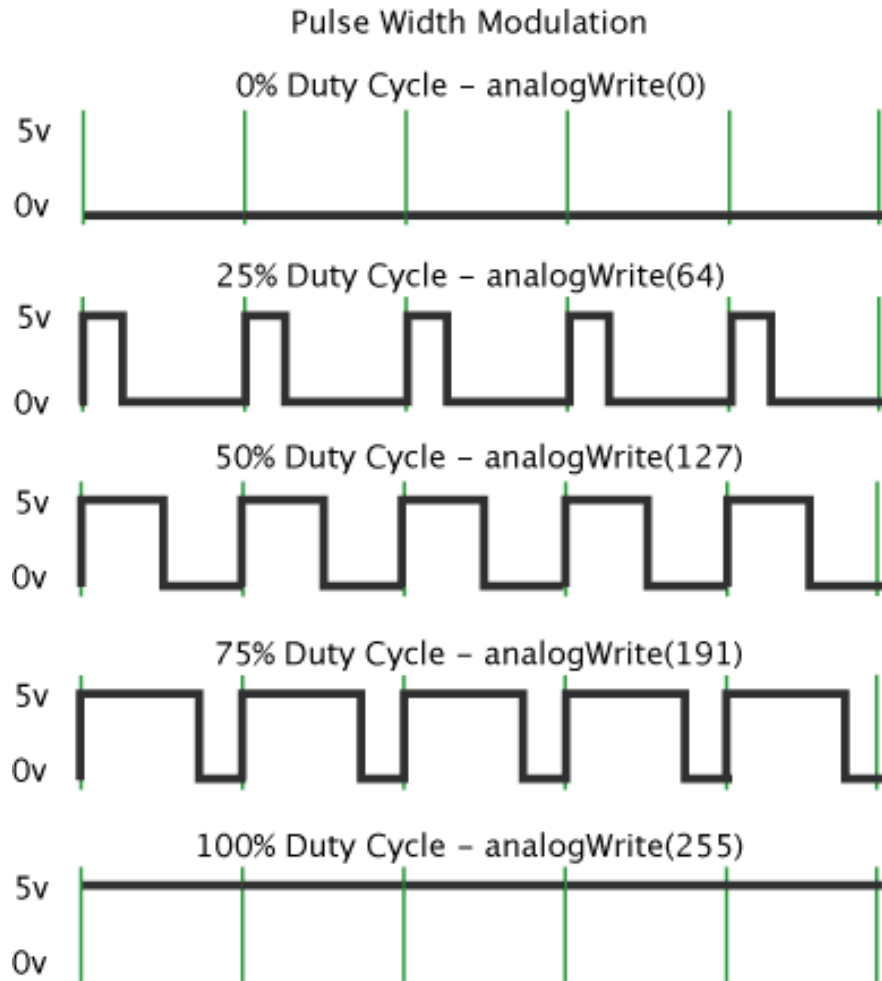


Figure 12: Example of 8-bit PWM

Reproduced in accordance with the Creative Commons Attribution-Share Alike 3.0 Unported license

5.4.2 Prototype of Temperature Control Subsystem

A prototype of the temperature control subsystem was created on a breadboard. The prototype uses the Arduino Mega development board which contains the ATmega2560 chip that will be used in the final design. The transistor used is the F12N10L which is a logic level N-Channel MOSFET with a gate threshold voltage of 1 - 2V and a maximum drain current of 12A. The fan is powered by a 12V AC adapter directly connected to the breadboard. The Arduino Mega requires 5V and is powered through USB. The fan is connected between the 12V rail and the drain of the MOSFET. The gate of the MOSFET is connected to a PWM port on the Arduino and the source is connected to ground. By applying a high output to the gate current can flow through the MOSFET and the fan turns on. A 10k Ω thermistor is connected between the 5V output and a voltage divider created using a 10k resistor. The voltage divider is connected to an analog port on the Arduino. The Arduino Mega's 10-bit ADC, as explained in the previous section, can be utilized through the

analog read function in the IDE. The ambient temperature was measured to be 77 degrees Fahrenheit which coincidentally is the temperature when the thermistor has a resistance of 10k.

Before connecting the fan, the ADC from the voltage divider was tested and read 510 on the serial communications monitor. This is very close to the expected value of 512. The bit controlling the digital pin going to the gate of the MOSFET was then configured so that it would be set high if the value of the analog read function became greater than or equal to 570. To test this the thermistor was touched to increase the temperature. The serial monitor read 1 which corresponds to high when the ADC value reached 570 as expected. Finally, the fan was connected as mentioned above and the entire circuit was tested. The fan only ran while the ADC value was greater than 570. Therefore, the prototype was a success.

It was decided that the MOSFET for the temperature control circuit would be part of the PCB. In order to do this a surface mount logic level MOSFET that could handle at least 0.5A of current was required. It was determined that the FDN359 N-channel logic level MOSFET would be used. This MOSFET has a maximum drain current of 2.7A and a gate threshold voltage of 1 – 3V which are well within acceptable ranges. It also comes in a surface mount package so it will take up less room than the F12N10L. Once a MOSFET was decided on a schematic of the temperature control circuit could be constructed. The schematic shown in Figure 13 contains the MOSFET, a 2-pin header for connecting the fan, solder points for connecting the thermistor and a voltage divider. The reason the thermistor will be soldered instead of making it part of the PCB is so the device can be placed in the optimal position to detect the temperature of the enclosure.

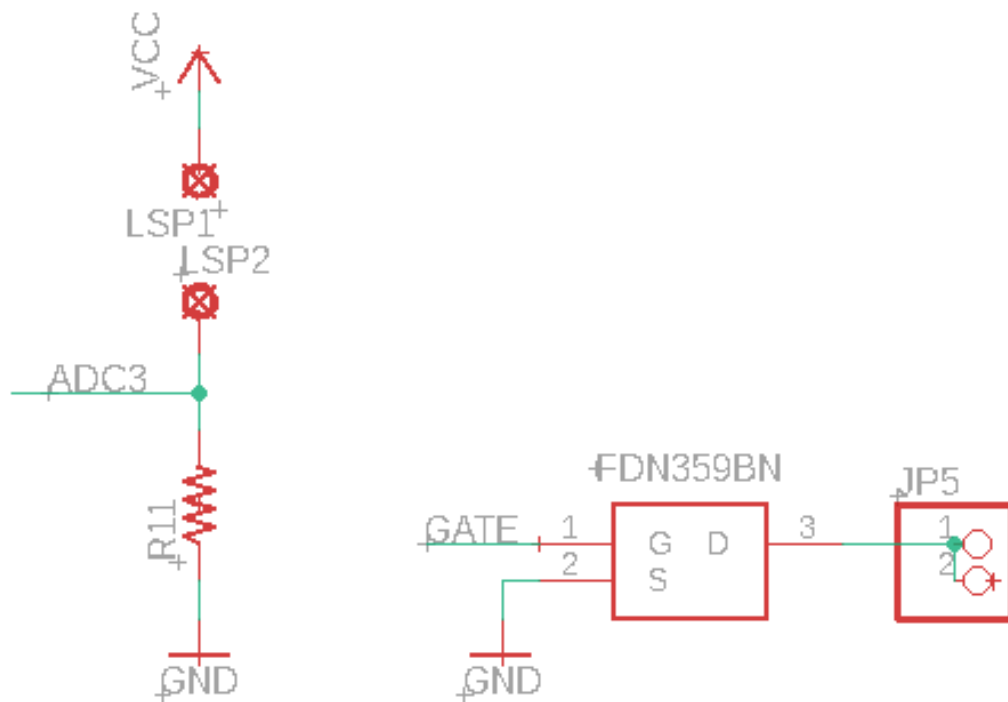


Figure 13: Temperature Control Circuit Schematic

5.5 Timer

A timer was added to the table for use in timed skill challenges and limiting turn durations. To display the time separately from the touch screen and phone apps, some sort of display unit is required. This display unit need not be very large, so we consider both LCD displays and LED displays. Since the table may need to operate in dim lighting, if we use an LCD display unit we must choose one with a backlight for visibility. Among LED displays, we consider 7-segment displays and dot matrix displays. Among the three categories, an LCD display provides the most flexibility as it can be used to display any black and white design that fits in the display window. If we decided later in development to generate some additional output, such as a welcome message as the table is turned on, or an error message when some problem is detected, this would provide another method of communication with the user. A dot matrix LED display could also be used for a wide variety of designs, but they would have to be less cohesive and smaller than a design for an LCD display. A seven-segment LED display is the most restrictive, able to display only a small set of alphanumeric characters in a few fixed positions. However, the more restrictive the display unit, the simpler the implementation, as a seven-segment LED display can be connected to the microcontroller by a relatively small number of binary I/O pins, whereas a dot matrix LED display or an LCD display typically requires serial digital communications to transmit the complicated design. Further, seven-segment displays are the cheapest option out of the three. Since we should be able to display complicated outputs on the table's surface, in apps, and on the host PC, a complex output for the timer should not be necessary. Thus, we choose a seven-segment display for our timer.

The timer shows 2 digits of minutes and 2 digits of seconds, as the longest time challenges are expected to be 30 minutes or less. To indicate that the timer has finished counting down, the display flashes 00:00 until stopped. A four-digit seven-segment LED display can be used to show the full duration and simplifies the timer construction. The four-digit seven-segment LED display has 8 input pins for the seven segments plus decimal point, and common anodes for each digit in addition to the colon between minutes and seconds. Multiplexing is used to set each digit and the display must be constantly refreshed at a high rate to avoid flickering. As decimal points are not needed, 12 pins are used. The timer interface relies on 4 buttons attached to the table. The explicit functions of the buttons are: power on/off, start/pause time, increment minutes, increment seconds. Simultaneous pressing of both increment buttons resets the timer to 00:00. The timer's logic and timing are provided by the microcontroller, with 16 I/O ports required in total. To meet timing accuracy requirements, the microcontroller should support real-time clock functions.

The timer relies on a single "time" counter, which is set by the user via incrementation, and which decrements each second as the timer is run. The timer has 4 states corresponding to different stages of operation. Each state may accept only a certain subset of possible of inputs, based on expected changes during the state's operation. These states are detailed in Table 21.

Table 21: Timer Operating States

Timer States	
State 1 - Off	
Input	Function
Button pressed: Power on/off	Enter state 2, set time = 00:00, update display
State 2 – On, Setting Time	
Input	Function
Button pressed: Power on/off	Turn off display, enter state 1
Button pressed: Increment minutes	Add 60 seconds to time, update display
Button pressed: Increment seconds	Add a second to time, update display
Buttons pressed: Increment minutes + increment seconds	Set time to 0, update display
Button pressed: Start/pause time	Enter state 3
State 2 – On, Counting Down	
Input	Function
Clock tick (1/second)	If time = 0, enter state 4 Else, decrement time by 1, update display
Button pressed: Power on/off	Turn off display, enter state 1
Button pressed: Increment minutes	Add 60 seconds to time, update display
Button pressed: Increment seconds	Add a second to time, update display
Button pressed: Start/pause time	Enter state 2
State 4 – On, Timing Complete	
Input	Function
Clock tick (2/second)	Flash 00:00 to indicate time is complete
Button pressed: Power on/off	Turn off display, enter state 1
Button pressed: Start/pause time	Enter state 2, update display

A prototype timer using a breadboard to connect components and used an already-acquired Raspberry Pi as the controller. The individual components of the timer were first tested in this setting. Once the microcontroller was acquired, the code was adapted and programmed into the microcontroller, and the breadboard-based prototype was tested once more. After demonstrating proper functionality under these conditions, a permanent PCB-based design was assembled and submitted to a final round of testing.

5.6 Brightness Adjustment

To maintain visibility of the game screen, the brightness of the displayed image can be adjusted. Manual adjustments can be made using two buttons on the table, one to increase brightness and one to decrease brightness. Further, an ambient light sensor will detect changes in the room’s lighting and automatically adjust the brightness of the image when a significant change occurs, if the maximum/minimum brightness has not been achieved. This is accomplished by sampling the ambient brightness once per second and comparing the sample at time t to a coarse array of brightness levels, assigning it to the brightness level $L(t)$. If the current level differs from the previous level, the image brightness is increased or decreased proportional to the number of levels of difference, i.e. $L(t) - L(t-1)$. The manual adjustment is treated as an offset to the levels, e.g. one

manual increase will result in brightness levels previously assigned to L(t) being assigned to L(t)+1. Some of the considered light sensors are given in Table 22.

Table 22: Light Sensor Options

Light Sensor	Cost	Source	Status
APDS-9005-020	\$0.51712	Digi-Key – Broadcom Limited	
APDS-9306-065	\$0.52601	Digi-Key – Broadcom Limited	Ordered
VEML7700TR-ND	\$0.6996	Digi-Key – Vishay Semiconductor	
APDS-9300-020	\$0.7272	Digi-Key – Broadcom Limited	
OPT3002DNPR	\$0.792	Digi-Key – Texas Instruments	
OPT3006YMFR	\$0.885	Digi-Key – Texas Instruments	

The levels are chosen once the projector has been acquired and are selected based on testing in development. The benchmark image quality is determined based on our requirement specification. The first brightness level is determined the ambient brightness at which the image quality matches the benchmark with the projector brightness at the minimum achievable by the projector. Subsequent levels are determined by increasing the projector brightness by one level (as supported by the projector itself), then increasing the ambient brightness until the clarity of the image has diminished to the same level as the benchmark. The process is repeated until reaching the maximum brightness of the projector. When assigning a brightness measurement value to a level, we select the minimum level for an ambient brightness matching or exceeding current level, assuming there one, to ensure the image quality matches or exceeds the benchmark, while minimizing the projector’s brightness and power usage. If there is no greater level, we assign the maximum level, as the brightness of the projector cannot be increased.

5.7 Special Effect Lighting

Lighting effects will be used for player actions such as attacking and casting spells. RGB LEDs will line the edges of the table and will be controlled by the MCU. When an action is done on the player’s phone a signal will first be sent through Wi-Fi to the single board computer. The computer will then send a signal to the MCU that selects which function the MCU will perform and a preprogrammed LED effect will occur.

LEDs are current driven devices. This means the illumination is determined by the amount of current flowing through it. LED drivers regulate this current at the desired level to make sure the current stays constant. In total 20 RGB LEDs will be controlled. If you tried to control these directly from the MCU it would take a very large number of pins which is impractical. LED drivers can be combined with SPI and PWM to reduce the number of output lines needed from the MCU. Two potential LED driver devices are looked at: the TLC5940 and WS2812B.

5.7.1 TLC5940

The TLC5940 is a 16-channel, constant-current sink LED driver that is capable of driving 120 mA per channel. It is a PWM unit with 12-bit duty control and 6-bit current limit control. The IC also contains integrated dot correction circuitry to compensate for variations in LED brightness. This is useful when there is a display containing an array of pixels that need to have uniform brightness but is not very important for this project. The device is controlled using Serial Peripheral Interface (SPI) and can be daisy chained together to increase the number of LEDs that are being controlled. Each IC has the ability to control 5 RGB LEDs since each LED requires 3 channels. The device has the ability to control the brightness and color of each individual LED which is important when trying to create effects such as simulating light moving around the table. There are many tutorials online about how to use this IC and overall it is a strong contender for this project

5.7.2 WS2812B

The WS2812B is an intelligent control LED light source where the control circuit and RGB chip are integrated into a package of 5050 components. It includes an intelligent digital port data latch and signal reshaping amplification circuit. It also includes a precision internal oscillator and a 12V voltage programmable constant current control circuit.

The device only requires a single data line and can be connected in series to form chains of LEDs. There is a delay as the signal propagates through the chain but this is not noticeable when using a small amount. Unfortunately, the communication protocol this device uses is not standard and is not supported by most microcontrollers. This means data transmission must be implemented by the software in a process known as bit banging. This can be challenging with low clock rates due to the high data rate of the protocol (800kbps). Therefore, although this device would be easier to wire than the TLC5940 discussed in the previous section the difficulties of using a custom protocol outweigh the benefits and this device will not be used for this project.

5.7.3 Led Effects Hardware and Software Design

It was decided the TLC5940 LED driver would be used to control the LED effects. The reason for using this IC is to reduce the number of outputs needed from the MCU by utilizing channels on the chip instead of directly connecting LEDs to the MCU. It also allows the ability to daisy chain multiple ICs together to control more LEDs if necessary. A tutorial was found that discussed how the chip could be controlled with an Arduino Uno and this was used as the basis for the design [15]. The next few paragraphs will give a more detailed explanation about how the chip works and how to set it up in the firmware.

The TLC5940 has 16 channels each with its own 12-bit PWM value (an example of PWM can be seen in Figure 12. The PWM value ranges from 0 to 4095 and the higher the value is the brighter the LED connected to that channel will be. By connecting the red green and blue leads of the RGB LED to three different channels the color of the LED can be precisely controlled (common anode LEDs must be used). There are a massive 2^{36} color combinations in total (4096x4096x4096). Each channel is also given a 6-bit (0 to 63) value for dot correction. Dot correction is used to compensate for the fact that the brightness for a given current will vary for each LED. It automatically adjusts the brightness variations between LED channels and other drivers in the SPI chain. The correction data is stored in a table on EEPROM that is integrated into the chip.

The device is controlled by a clock signal going from the MCU to the GSCLK pin. The frequency of this signal is 16 MHz for the ATmega2560 (an external 16 MHz crystal oscillator must be used)

but this can be divided to generate lower frequencies. This clock continuously counts up from 0 to 4095 and is reset by pulsing the blank pin on the TLC5940. After every cycle, which takes 256 μ s, the PWM value for each channel can be updated. This means the brightness and colors of the LEDs are changing about 3900 times per second. This is much faster than what would be possible using the pins on the MCU.

Now that the inner workings of the TLC5940 have been established the steps needed for proper setup in the firmware are presented. The steps are as follows:

- Establish ports on the MCU for the XLAT, GSCLK, VPRG, MOSI, and SCLK pins.
- Determine the bit order, data mode, and clock divider for the SPI
- Create a subroutine for dot correction. VPRG must be set to high while dot correction values are transmitted and XLAT must be toggled on and off to write the data to the dot correction register.
- Set up two timers for updating PWM values. The first timer is connected to GSCLK and counts from 0 to 4095. The other timer is much slower and will control the interrupt that transmits data via SPI every cycle and toggles the Blank and XLAT pins.

Care must be taken when populating bytes to be transmitted with SPI because the dot correction and PWM values are 6 and 12 bits respectively whereas a byte is 8 bits. A timing diagram can be found in the datasheet for two cascaded devices that shows how the XLAT and Blank pins are used to clock in data [16]. The diagram was not included in this report due to copywrite protection.

Another important feature of the TLC5940 is the ability to control the maximum current for each channel. The maximum current is determined by the I_{ref} pin and is given by an equation found in the datasheet: $I_{max} = 43.4 / R$. The R in the equation represents a resistor going from the I_{ref} pin to ground. For this design a 2.2k resistor will be used which gives a max current of about 20mA. Therefore, if 20 RGB LEDs are used, and each has 3 channels, the maximum current draw would be 1.2A. However, due to PWM this would not be a constant current and the LEDs are only in use periodically.

5.7.4 Prototype of LED Effects System

The TLC5940 IC has a 28-pin PDIP package that allows for testing on a breadboard. A prototype shown in

Figure 14 was constructed using an Arduino mega with 5 RGB LEDs to be controlled by the LED driver. The system is powered by an AC adapter connected through a barrel jack and a 5V linear regulator. The goal of this prototype was to gain a better understanding of the timing diagram found in the datasheet of the IC and to test different possible effects for the game. Basic effects such as setting all of the LEDs to one color were first tested. Then effects that could potentially be used in the game were tested. A list of the tests performed is given in Table 24. Certain pins on the Arduino must be used for access to specific timers or for SPI functions. This is why the ports used in the figure below appear to be nonsensical. Table 23 shows which digital pins on the Arduino were used, the corresponding pin number on the ATmega2560, and the pin on the TLC5940 that they are connected to.

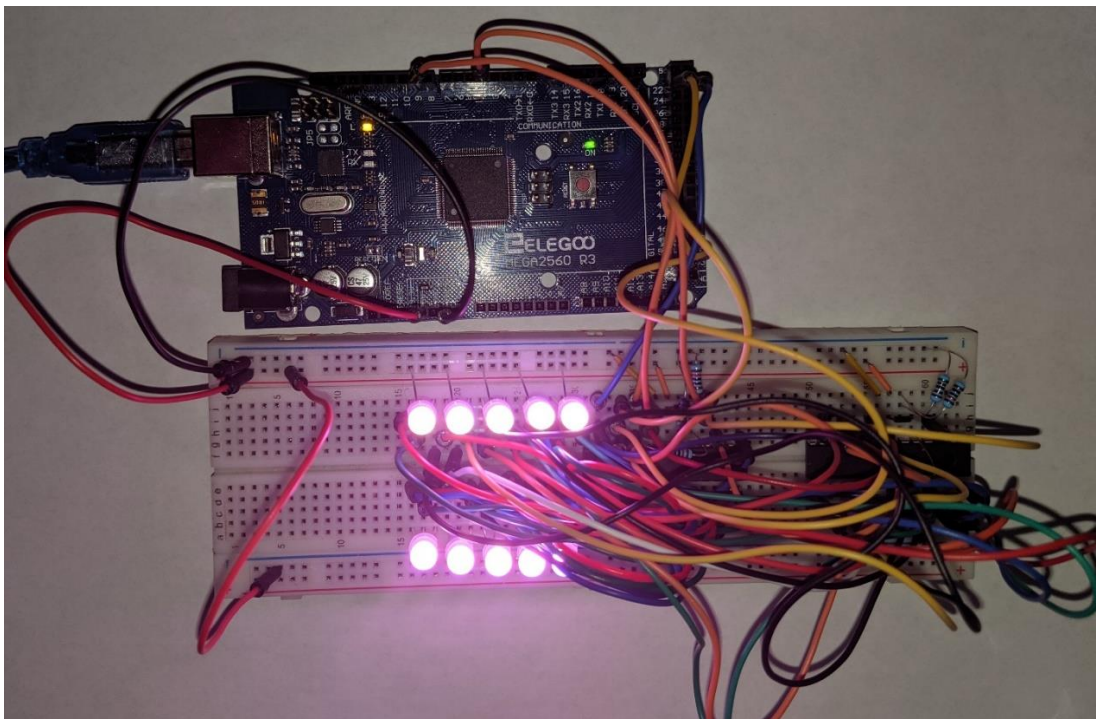
Table 23: Pin Connections for LED Effect Prototyping

Arduino Mega Pin	Port & Bit	ATmega2560 Pin	TLC5940 Pin
Digital Pin 22	PA0	Pin 78	XLAT
Digital Pin 9	PH6	Pin 18	GSCLK
Digital Pin 23	PA1	Pin 77	VPRG
Digital Pin 51	PB2	Pin 21	MOSI
Digital Pin 52	PB1	Pin 20	SPI CLK
Digital Pin 4	PG5	Pin 1	Blank

Table 24: LED Effects Tested Using Prototype

Test	LED Effect
1	All red
2	All green
3	All blue
4	LEDs flash white twice
5	Green light moves through row of LEDs
6	LEDs glow blue with increasing intensity and then dim
7	Random behavior

Figure 14: Prototype of LED effects system



After using an Arduino to become familiar with the TLC5940 a schematic shown in Figure 15 was built in Eagle. The pins for the XLAT and VPRG were changed from 78 and 77 to pins 6 and 7 respectively. The reason for doing this was proximity to the other pins being used such as for SPI and the fact that pins 78 and 77 were already being used in the digital pin header. The LED drivers will be a permanent fixture on the PCB and will not be connected to this header. An array of solder points was created and arranged so that each row contains a point for red, green, and blue. 60 of the 64 solder points will be used for the 20 RGB LEDs and the remaining 4 could potentially be used for single color LEDs.

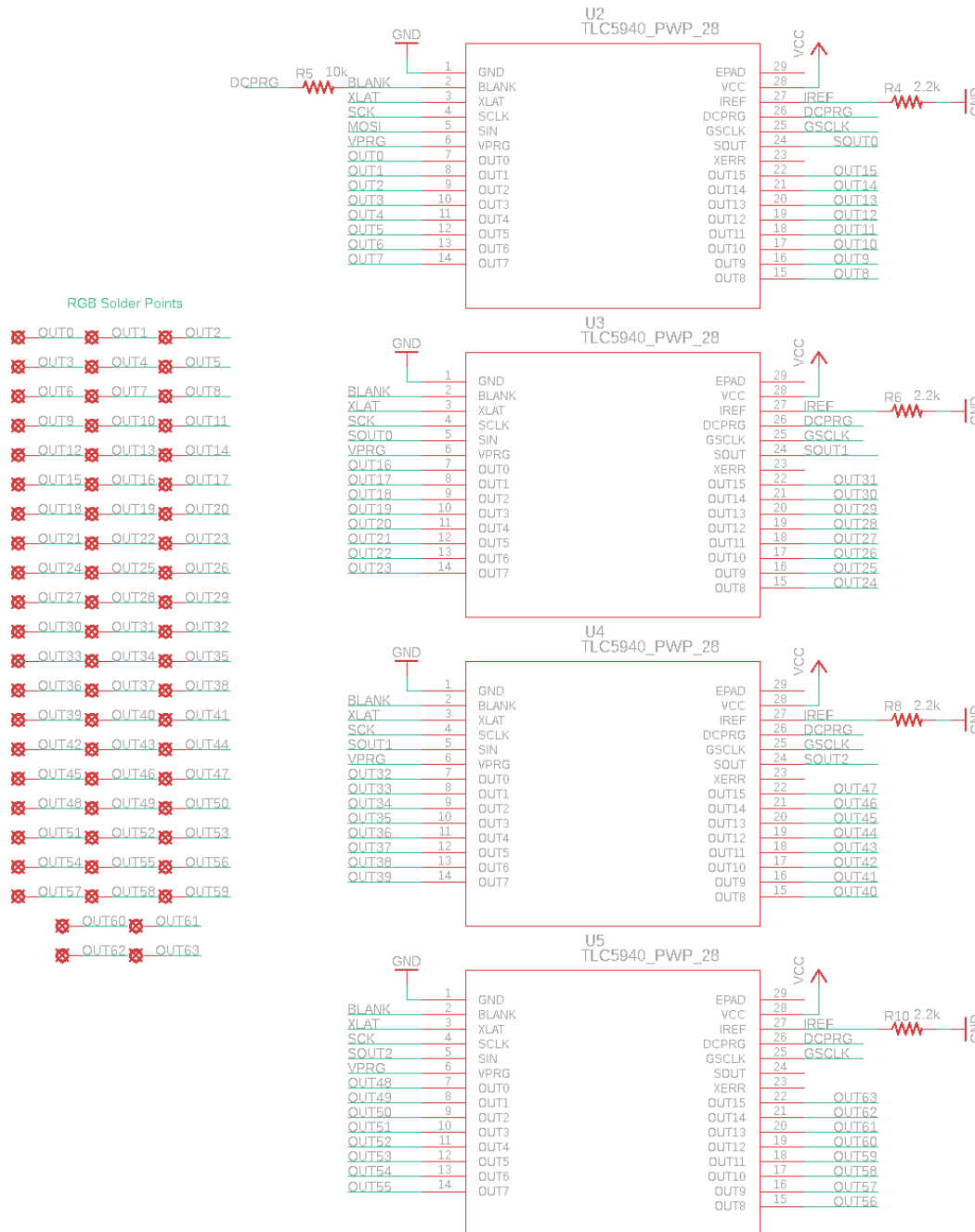


Figure 15: Effect Lighting Schematic

5.8 Sound Design

Sound effects will be used to complement character actions such as combat and using magic. These sound effects will accompany the LED effects. However, they will be controlled using software on the PC as opposed to being controlled using the MCU. This is mostly due to memory limitations with the MCU because sound files take up large amounts of memory. Free use sound effects will be used and if music is needed then royalty free music will be used. To play the sound effects speakers will be connected either to the projector or directly to the PC using a 3.5mm jack. Pressing an action on the user's phone will play the corresponding sound effect. A list of possible sound effects is given in Table 25.

Table 25: List of Possible Sound Effects

Action	Sound Effect
Melee Combat	Swords clashing
Ranged Combat	Bow shoots arrow
Casting Spell	Generic magic sound
Movement	Sound of footsteps

5.9 Power Supply

As we plan to use a commercial projector and laptop as components of our system, and we do not wish to put forth excessive effort designing a custom power source, running the risk of permanent damage to our most expensive components, the table will use a standard wall outlet for power. In order to convert power coming from the wall outlet to DC power an AC adapter is needed. For safety purposes instead of creating our own design an adapter will be purchased. To determine which adapter to buy the output voltage is compared to the highest voltage required by an individual component in the design. These components include the microcontroller, the LED drivers, the fans, and the 7-segment display. The projector will be connected directly to a power strip and will not be considered. Out of the previously mentioned devices the highest voltage requirement is 12V for the fans. Therefore, the AC adapter will need to supply 12 volts. A 12V power supply with a maximum current of 2A was selected. The device will connect to a power strip along with the projector and will be used to with a series of voltage regulators to provide the required voltages for all peripherals.

5.9.1 Voltage Regulator

The 12 volts coming from the AC adapter is too high to be used for the MCU and other peripherals. A step-down converter, also called a buck converter, will be needed to step down the 12 volts to 5 volts while also regulating this voltage. The Texas Instruments Webench® Power Designer gives recommendations for designing voltage regulators based on desired specifications such as input voltage, output voltage, and max current. Using the Power Designer multiple possible regulator configurations were compared to determine which one would best fit our design. Three step-down voltage regulator ICs, the LM2576, TPS562208, and TPS565201, were considered and are discussed below.

The LM2576 is a monolithic integrated circuit that provides the active functions for a step-down switching regulator and is capable of driving a 3A load. A switching regulator is different from a linear voltage regulator in that it uses a switching element, such as a MOSFET, to transform the incoming power supply into a pulsed voltage [17]. They are able to regulate the output more

efficiently than linear regulators and generate less heat. Out of the three ICs considered the LM2576 required the lowest part count making it the simplest choice. There is also familiarity with this IC due to using it in coursework.

The TPS562208 is a simple, easy-to-use, 2A synchronous step-down converter. It operates in force continuous conduction mode where the switching frequency is maintained at an almost constant level over the entire load range and is optimized to achieve low standby current. A synchronous buck converter is a modified version of the basic buck converter circuit topology where the diode is replaced by a switch. This is more expensive but improves efficiency. Overall the TPS562208 is a good option that balances cost, efficiency, and complexity.

The TPS565201 is another synchronous step-down converter. It has a higher maximum load at 5A and has higher efficiency than both of the other options with an estimated efficiency of 96.4%. It is optimized to operate with minimum external component counts and a low standby current. This device employs D-CAP2 control which provides fast transient response and requires no external compensation components. It also allows the use of low-equivalent series resistance specialty polymer capacitors and ceramic output capacitors. Like the TPS562208 it operates in pulse skip mode to maintain high efficiency even when operating with a light load.

Table 26 below shows a comparison of the three ICs that were considered. The LM2576 has the lowest BOM count, which is appealing, but lacks in efficiency. A low efficiency means the device will run hot and this is undesirable especially since the table will be in use for potentially hours at a time. The TPS562208 has a much better efficiency and has the lowest cost of the three devices. The number of external components required to use the device increases but the IC is smaller so the total area used actually goes down by quite a bit. Finally, the TPS565201 has the highest efficiency, has a higher max output current, and is only 74 cents more than the TPS562208. The high efficiency means the device will be able to run for long periods of time without generating too much heat and the higher max current gives us room to add more power consuming devices such as LEDs. Therefore, this is the IC that will be used for our step-down voltage regulator.

Table 26: Comparison of Voltage Regulators

IC	BOM Area (mm ²)	BOM Count	Cost (\$)	Efficiency (%)	V _{in} Range (V)	V _{out} range (V)	I _{out} Max (A)
LM2576-5.0	679	5	2.67	83.9	5.4 - 40	5	3
TPS562208	188	9	1.16	92.2	4.5 - 17	0.77 - 7	2
TPS565201	186	9	1.9	96.4	4.5 - 17	0.77 - 7	5

5.9.2 Voltage Regulator Design

Along with the TPS565201 IC external components such as resistors and capacitors must be used to give the desired output voltage. Using the recommended configuration from the Texas Instruments Power Designer a schematic was built in Eagle and is shown in Figure 16 A barrel jack is used to accommodate a 2.1 x 5.5mm plug. This will receive a regulated 12V from the AC adapter. The output voltage of the barrel jack leads to a toggle switch which will be used as an on/off switch for the MCU. When the switch is in the on state the 12V is connected to the input of the voltage regulator. Two parallel decoupling capacitors are used for the input and output of the regulator as recommended. The purpose of decoupling capacitors is to suppress high frequency

noise. An inductor is also placed at the output of the regulator. The main purpose of inductors in switching regulators is to maintain current flow during the off state. They can also be used to create a higher output voltage than input voltage but that is not relevant to this design. The output voltage is determined by Equation 1, found in the datasheet for the TPS565201:

Equation 1: Output Voltage Equation

$$V_{out} = 0.760 * \left(1 + \frac{R1}{R2}\right)$$

The datasheet explicitly states to use a 54.9k resistor for R1 and a 10k resistor for R2 to get an output voltage of 5V. Using the equation above, the output voltage with the recommended resistors will be about 4.93V. This is suitable for the desired applications. Two 4-pin headers are also included. These will serve as ports for peripherals such as the fans. This will allow us to easily remove and reconnect these devices by providing solderless connections as optimal positioning is determined.

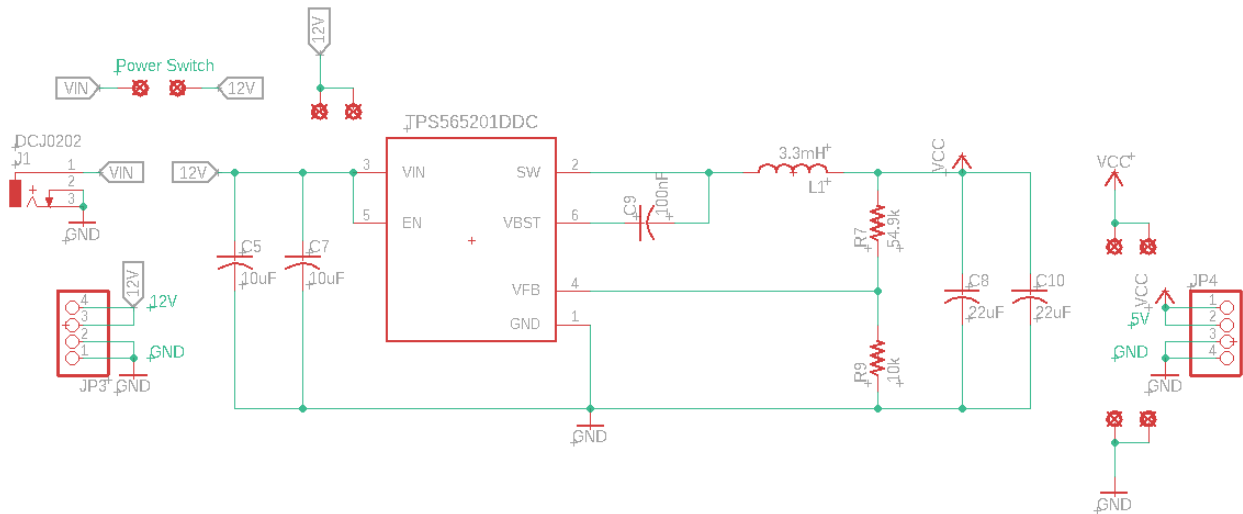


Figure 16: Schematic of Step-Down Voltage Regulator

5.10 PCB Design

After designing the subsystems for the PCB, a full schematic was created and is shown in Figure 17. An attempt was made to locate devices near their connections to the MCU.

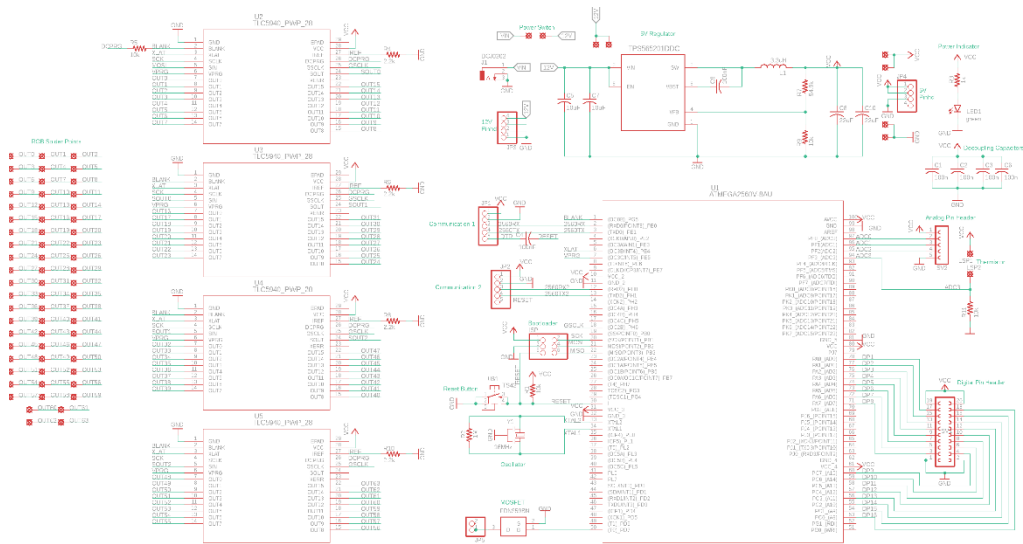


Figure 17: PCB Schematic

A PCB layout was designed and is shown in Figure 18. Initially parts were placed in the desired spots on the board and then the auto routing feature of Eagle was used. Traces were then modified until no violations were observed. Four 40mm holes were added that will be used to hold the board within the table. The design check tool in Eagle was used to make sure the design could be manufactured, and no errors appeared.

A possible vendor is still being researched and will be decided between semesters. The most likely candidate is JLCPCB due to its cheap pricing. However, factors such as part availability must be considered still. Parts were selected through the design link in Eagle and a bill of materials was created. Parts may be changed in the near future due to availability.

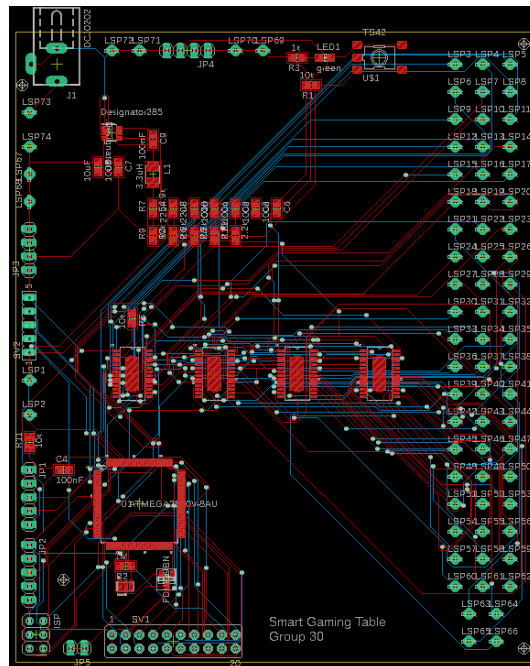


Figure 18: PCB Layout

5.11 Serial Communication Protocols

Serial communications are a way for electronic devices to communicate with each other by streaming data one bit at a time using a hardwired connection. The main benefits of using serial communications over parallel communications, which transfers multiple bits at the same time, is that it requires less input/output (I/O) lines and is easier to implement. There are two types of serial communications: asynchronous and synchronous. In asynchronous communications the data is transferred without support from an external clock signal which means the rising and falling edges are not guaranteed to coincide. For synchronous communications all devices share a common clock by always pairing data lines with a clock signal. This is more straightforward and faster than asynchronous communications but requires at least one extra wire between communicating devices. Three types of serial communication protocols are explored below: UART, I²C, and SPI. The advantages and disadvantages of each protocol are then compared in Table 28.

5.11.1 UART

Universal Asynchronous Receiver and Transmitter (UART) is a simple asynchronous communication protocol commonly found inside microcontrollers. There are three ways it can operate between devices: simplex, half duplex, and full duplex. In simplex mode data is only transmitted in one direction using one wire. In half duplex mode data is transmitted in either direction but still uses one wire so the devices must take turns transmitting and receiving. Finally, in full duplex mode two wires are used so data is transmitted in both directions simultaneously. Figure 19 shows a transmission pattern for 0X1F using UART. The scheme works like this: the line is idle at high, the line drops to low for one bit duration to signal the start bit, the data is transmitted one bit at a time, and finally, a stop bit with a value of high is transmitted to signal the end of the transmission. The bit duration is defined by the transmitter's clock rate and is known as the baud rate. The most commonly used baud rate is 9600 which means a bit lasts 1/9600 seconds. Table 27 shows a list of parameters for UART and the most popular configuration for those parameters.

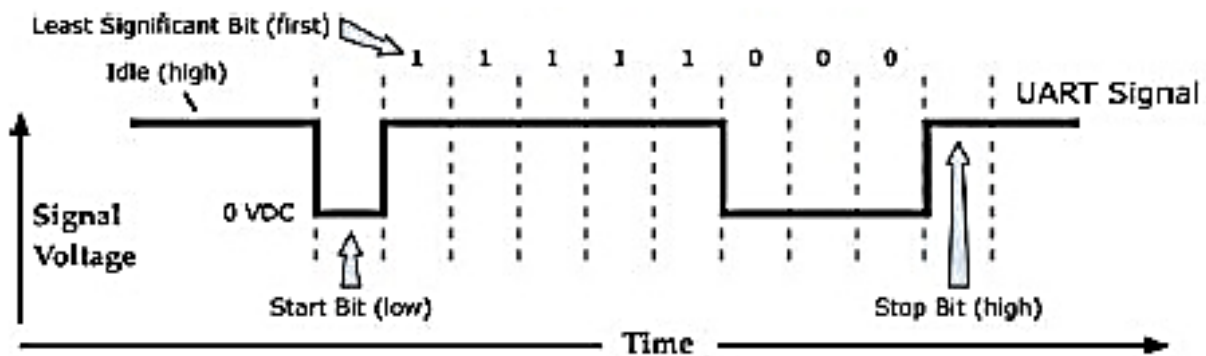


Figure 19: UART Operation

Reproduced in accordance with the Creative Commons Attribution-Share Alike 3.0 Unported license

Table 27: UART Parameters

Parameter	Meaning	Popular Configuration
Baud rate	Transmission speed	9600
Data size	Number of bits	8-bit
First bit	Significant bit	LSB
Parity	Bits to detect errors	None
Stop bit	Signals end of transmission	1-bit
Flow control	Mechanism to pace transmission	None

5.11.2 I²C

Inter-Integrated Circuit (I²C) communication is a synchronous serial communications protocol similar to UART mainly used with modules and sensors. It is based on a bus topology and has two wires: Serial Data (SDA) and Serial Clock (SCL). The clock line is used for synchronizing transmission and the data line is the line through which bits of data are sent or received.

In the bus topology all devices plug into the same set of wires as shown in Figure 20. This topology is useful because it allows multiple devices to be connected to the bus. Each device is assigned a 7-bit address so they can be distinguished from each other. One device is designated as the master and the other devices are designated as slaves. The master initiates all transmissions, reads from or writes from the other devices, and is responsible from driving the clock signal.

The SDA and SCL lines are pulled up via pull-up resistors and read high if no action is done. To start and end transmissions the master uses start and stop signals. These are unique signals that can't occur during the data bits. To begin transmission the master transmits the start signal and then sends each slave the 7-bit address of the slave and a read/write bit to the slave it wants to communicate with. The slave compares the address with its own and if there is a match the slave returns an ACK bit which switches the SDA line to low for one bit. If there is not a match the SDA line remains high. The master then sends or receives the 8-bit data frame. After each data frame has been transferred the receiving device returns another ACK bit to acknowledge successful transmission. To end transmission the master sends a stop signal by switching SCL high before switching SDA high.

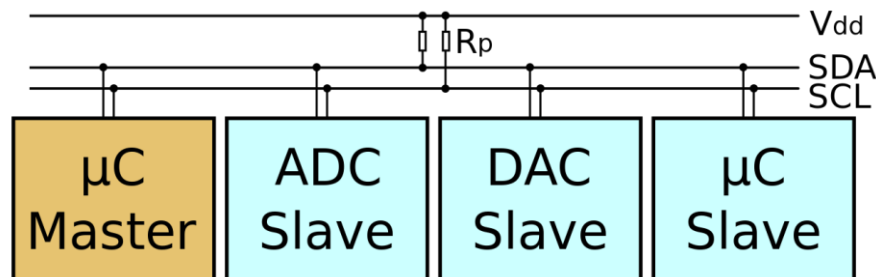


Figure 20: I²C Configuration

Reproduced in accordance with the Creative Commons Attribution-Share Alike 3.0 Unported license

5.11.3 SPI

Serial Peripheral Interface (SPI) is another synchronous serial communications protocol similar to I²C that is specifically designed for microcontrollers. It operates at full duplex where data is sent and received simultaneously using two data wires. A fourth line known as the chip select is used to choose which device is being communicated with. The devices that are not selected place high-impedance on their data out lines to not interfere. Therefore, the SPI interface uses four wires: Serial Data Out, Serial Data In, Serial Clock, and Chip Select. When the master interfaces with only one device the device's chip select signal can be connected to low to save a pin at the master in what is known as 3-pin SPI. SPI is not an official standard so the names of each wire may be different. For example, the data lines are sometimes called Master Out/Slave In (MOSI) and Slave In/Master Out (SIMO). Figure 21 shows an example of three devices using SPI in a daisy chain configuration (output of one device is wired to input of another device).

SPI is implemented using two shift registers: one at the master and one at the device. Communication works by exchanging the contents of the two shift registers. Bits coming out of the master's shift register goes into the device's shift register and vice versa. Transmitting a byte consists of latch/shift actions repeated eight times. There are four modes of operation based on combinations of the clock polarity and what triggers the latch and communicate actions (clock phase). The two types of polarity are clock idle at low and clock idle at high. The two types of clock phase are latch at trailing edge/communicate at leading edge and latch at leading edge/communicate at trailing edge.

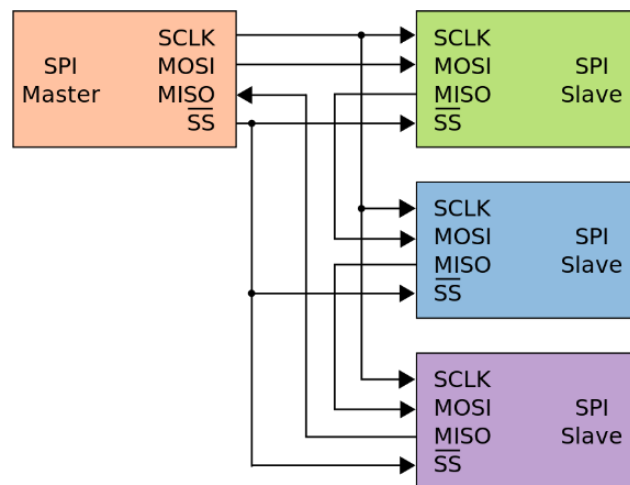


Figure 21: SPI Configuration

Reproduced in accordance with the Creative Commons Attribution-Share Alike 3.0 Unported license

5.11.4 Summary

The advantages and disadvantages of the three serial communication protocols discussed are summarized in Table 28. Understanding these protocols is useful for this project because there are many different electronic devices being used, each with different protocol compatibilities. Being able to communicate between these devices to create a coherent product is essential.

Table 28: Serial Communications Summary

Protocol	Advantages	Disadvantages
UART	Simple to operate	Size of data from is limited to 9 bits
	Well documented	Cannot use multiple master systems and slaves
	No clock needed	baud rates of each UART must be within 10% of each other to prevent data loss
	Parity bit allows for error checking	Low speed
I2C	Simple to operate	Limited speed
	Low pin/signal count even with numerous devices	Requires space on PCB for resistors
	Supports multi master and multi slave communication	Becomes complex when many slaves are implemented
	Adapts to needs of various slave devices	
SPI	Faster than asynchronous serial	More pins needed than other protocols
	Receiving device can be as simple as a shift register	No acknowledgment mechanism
	Supports multiple slaves	No form of error check
	Data is transmitted continuously (no start and stop bits)	Only one master allowed. Slaves cannot communicate with each other.
	Data can be transmitted and received simultaneously	

5.12 Wireless Communication Protocols

The smart table will require the ability to have multiple smart phones connected wirelessly and simultaneously. Information such as player actions and character statistics will need to be sent from the phones to the table with as short of a delay as possible. Two possible wireless technologies, Bluetooth 5.0 and IEEE 802.11ac (Wi-Fi 5), are compared and a decision is made on which technology will be incorporated into our design. These technologies were selected because they are both used by most modern mobile devices.

5.12.1 Bluetooth 5.0

Bluetooth is a wireless technology that uses ultra high frequency radio waves to connect fixed and mobile devices over short distances. It operates in the 2.4 to 2.485 GHz ISM band and uses frequency hopping, which is a technique where the signal moves from one frequency to the next at regular intervals. The data is split into portions called packets and is transmitted across 79 bands that are 1 MHz in size. This technique allows transmissions to avoid interference from other signals using the same frequency band such as Wi-Fi. The hopping occurs at 1600 times per second and hops over all of the available frequencies using a pre-determined pseudo-random hop sequence based on the address of the master node in the network.

Bluetooth uses a master-slave structure where one master may communicate with up to seven slaves in something called a piconet. Packet exchange is based on the master's clock which all the devices share. Multiple piconets can be connected to form a scatternet where devices can simultaneously be a master for one network and a slave for another. This network configuration is shown in Figure 22. Bluetooth pairing is a scheme that allows devices to connect easily and

quickly. Pairing is usually initiated manually by a device user and a link is made visible to other devices.

Bluetooth standardization was previously defined by IEEE 802.15.1 but is now managed by the Bluetooth Special Interest Group and to market something as a Bluetooth device a manufacturer must meet the Bluetooth SIG standards. Bluetooth 5.0 was first presented by the Bluetooth SIG in June 2016. It mainly focused on improving technology for Internet of Things (IoT) and boasted four times the range, two times the speed, and eight times the broadcasting message capacity of older versions of Bluetooth.

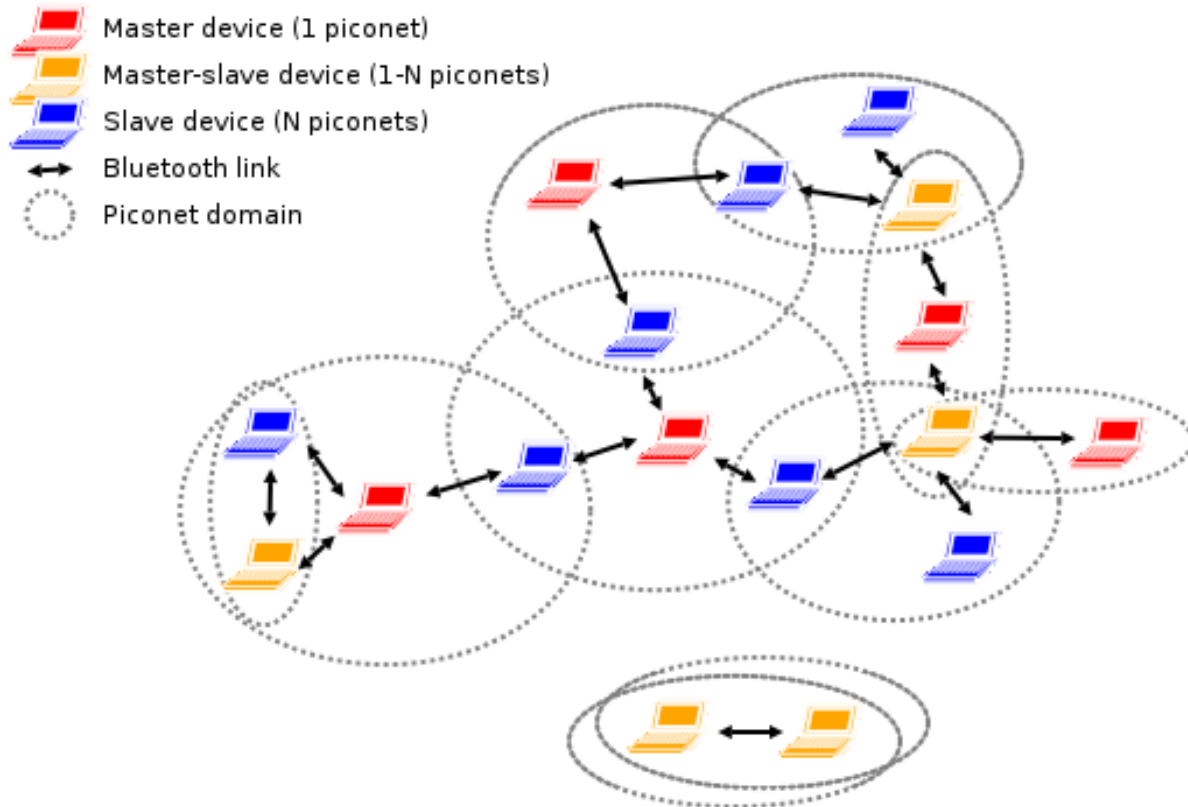


Figure 22: Bluetooth Scatternet Configuration

Reproduced in accordance with the Creative Commons Attribution-Share Alike 2.5 Spain license

5.12.2 Wi-Fi IEEE 802.11ac

Wi-Fi is a radio frequency technology based on the IEEE 802.11 [2] family of standards. It most commonly uses the 2.4 and 5 GHz super high frequency ISM bands. It is a major form of communication used within most homes for connecting to the internet. Wi-Fi uses an access point that acts as the base that communicates with Wi-Fi enabled devices. The data is then routed onto a local area network normally via ethernet. Home Wi-Fi systems often use an Ethernet router that provides the Wi-Fi access point and links to the internet via a firewall. Many routers now provide dual band Wi-Fi connectivity and automatically select the optimal channel and frequency (2.4 or 5 GHz).

There are two basic Wi-Fi network types: local area network (LAN) and ad hoc network. In a LAN based network an access point is linked onto a local area network to provide wireless as well as

wired connectivity. In many cases only wireless connections are used in what is known as a Wireless Local Area Network (WLAN). An Ad hoc network allows devices to connect directly without the use of a server. In this case users communicate with each other and not with a larger wired network. One of the peripherals takes the role of master and the others act as slaves. Ad-hoc mode is also known as Peer-to-peer mode. Another standard called Wi-Fi Direct builds on ad-hoc mode and makes it easier to discover and connect to nearby devices. Wi-Fi Direct allows two devices to establish a direct Wi-Fi connection without requiring a wireless router. This is a single hop communication as opposed to the multihop used by ad hoc networks. This mode is the most similar to Bluetooth.

802.11ac is a variant contained within the 802.11-2016 standard that was released in 2013. The organization Wi-Fi alliance labeled this standard Wi-Fi 5. 802.11ac improves upon 802.11n (Wi-Fi 4) in the following ways:

- More channel bonding, increased from 40 MHz to 160 MHz.
- Denser modulation, using 256 Quadrature Amplitude Modulation (QAM), up from 64QAM.
- More Multiple Input Multiple Output (MIMO), using eight spacial streams, up from four.

Wireless speed is the product of three factors: channel bandwidth, constellation density, and the number of spacial streams. 802.11 ac increased the boundaries of these parameters to allow for much greater speeds (1300 Mbps up from 450 Mbps). However, this protocol only works in the 5 GHz band which lowers the range.

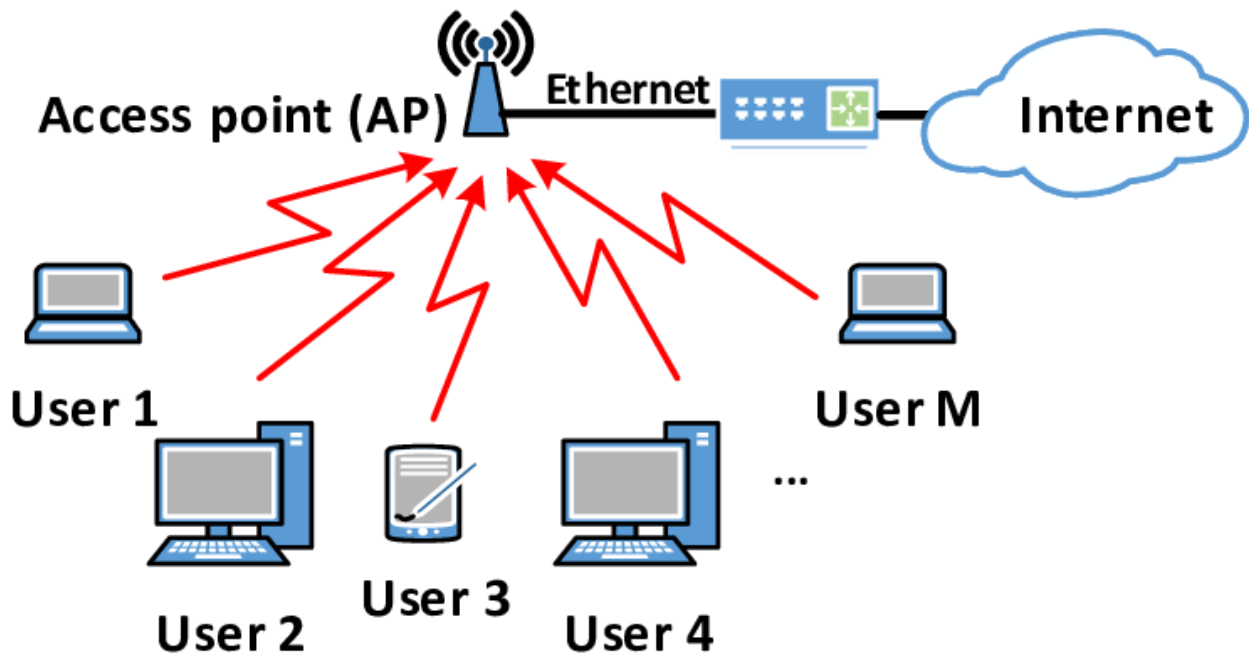


Figure 23: LAN Network

Reproduced in accordance with the Creative Commons Attribution-Share Alike 4.0 International license

5.12.3 Wireless Technology Comparison

Bluetooth and Wi-Fi technologies are compared to determine which one will best support our use case. In our case the cost can be neglected because the computer used supports both Bluetooth and Wi-Fi, and the smartphones are guaranteed to have both these technologies. Power consumption can also be ignored for the purposes of comparison because the table will be connected to a wall outlet. The ranges for both protocols are plenty sufficient for the purpose of this project. A benefit of Wi-Fi is that it boasts less latency and a higher data rate compared to Bluetooth. This is important because there should be as little of a delay as possible when performing actions. Therefore, Wi-Fi will be used to connect the smartphones to the table. A summary of the comparisons between Wi-Fi and Bluetooth are shown in Table 29.

Table 29: Wireless Communication Protocol Comparison

Specification	Bluetooth 5.0	Wi-Fi IEEE 802.11ac
Frequency	2.4 GHz	5 GHz
Cost	Low	High
Channel Bandwidth	1 MHz	22 MHz
Range	5-30 meters	20m indoors
Power consumption	Low	Medium
Latency	200ms	150ms
Bit-rate	2.1Mbps	600 Mbps

6.0 Software Design

6.1 Software Overview

The tabletop game software will be comprised of three different subsystems. As seen in the detailed software block diagram in Figure 24, the subsystems will be the object detection, game software, and the android app. The game and object detection subsystems will both be located on the computer system. The different subsystems will run on different threads to allow for simultaneous execution. Specifically, this will enable the object detection subsystem to generate locations while the game software handles requests from the mobile app subsystem. The game subsystem will receive player piece location data from the object detection subsystem, and it will receive explicit user commands through the connection with the app. The game software needs to be able to maintain the location of each player, maintain the location of all digital non-player controlled (NPC) entities, maintain a connection to all players, send the correct images to be displayed by the projector, and manage all request from users. The object detection software uses a video stream provided by a camera and converts that stream into location data that is sent to the game software. The video stream will be facing the bottom of the display and looking for blobs that contrast with background illumination to distinguish objects. The user app is the endpoint where users are able to directly communicate with the software. The app acts as a replacement to the character sheet which is traditionally played using a physical piece of paper. The character sheet is the location of all the character information. The app will contain functionality to turn what would normally be spoken word in the game to digital actions on the display. Communications between the app and computer systems will be carried out using the Wi-Fi Direct protocol.

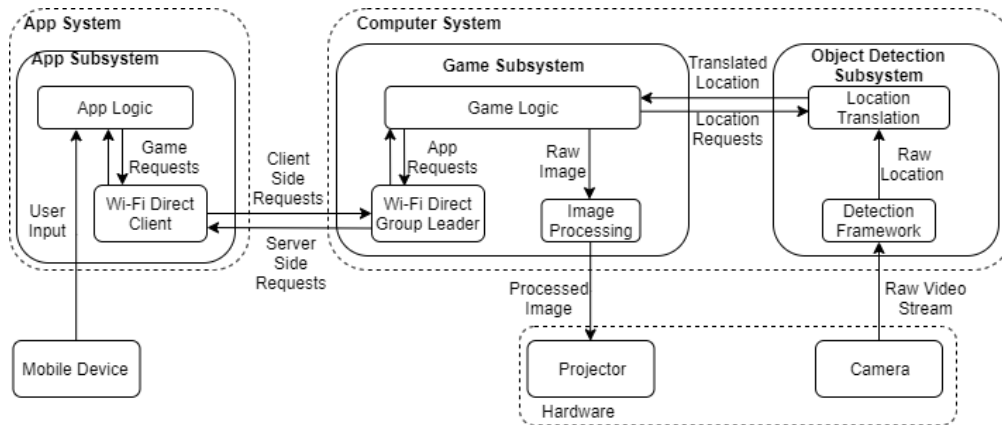


Figure 24: Detailed Software Block Diagram

6.2 Object and Touch Detection

Figure 25 displays the use case for the object detection subsystem. The key feature for this subsystem will be the ability to track user pieces on a display. This can be accomplished by using third-party open source object detection framework. The framework must be able to track cursors and blobs. For each tracking object the framework must be able to transmit the data readings. The input to this subsystem will be a camera sensor and the output will be the object detector class. This class will be created to handle data transfer and manipulation for the project’s software. The third-party framework must accommodate the rear DI tracking scheme as that is the scheme used to track objects in this project.

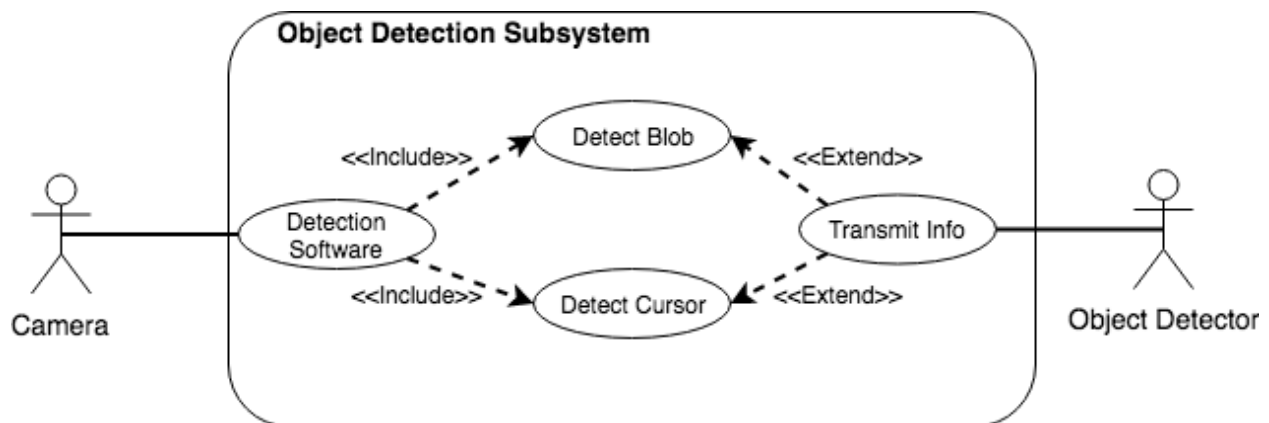


Figure 25: Object Detection Use Case Diagram

6.2.1 Framework Discussion

6.2.1.1 CCV

The CCV (Community Core Vision) [18] framework was developed by a group called The Natural User Interface Group. The current version was released in November of 2014. They are a community dedicated to open source software related to research into human computer interactions. CCV provides a solution to tracking objects. After some configurations, the software ideally will spot white blobs through a video input stream where there is no background noise.

This applies directly to the rear DI scheme because the video input will be the display where only touches or objects on the display will reflect enough light to become a blob.

The software requires a Pentium 4 processor of better, 512 MB of ram, a camera, and windows QuickTime. There is a list of supported cameras for the software. They do also include a more option implying that any camera can work with the correct configurations. The software does support GPU acceleration to improve latency and capacity of tracking objects. The recommended GPU is listed as a modern GPU which can be assumed to be a GPU made any time after the software was created. The software does offer cross platform support to for Windows, Mac, or Linux systems. The original development platform was Windows and Windows also has the most amount of stable releases.

The software provides a graphical user interface that displays the video input and the tracked object output. However, for this project only the raw data is required as tracking of objects will be done within the software. The interface will be useful for testing the hardware setup for the display. There is a method that allows object locations to be outputted as raw coordinates. Those raw coordinates will be translated to more useful values in the project's software. The software also tracks the size of each blob that is detected and output those values as well. The sizes can be used to identify what an object is; however, for this project the sizes will be used to better determine the location on the game map.

6.2.1.2 Scene

Scene [19] is an open source free to use software that tracks objects. It comes with a graphical user interface as a way to show the manipulations that the software has done to the input frames. Similar to CCV, the interface will be used for testing purposes rather than directly implemented into the software. The interface design was actually provided to Scene by CCV.

The software is developed to take any video stream input and process those frames through different techniques to determine the difference between the foreground and the background. After the foreground and background objects are sorted, the background is eliminated. Blobs are then formed using only the foreground elements of the video stream. The blobs will be packaged and transferred through the TUIO framework. The video source that the software will receive in this project will be an already black background with only the blobs in the foreground.

The techniques used to determine and eliminate the background elements of the video stream are simple gaussian, fuzzy gaussian, or mixture gaussian techniques to remove background objects. They are referred to as subtraction background methods. These methods use statistical analysis of image matrices to determine what pixels belong to the background and foreground. Of the three methods, fuzzy gaussian is the only method that does not eventually incorporate stationary objects into the background. However, simple and mixture gaussian are able to pick up on and track moving objects with accuracy. There are also two self-organizing background subtraction algorithms options; adaptive and fuzzy adaptive. These options use neural networks to determine what pixels should be considered foreground and background.

6.2.1.3 TouchLib

TouchLib [20] is a free to use library to implement a multitouch interactive display. The website for TouchLib explicitly says that the library works for FTIR and DI configurations. This library is provided by The Natural User Interface Group who also create CCV. The last published version of the library was in February of 2016. This option does not come with a graphical user interface.

The library provides a configuration application to properly set up hardware for accurate touch readings. Demos are provided as examples on how to create different applications with the framework. The benefit of using a library would be the flexibility to create an application that only functions for a specific purpose. The solution will be smaller and more efficient in terms of computations. Testing will become more difficult with the lack of a graphical interface. Instead, methods will have to be created in order to test functionality of the software. Development time will be increased with this option over an already built application. Similar to the other object detection software options, transmission of blob objects is done through TUIO.

6.2.1.4 Conclusion

Scene is a robust solution for the problem that is present with our project. The images that will be processed will already have the background eliminated and blobs desired blobs brought to the foreground. The background elimination techniques will be unnecessary and to save computation should be disabled. By disabling them the solution becomes functionally the same as CCV. Unexpected difficulties could come from modifying the software to the extent that would be necessary to eliminate the irrelevant functionality. TouchLib is a tempting choice because it provides freedom to create apps that only include features required. Development time will increase with more software needing to be created and tested. The last version of the library being from 2016 is troublesome in that support may not be readily available and any complications may result in major time loss. If time was not a constraint, this option would be explored for the customization ability that it inherently brings. A fully developed and tested software will cut down on development time. The only time requirement will be to setup the configurations for the hardware system and test to make sure that the software is reading the input stream and providing an output stream. CCV does just that and provides the desired functionality for DI. There is no need to create or repurpose software because the package has everything included. The only additionally software needed will be to set up the other subsystems to allow for TUIO communication. This software would have been developed regardless of the decision as each option uses the TUIO protocol. CCV will be chosen to cut down on software development time for design, implementation, and testing.

6.2.2 TUIO

The TUIO (Tangible User Interface Objects) [21] protocol is an open source framework. The framework is free under the less restrictive L-GPL license. The protocol defines two different message classes; Set and Alive. Set details location data for current alive objects. Attributes such as coordinate location, orientation, and directional velocities are recorded and sent in these messages. Alive messages show the current alive objects being tracked; therefore, comparing consecutive readings can identify the addition or removal of objects from the display. This allows the software to not create add or remove messages to reduce the number of messages being sent. This will avoid information loss due to packet loss through transmission. Transmission is also defined in this protocol to be through a TCP connection.

The CCV software utilizes this framework to package and transmit object detected. This format provides dimensional values that describe an ellipse that is created within the bounding box of the blob. The bounding box is way for the software to enclose the blob detected into a range of pixels. The box will have a width and height equal to the maximum dimensions of the blob. The center point of the box is the center point of the blob and it is with respect to the top left corner of the image. The top left corner of the image is located at (0,0). After the bounding box is created, an ellipse is inscribed, and the area of this ellipse will be the approximate area of the blob. The

bounding box is oriented based on the direction that the object is placed on the display. An angle will be associated with this bounding box and dimensions will be normalized after a rotation for the box.

Tuio objects need to provide a variety of information about different attributes that describe the ellipse approximation. This project requires the ability locate objects at more than just the center point of that object, but also all surrounding pixels that the object overlaps into. Therefore, the (x,y) coordinates for the center point of the blob as well as the ellipse dimensions will be needed. All of these attributes are normalized and returned as floating-point values.

Tuio does not maintain the exact values for the height and width because the Tuio objects provide the necessary values to allow the programmers to calculate it themselves. The coordinate normalization is done using Equation 2, found on the TUIO website, taking the sensors location and dividing it by the corresponding sensor dimension. This will produce a floating-point value that will be between 0 and 1.

Equation 2: TUIO (x,y) Coordinate Normalization

$$x = \text{sensor_x} / \text{sensor_width}$$

$$y = \text{sensor_y} / \text{sensor_height}$$

The (x, y) coordinates provide the location of the center of the blob. This location can be used to determine what grid space the center of the blob is located in. For the majority of cases, the center point location will be enough to determine the grid space the player piece is meant to be in. However, there can be times that the center point is on the grid lines or close enough to cause issues. The solution to this problem would be the area of the blob.

Similar to the coordinates, this area value returned will be a floating-point value. The value is found using the normalized width and height as well as the approximate area of the blob in pixels. Equation 3, found on the TUIO website, shows how the area is achieved.

Equation 3: TUIO Area Normalization

$$\text{Area} = \text{Pixels} / (\text{width} * \text{height})$$

TUIO uses inheritance to create a set of related classes for maintaining detected object states. Blobs are maintained in a TuioBlob objects. This object inherits attributes and methods from the TuioPoint class. Specifically, the TuioPoint class maintains fields for the object as if it were a single point on the display. The fields track the center point of the blob. The TuioBlob class maintains all the values need to describe the ellipse approximation.

6.2.3 Object Detector Class

CCV is the open source software used for this project. The software incorporates the TUIO API to transmit objects. On the framework's end, a TuioClient object is used to transmit blobs. The blobs are sent through a TCP socket using the TuioSender object that is instantiated by the TuioClient. To read the broadcast, an ObjectDetector class will be created. The object detector class will also use the TUIO API and generate a TuioClient that adds a TuioListener. This listener will receive the blobs by creating a TCP socket endpoint. The transmitted blobs will then be placed into local fields that will be manipulated by a decoding method. The high-level flow of communication can

be seen in Figure 26. The transfer of blobs begin with the video stream input and end with the getLocation() function implemented in the ObjectDetector class.

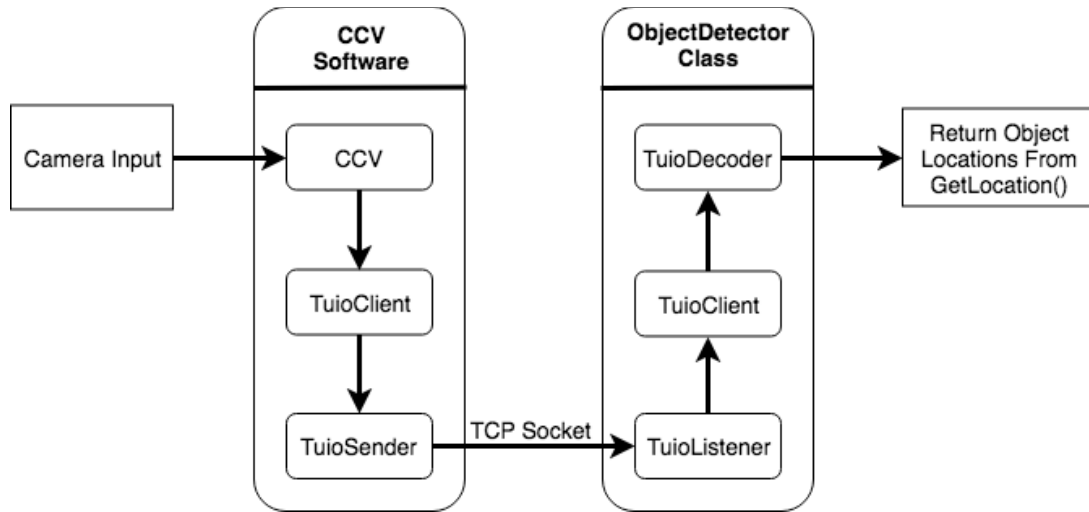


Figure 26: CCV and ObjectDetector Communication

There are a few methods that need to be implemented for the TuioListener object to function. These methods are shown in Table 30. Not all methods will be programmed to have functionality. This project’s software will only require the methods related to Tuio blobs; therefore, methods related to Tuio objects can be left empty. The only functional requirement for these methods is that the methods are defined in our implementation.

Table 30: TuioListener Methods

Method	Method Purpose
addTuioObject(TuioObject *tobj)	Handles objects becoming visible
removeTuioObject(TuioObject *tobj)	Handles objects being removed
updateTuioObject(TuioObject *tobj)	Handles moved objects
addTuioCursor(TuioCursor *tcur)	Handles new cursors
removeTuioCursor(TuioCursor *tcur)	Handles removing cursors
updateTuioCursor(TuioCursor *tcur)	Handles moving cursors
addTuioBlob(TuioBlob *tblb)	Handles newly detected blobs
removeTuioBlob(TuioBlob *tblb)	Handles blobs being removed
updateTuioBlob(TuioBlob *tblb)	Handles moving blobs
refresh(TuioTime bundleTime)	Marks end of a Tuio Transmission

The constructor will setup the Tuio objects to initiate communication with the CCV software. When blobs are received by the Tuio protocol, they will be passed into the blobDecoder method that will translate the values within the blob into usable game values. After the translation, that location will be stored in another field that will be accessed by the game software when needed. Table 31 shows the methods needed by the object detector class to translate the TUIO object into usable values for the game. The vector of locations will be encapsulated to avoid any unnecessary manipulation of the values. Therefore, a getLocation() method will be used to retrieve these locations. The constructor will setup the Tuio objects to initiate communication with the CCV

software. When blobs are received by the Tuio protocol, they will be passed into the blobDecoder method that will translate the values within the blob into usable game values. After the translation, that location will be stored in another field that will be accessed by the game software when needed.

Table 31: Object Detector Class Methods

Method	Method Purpose
ObjectDetector(): void	The constructor that boots up the TuioListener and prepares the game to receive input from the object detection software.
blobDecoder(std::list<TuioBlob *>): void	A method that decodes the normalized Tuio Blobs into grid values.
getLocation(): std::vector<Location>	The method will return a vector of blob locations.

Table 32 references the encapsulated fields that will be maintained by the object detector class. These fields will be used to create the ability to communicate with CCV and store the messages. The TuioClient will create the connection with the CCV software on the default port of 3333. The listener will be the object to receive communication data. This data will be stored in the encodedLocations list. After decoding, the values will be stored in the locations list.

Table 32: Object Detector Class Fields

Attribute	Descriptions
Private TuioClient client	The object used to connect to CCV.
private TuioListener listener	The object that listens for communication.
private std::vector<Location> locations	The vector of the decoded blob locations.
private std::vector<TuioBlob> encodedLocations	The vector of encoded blob locations.

6.2.3.1 Blob Decoder

The blob decoder method will take in as an input a set of blobs that represent the current array of objects on the display. The method will return a set of coordinate points that can be used to distinguish a location on the display. The location coordinates within the blobs are normalized using the sensor's width and height. To decode these normalized points, the points will be multiplied by the sensor's width and height. These values will be known constants that can be easily accessed by the object detector class. The TuioBlob object has the location coordinates as well as blob dimensions saved as attributes. The attributes are private members of the class that can only be read using the no argument get methods provided. The blobs that are being decoded contain all the information required to decode them.

6.3 Mobile App

The mobile app software is the input endpoint point for a player's gameplay decisions. The decisions will be collected through a series of menus that are displayed depending on the current state of the game. For the normal player, those states will be either character development or action. During an action state the player will have the decision to move or attack. A selection will need to

be made as well as confirmation of the physical action. The Game Master will at times act as a normal player. This implies that the Game master will have all the menus that the normal player has as well as additional menus to help with the logistics of the game. The Game Master requires the ability to initiate the save game protocol and change map as well as create and place NPCs. The use case for this subsystem can be seen in Figure 27. All-important functionality that will be implemented is displayed.

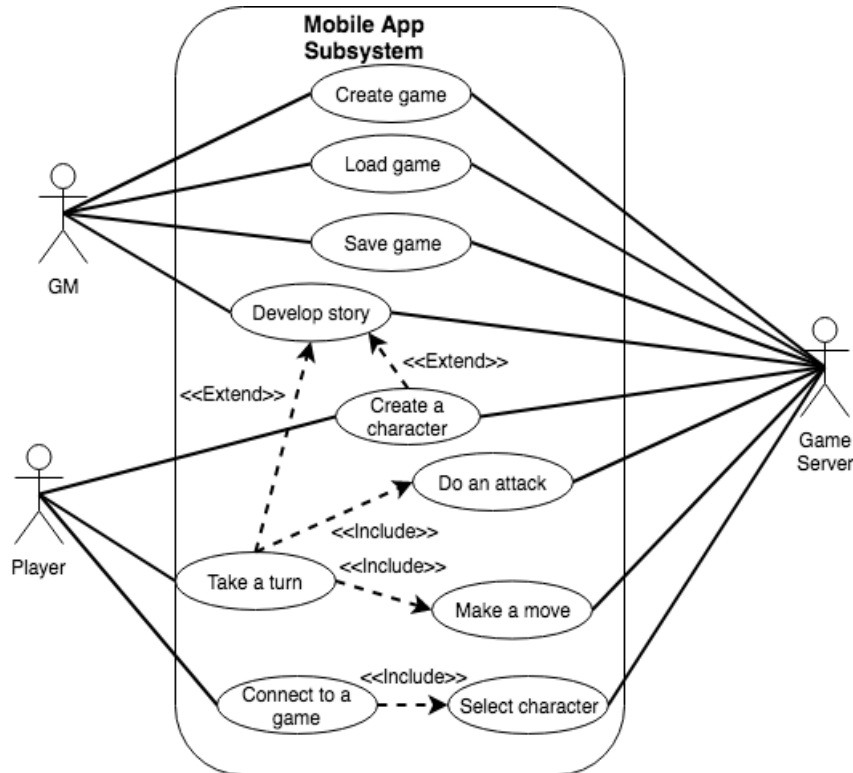


Figure 27: Mobile App Use Case Diagram

The app will launch into a visual of the app’s logo. In the background, the main menu components will be populated by a start-up procedure. Once the image is fully created, it will be shown to the user. The first menu that will appear for all users will be the option to create a new campaign, load an already existing campaign, or connect to a currently active game server. Each option will launch a different set of menus to set up the game. For the create a new campaign option, the creator will be the Game Master for the campaign and be required to input some information. The information will be basic characteristics for the campaign such as the number of players playing, name of the campaign, the starting map, and the starting location. After the information menus are completed, the game will create a server and allow other users to join. For every other player, they will select the connect to a server option. This option will then create a link to the game. The next set of menus will depend on the state of the campaign. If the campaign is newly created, then the players will begin a character development process. If the campaign already exists, then the players will see a selection of character names that they can select from. The player’s mobile device will then be linked to that character. The last option will be to load an already existing campaign. This will take the game files stored on the Game Masters device and load them to the game server. The server will then broadcast to other players and create the communication links. During gameplay, a menu will be shown that provides different options for the player to select from. Depending on

the choice, the app will request the game to do a specific action. Table 33 lists the major methods for the app and their purpose.

Table 33: Mobile App Methods

Method	Method Purpose
startup(void): void	Creates the main menu screen
createGame(void): void	Creates a menu to record game data. Sends game data to server to generate game.
loadGame(void): void	Creates a menu to select save data. Load save data to server to generate game.
saveGame(void): void	Starts the save game protocol.
connectToServer(void): void	Connects mobile device to server.
createCharacter(void): void	Creates a menu to create a character.
turnChoice(void): void	Creates a menu to choose turn action.
developStory(void): void	Create menu of options for GM to use to develop campaign story.

The fields for the app, detailed in Table 34 will be used to maintain different components as well as the specific subclass that the device is utilizing. While at the main menu, a device will only have an instance of the basic app user interface. Only after a game has begun will a player have an instance of the GM or Player class. These classes will define more menus that will be displayed. A device will only ever have an instance of either the GM or the Player class. There will never be a time where both exist. This allows for security that no player can hijack the game by gaining access to the GM interface. Other fields for the player stats will be maintained to create faster collection of data when communicating with the game software.

Table 34: Mobile App Fields

Attribute	Descriptions
private GM gmInstance	The instance of the GM interface class.
private Player playerInstance	The instance of the player interface class.
private Widget widgets[*]	A list of all interface components.
private Int stats[*]	A list of all stats being displayed and maintained.

The methods and fields can be split into three different classes. The parent class being the overall app user interface with two subclasses for the GM and the player. Regardless of the type of being a player or GM, each user will interact with the main menu. Only after the game has been started will the menus differ. At that point, the different subclasses will be instantiated. The differences in the menus is accompanied with difference in functionality. As seen in Figure 27, the GM has the ability to create, load, save, and develop the game. This functionality must be limited to the GM of the game. The separation of the methods and fields discussed above can be seen in the class diagram in Figure 28.

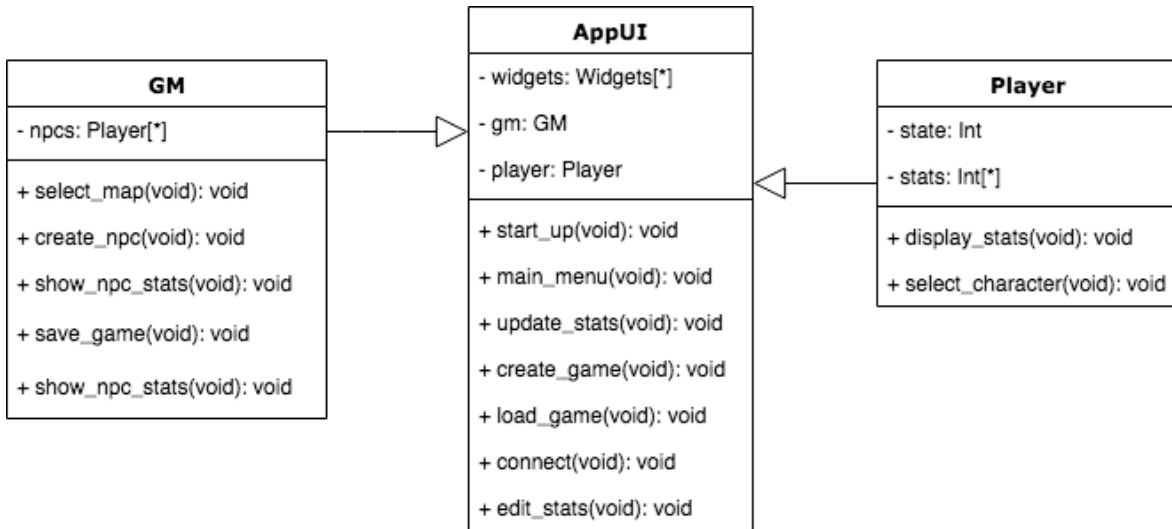


Figure 28: Mobile App Class Diagram

The mobile app will remain looping through a set of menus. Each player will only have access to turn actions when their turn is active. The state diagram seen in Figure 29 shows the relationship between the different menus. To move from the start menu to the idle menu the game will begin. The idle menu will be the menu that will be seen while not actively doing any actions. The state will only return back to the start menu when a game has ended. On a player's turn, the player will choose a primary turn action. This will lead the state to either an attack menu or a move menu. The player will then have the choice to end their turn or conduct a secondary action. After the secondary action, the turn will end resulting in the idle menu. Also, the player will have the option to update their player stats. After any modifications, the app will confirm the changes and return the user to the idle menu.

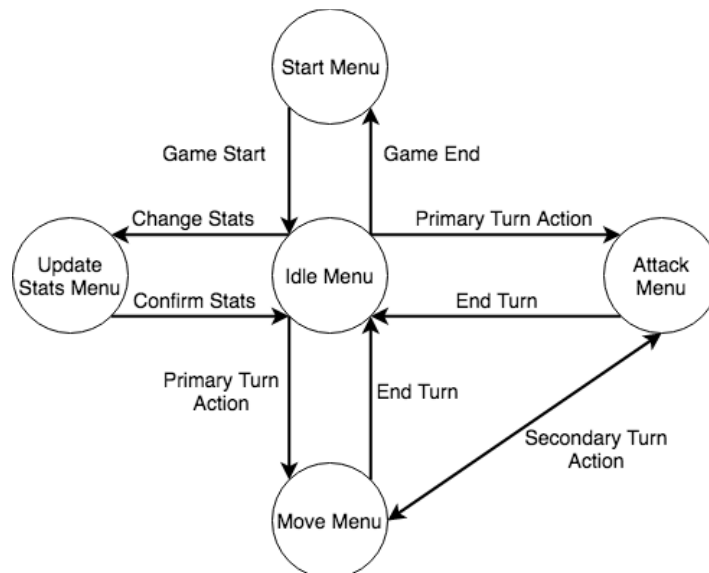


Figure 29: Mobile App State Diagram

When deciding which application environment to program our mobile app with we had two realistic options to choose from, Apple iOS or Android OS. There are less popular mobile

operating systems, but most of them are based on Android’s Open Source Project (AOSP) or, like Chrome OS, limit the application development to work with a browser only. As we are trying to limit our use of Wi-Fi and instead use Wi-Fi Direct for our interactions with the mobile application, a non-browser mobile OS is preferred. The main contributors to our decision included accessibility, financial, and developmental tools.

Since we are developing a smart table where the use of a smart phone with our application will be required, we needed to bring the app to the greatest percentage of people by choosing either iOS or Android OS. We can extrapolate this data by checking the amount of sales of each OS device per yearly quarter to see which OS is being used the most and check trends in purchases for the future. In Figure 30: Smartphone Sales, 2007 - 2018, we see that Android is clearly leading in number of sales for their OS in the recent decade. This means that there are many more users of Android OS than Apple iOS. We also can see a steady increase in the popularity of Android OS whereas iOS is barely changing in terms of sales per year, so we can assume that there will be even more Android users in the future. This seems to indicate that if we choose Android OS over iOS, more people will likely have access to our overall game through the app.

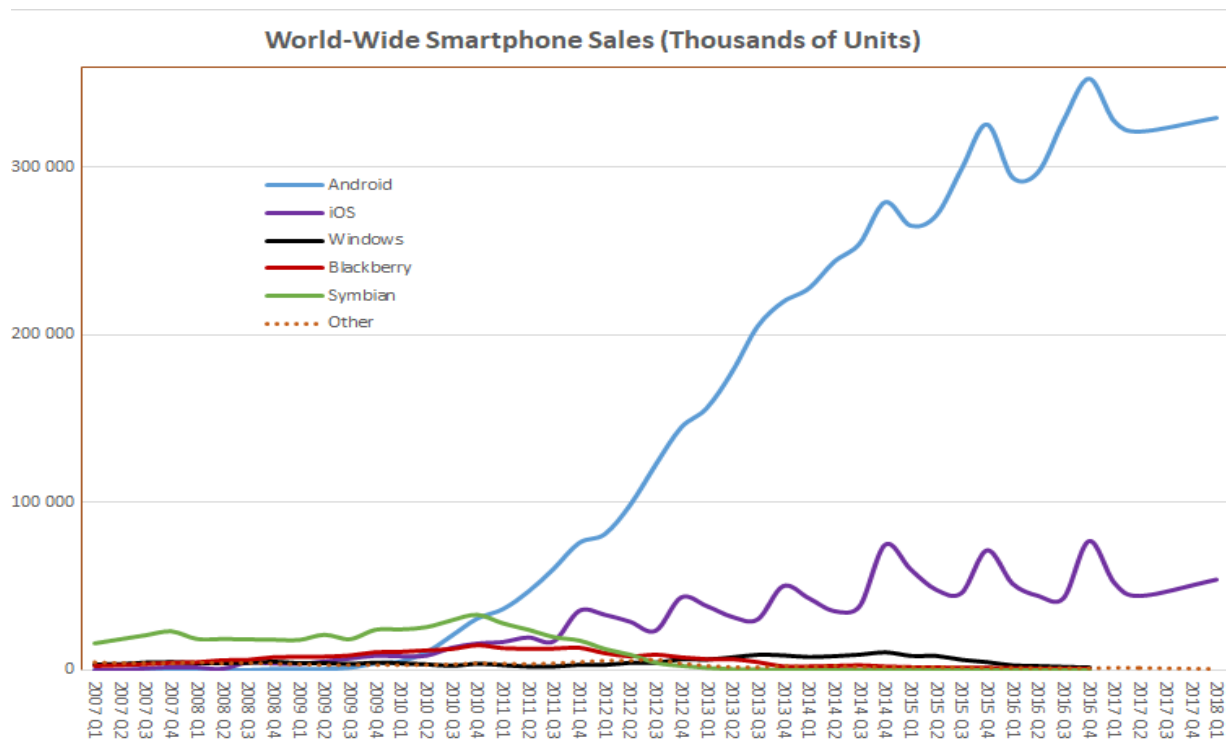


Figure 30: Smartphone Sales, 2007 - 2018

Reproduced in accordance with the Creative Commons Attribution-Share Alike 3.0 Unported license

6.3.1 Android OS Overview

Even though Android is clearly leading in terms of rising sales, there are some problems with just assuming there are more Android OS users. A major problem that Android developers face is software updates and obsolete units. Since Android OS is open-source, phone manufacturers have to choose whether or not to integrate a new software update with their older models of phones.

Since Apple manufactures their own phones they have a greater incentive than Android manufacturers to integrate older models with their newest update since the sale of those older models is still a large enough profit for them. On the other hand, Android phone manufacturers have less revenue coming in from older models since they have more competition each update cycle. This leaves older generations of Android phones obsolete much faster than those of Apple, with many still on older versions of the OS.

In Figure 31, we can see that only 10% of Android OS users are on the most recent big update that includes more features and changes than previous updates. This creates a problem when trying to develop an app for the platform. While we can program our app to work with the most recent update, it is not guaranteed to work with previous versions of the OS, so we would be losing a large number of potential people who can play our game. The solution to reaching a bigger audience when developing for Android is to program the app on a lower version of the OS since future updates are backwards compatible with apps in previous updates. So if we were to program our app for Oreo 8.0 we would have a potential 40% of Android users able to play our game. Just for one quarter in Figure 30, 40% of Android sales which totaled around 350 million are 140 million users a quarter in the more recent years.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%

Figure 31: Android OS Version Distribution

Reproduced from work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License

6.3.2 Apple iOS Overview

Apple iOS does have some distribution of OS versions since they usually have a major update every year alongside the release of their new phone. Although app developers still deal with some problems with updates and incompatibility, the problem is much less wide spread than Android OS. This happens because Apple is closed-source, making all production and applications of

devices with the OS happen in house. This means that Apple does not have to worry about manufacturers changing their phones to fit different AOSP derived OSs'. With this in-house production comes an easier time when updating older models of phones to the newer update. This makes it so that phones that released around the same window of now obsolete Android phones are still being updated to the new OS. The Apple Developer site [22] provides the distribution of OSs' for iPhones that released in the past four years. 55% of users are running iOS 13, 38% are running iOS 12, and the remaining 7% are running earlier versions. We see that iOS users are primarily using the two most recent updates compared to the many versions in use by Android users.

From Figure 30, we see that the total sales for Apple iPhones is around 60 million per quarter, so around 56 million (93%) of users are on iOS 12 or higher. Unlike Android OS, apps developed to work for previous generations may not be automatically compatible for future generations as Apple introduces new features and requirements with each update that an app must follow. App developers must decide to keep supporting each iOS when a major update occurs. Even though the sales are lower for iPhones, the adoption rate of previous devices for new updates is much higher than Android. As long as an iOS app works with the two most recent updates, then you are likely to reach most iPhone users.

6.3.3 Financial Costs

Since Android OS is open source, the developmental costs are much cheaper than the closed source iOS. Android Studio is the free IDE required to develop an Android OS app. It is available for free for Windows and is based on Java for coding. To publish your mobile app on the Google Play Store, there is a one-time developer fee of \$25. For iOS, you must own a MAC in order to develop an app for iOS. On the MAC OS, Xcode is the free IDE used with the Swift programming language to code iOS apps. App developers for iOS are charged an annual developer fee of \$99 to submit their app to the App store. Although we would only get a one-year fee requirement for our project, if we ever wanted to update our app in the future, we would need to renew with another developer fee.

6.3.4 Developmental Tool Differences

Android and iOS have different approaches when it comes to tools when developing apps. Since the computer's OS is also different there are many differences in UI and file management for mobile app development.

6.3.4.1 Android Studio

When developing for an Android OS app, programmers must use Java to handle creating user interfaces (UI) and interactions between the App and external communications like Wi-Fi or Bluetooth. Android Studio has a set of software tools called Android SDK which includes their required libraries, a debugger, an emulator, and some tutorials to help a new developer. The Android SDK also includes a feature that allows you to choose which version update of Android to support. The debugger has a log for error messages, a pane for seeing variables and threads, and testing frameworks. The emulator allows you to test your program in a virtual environment before pushing onto the Google Play store. Unfortunately, the emulator cannot simulate external communications like Wi-Fi or Bluetooth so a developer must have a physical phone to test those features. The Android emulator can also specify different versions of the Android OS to test for backwards compatibility of an app. Each Android virtual device has its own storage and cache for each instance of the virtual machine. The basic tutorial can help a new developer create an app

with features such as adapting screen size for different phones, UI, and performing background tasks not dependent on the UI.

6.3.4.2 Xcode

Developing for iOS uses Xcode for an IDE in comparison to Android Studio. Apple apps are programmed using Swift which Apple incorporated back in 2014 to help users create their own iPhone apps. Swift is a good language for beginners as it has many safeguards in place to minimize errors and allow fast execution of code through optimization. Xcode itself has many easy to use design tools to help ease in a new mobile app developer. The declarative syntax helps distinguish what each element of the UI should do. For designing new UI elements, Xcode has a drag and drop feature that allows for easy creation of visual UI designs. Xcode has a simulator that can prototype the entire created UI and app testing that will test background app functionality. Like Android Studio, Xcode cannot simulate WI-FI or Bluetooth functionality to external sources because those functionalities need specific hardware to run.

6.3.4.3 Feature Comparison

Both of these IDEs have similar features such as emulation or simulation, but they also have a few differences that make them distinct. Android Studio offers a Gradle-based build system which allows custom builds through the use of plugins created or downloaded by the user. Android Studio also allows multiple APK generation and variants. Android OS has more community resources because it is open source allowing more people to share their builds and plugins for Android Studio. Unique to Xcode is an assistant editor and asset catalog that allows for easy creation of UI elements. UI is one of the most difficult part of any app creation to make interact correctly with the features and look comprehensive enough for new users. The catalog greatly helps cut down on the time required to build a UI and instead allows developers to focus on features and interactions between the app and input from external communication like Bluetooth.

Based on user experience, Xcode does more than Android Studio in terms of a friendly development environment and easy to use assets. Although in terms of each programming language, Android Studio's Java based coding is more familiar to programmers trying to make their first mobile app. Although the programming language might feel more familiar in Android Studio, Xcode more than makes up for it using its storyboard seamless IDE integration with the simulation. Although because of this integration, Xcode can be quite slower in compiling since it is connected to the simulator which slows down while trying to compile simple changes in code. Swift is a very strict language that updates to change library functions which can make a previously working apps crash or not compile at all in different versions. The overall financial requirement is much greater for iOS than Android OS, especially if no user had a MAC to begin with.

6.3.5 OS Selection

With both OS options researched we compile Table 35 and compare the benefits and disadvantages of each. With this comparison we can choose which OS to use in our project.

Table 35: iOS vs Android

	iOS	Android OS
Benefits	+ Simpler and streamlined Xcode helps new developers, especially when creating visual UI	+ Simple one time developer fee
	+ Brand loyalty of customers and distribution of customers is for the most recent two big updates	+ Updates are backwards compatible with apps made in previous versions
	+ Simulation of code is possible using Xcode programmed with Swift	+ Emulation of different versions is possible with Android Studio
		+ Increasing sales for OS means more potential future users.
		+ In our group, we all have Windows, so we have free access to Android Studio
Disadvantages	- No one in our group has a MAC so we would need to borrow one since purchasing one is too expensive	- Fragmentation of users mean fewer people on each update
	- An annual developer fee of \$99	- More difficult to make UI in Android Studio
	- Smaller percentage of people have iOS based on number of sales per quarter	

The benefits and disadvantages are not equally important, and we decided going with Android OS for our mobile app. The lack of a MAC and the high annual developer fee for Apple iOS is what ultimately made the decision. While we can most likely borrow a MAC from someone we know or even UCF, there might be a reason that we cannot one day and all progress or testing on the app portion of our project would halt.

6.3.6 App User Interface Framework

Because the app will be built in C++, a third-party open source library will be required for the interface. Although the project will only develop an android app, some third-party libraries do enable the ability to run the same code on different platforms. This allows the same look to be ported to different platforms such as an iOS environment or even a desktop environment. This ability to have the same look across different platforms comes from the frameworks process of drawing their own components based on the operating system the code is running on. This also helps the performance and ability to look like other native applications. Two open source frameworks that will be considered are QT and wxWidgets.

6.3.6.1 QT

QT [23] is a cross platform, open source user interface framework. QT is developed in C++ and performance is just under native libraries in terms of speed of creating components. There are two license options that QT allows developers to use, a commercial license and free license under the L-GPL licenses. To use QT for free, source code must be visible to the users or the QT libraries need to be dynamically linked to the source code. In other words, if the library functions need to be modified to better work with the project, the project must be made open sourced. This will allow other developers to see modifications made and help the framework overall. If the libraries do not need to be modified, then the libraries can be dynamically linked to the project which shows that no modifications were made to the framework. The goal of QT is to provide a framework that can

easily be ported across platforms. Therefore, they provide plenty of documentation on how to use the different library classes and their methods, how to start a project, and how to compile and build the projects. In regard to the android app, QT provides documentation that shows how to convert a C++ project to a build compatible with android development.

6.3.6.2 wxWidgets

The user interface framework wxWidgets [24] is also cross platform and open sourced. It is developed and compiled through the C++ language making the speed of the code comparable with built in user interface libraries. wxWidgets is completely free for both free and commercial products under the L-GPL license with an exception that code does not need to be distributed to users. In other words, the framework libraries can be modified without the need to show other developers what changes have been made. The framework has the benefit of age being created in 1992. There are many popular programs that use this framework. It is important to note that these programs are all for desktop platforms. Android development is not fully supported, and documentation is not at a level that would create ease of use while developing.

6.3.6.3 Android Studio Layout Editor

The Android Studio layout editor [25] is a drag and drop user interface that aims to provide easy to create and implement interfaces. As seen in Figure 32, programmers drag components from the component selection tool and place them on the window in the graphical component view. Once placed, the component view will display the component without the added graphic and an instance of that component will be added to the list in the component tree. The editor provides built-in basic components for development and the ability to create subclasses from those components. This creates the ability to modify components to better fit the needs of the application or to create entirely new components.

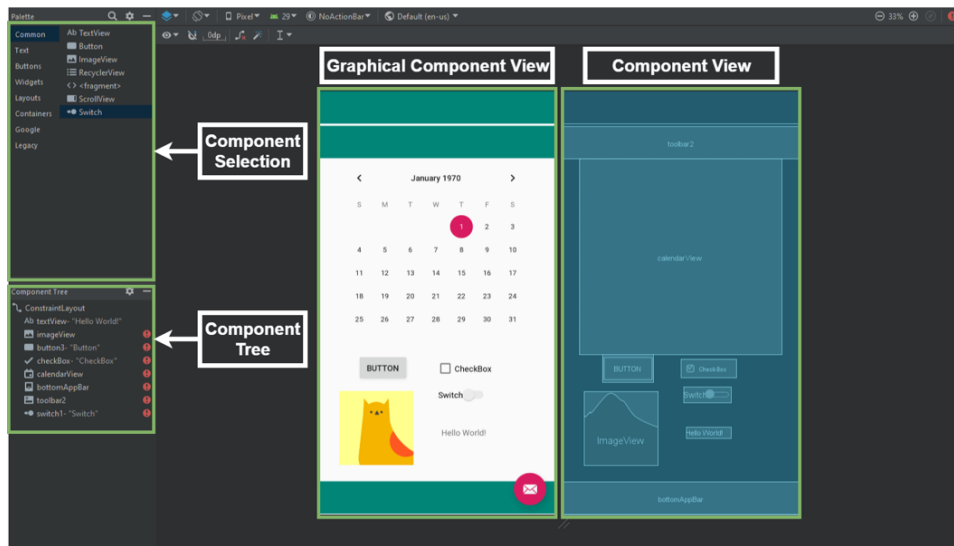


Figure 32: Android Studio Layout Editor

Reproduced from work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License

6.3.6.4 User Interface Framework Selection

Although this project is developing an android app, iOS development is still possible for the future and should be included in the decision. Therefore, Android Studio layout editor is ruled out

because it constrains the user interface to only apply to android devices. However, if time constraints do play a factor in interface development, Android Studio layout editor will be an easy fall back to create a fast easy to use interface. If the case does occur where Android Studio is used to create the interface and the project environment is moving or including iOS, then to develop a similar look for iOS will be difficult and time consuming. wxWidgets would be useful for developing interfaces on a Windows, Mac, or Linux platform; however, android documentation is lacking and will cause unnecessary issues when building the app. Therefore, wxWidgets will be excluded from the decision. QT remains and will be the interface framework used as it is free to use and provides cross platform support.

6.3.7 App User Interface Design

The app is designed to provide easy, logical access to character information for both players and game masters, as well as additional controls for game setup and turn assignment for game masters. The logic and functions available to users through the app are detailed in Figure 33. Some functions should also be available offline (i.e. when not connected to the table) to allow users to prepare for games at any time and from any location. Each function may expand a sub-menu or series of sub-menus as required by the complexity of the information being accessed.

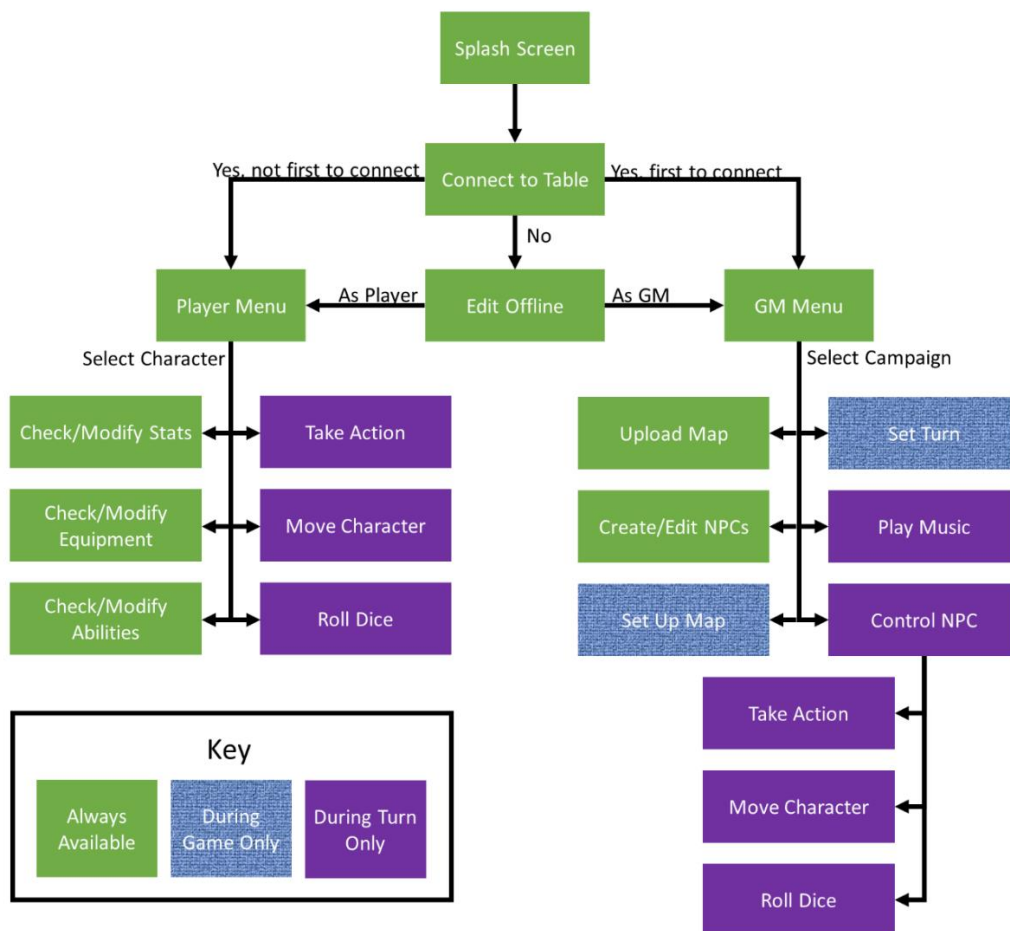


Figure 33: App Interface Logic and Functions

The app is styled to be reminiscent of the paper and pencil format of the original game, and at least two fonts will be available for the menus. The default font is selected to match the fantasy theme of the game and app, while at least one secondary font is provided in accordance with the Web Content Accessibility Guidelines (WCAG) for accessibility. Some preliminary splash screens and interface designs for the player character menu are shown in Figure 34, for a 1080 x 1920 pixel display (a standard resolution for modern mobile devices). The brown design on the left uses a traditional paper aesthetic with individual options for many aspects of character modification and is easily adapted to a variety of display dimensions. The blue design on the right shows how the app may incorporate a specific color associated with a character as a secondary identification into the menu system and provide a limited set of options during a player's turn for more efficient gameplay. However, this second design is less detailed and may be more difficult to modify to different dimensions.



Figure 34: Preliminary User Interface Designs

6.4 Game Software

6.4.1 Game Software Endpoints

Figure 35 shows the use case for the game software subsystem. The diagram lists major functionality that the game subsystem will handle. The actors on the left side of the system represent the inputs to the system while the actor on the right side represent the output. Each actor will be an endpoint managed by the software. The input endpoints are the mobile app and object detector class. The output endpoint is the display. Since the game is heavily dependent on user input, the game will handle the mobile app endpoint by being in constant contact with each user's app. Wi-Fi direct protocols will be in place to distinguish user devices and create communication channels. The users will be communicating turn information which will then be processed to properly display images. The GM will be initiating protocols for creating, saving, and loading games. All players will need to be able to connect to the network. The game software will be the location that maintains memory of player piece movements. Updated player piece locations will be detected through the object detection software and sent to the object detector class to be stored through a socket. Some functionality will manipulate the displayed images to create a more

coherent game experience. That functionality will be methods such as update map, display map, and display attack or range spaces.

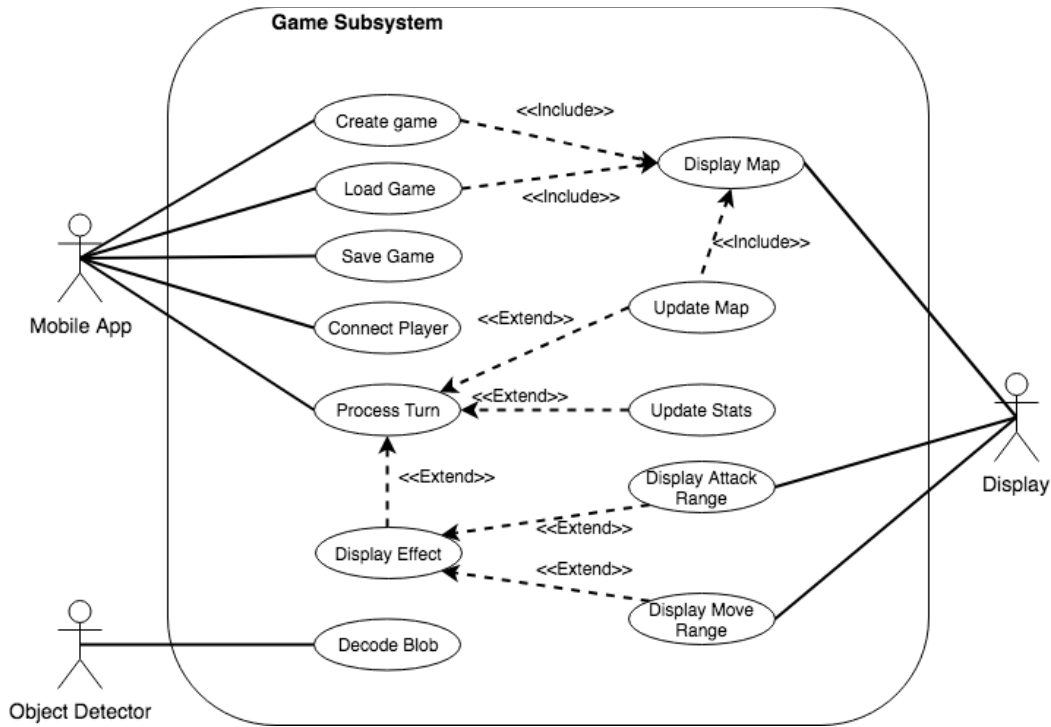


Figure 35: Game Use Case Diagram

6.4.2 Game Software Class Descriptions

The game software can be broken up into different classes that will encapsulate the different aspects of the program. By grouping the major functionality seen in Figure 35, four classes will be used to incorporate all functionality. Those classes are the player class, game class, Wi-Fi direct class, and display class. The basic class structure can be seen in Figure 36. The player class will be used to keep a local record of the character traits and provide methods for user actions. The game class will handle the logistics of the game logic. More specifically, the game class will maintain the instances of each of the other classes and control logic flow to each of the classes. The Wi-Fi direct class will be an object that creates a group network that will be broadcasted to the nearby android devices. The display class will handle all display related methods and attributes. An important aspect to the game software is that there will be no user interface. The users will not need to interact directly with the game software through menu options on the display. The projector will only display an image of a game map that can be interacted with depending on actions taken during player turns. The app will provide menus if there needs to be any for display configurations.

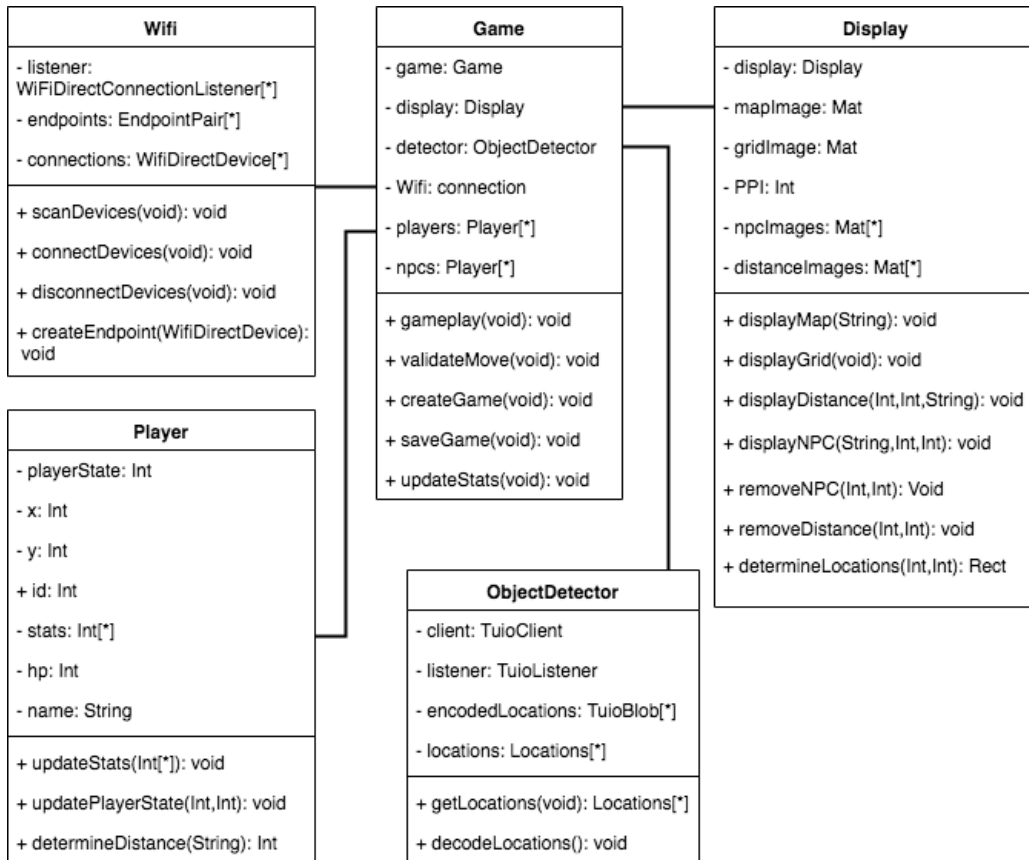


Figure 36: Game Software Class Diagram

6.4.2.1 Player Class

The player class holds the information about the user’s character and methods to manipulate that information. The methods required for the player class can be seen in Table 36. There are several public getter and setter methods that have been omitted from the table because they do not provide the major functionality of the class. The table instead will hold the methods that are important for the Game class. These methods will be updatePlayerStats which will take in a list of stats and update the player’s local field. Similarly, the updatePlayerState will take in information that can deduce the state and update that field. DetermineDistance will calculate a value based on a given string. This value will be used to display images on the display.

Table 36: Player Class Methods

Method	Method Purpose
Player(): Player	Player constructor that will instantiate a player object and return a reference to that object.
updatePlayerStats(std::vector<int> newStats): void	Method that takes in a vector of values and updates the corresponding field elements.
updatePlayerState(int hitPoints, int dice): void	Method that calculates player state based on a combination of a dice roll and hit points.
determineDistance(String action): int	Produces an integer value for the range of board spaces for a given action.

The class fields will act as a replacement for the traditionally character sheet. The character stats, the player location, and the player ID will be maintained. There are many values on the character sheet that will not be displayed in Table 37. Fields related to other class methods and their descriptions will be displayed below.

Table 37: Player Class Fields

Attribute	Descriptions
private int playerState	An integer that holds the state of the player. 0 maps to healthy 1 maps to knocked out 2 maps to dead
private int x	The x coordinate of the player location.
private int y	The y coordinate of the player location.
private int id	A unique ID for each player
private std::vector<int> stats	A vector of integers where each index refers to a player statistic.
private int hp	The current total of health points.
private String name	The character name

The stats vector will be where all the character attributes are stored. Examples of these player attributes are basic abilities such as strength, dexterity, constitution, intelligence, wisdom, and charisma. The id field will be used for distinguishing the different players. The player id will need to be readable from other classes to ensure that the correct player instance is being manipulate. The display class will be mapping blobs to players and this is achieved by mapping to the unique id value. The player id will be generated based on the mobile device. The Wi-Fi connection already requires a unique id for each device and this id can be repurposed for the player id as well.

Encapsulation is utilized in this class and public getter and setter functions will be needed for each of the field elements. Methods in the Game and Display classes rely on field data from this class. However, they should not be granted direct access to the fields to avoid any corruption in the data. The Game class will require the ability to read the player's id to select the correct player object to manipulate. The player stats will also be used to determine gameplay events such as attacks. Some player stats will need to be updated throughout the game. The only class that will be granted permission to modify the fields will be the Player class. Each field will therefore require methods to set those values. The player will also have different states that will be dependent on the players health points and a physical dice roll. There will need to be a menu to insert the dice roll on the mobile app so that the Player class can properly handle calculations need to update the player state.

The Player class will be in charge of managing player actions. These player actions come with values associated for either hit points or range of action. When performing an attack action, the attack range will be based on player stats and so will the hit points applied during that attack. Similar when performing a move action, the player will have a set range based on their player stats to move. Players have the options to perform both actions on any given turn assuming NPCs are present on the board. If the game state is not in combat, then the attack action will not be an option. Players also have the choice of just choosing a single action. Since there are a few options, the player will need to select what action they would like to perform through the mobile app. The

Game class will then need to ask for the distance in order to display the range and continue gameplay.

NPCs will be controlled by the game leader through the mobile app. For all purpose they will act as a normal Player. A subclass will not be required as no different functionality is required. A problem that will be faced will be in the unique id. The same procedure as was discussed above cannot be done for NPCs. Since each NPC technically is owned by the game leader, they would all have the same unique id which would void the uniqueness of the id. An obvious secondary id indicator would be the character name. Although unlikely, there should not be a rule to prohibit the use of the same name by two or more characters.

6.4.2.2 Game Class

The game class will be the starting point for the program. When the program is launched, an instance of the game class will be instantiated, and the game setup will begin. The key steps to setup will be to connect users to the game, display the map, and either restore player locations or begin tracking player locations. The constructor for this class will be used for setup. The class will also be in charge of validating player actions on the display and processing any input from a player's mobile app. The major methods for game functionality are shown in Table 38.

Table 38: Game Class Methods

Method	Method Purpose
Game(): Game	Game class constructor that instantiates a game object, runs the setup process, and returns a reference of that object.
gameplay(): void	Processes the gameplay requests.
validateMovement(Player player, int action): Bool	Method to validate movement location. Uses a player object and the type of action.
createGame(): void	Method to begin a game's startup
saveGame(): void	Saves all states of each class into a file and stores the save file on the game leaders app.
loadGame(File *gameData): void	Takes a save file and generates all objects stored in the file.
updateStats(int id, std::vector<int> newStats): void	Method that will find the correct player object using the given id and update that player's stats.

Since the game class will be maintaining the major functionality of the game, there will be private field elements for each of the object required for player and display interactions. The players and npcs will also be maintained in this class. A vector for each will be the data structure to hold the values. Both will be maintained as instances of the Player class. The fields can be seen in Table 39.

Table 39: Game Class Fields

Attribute	Descriptions
private Game instance	The object itself.
private Display display	The object that will maintain all display objects.
private Wifi connection	The object that will send and receive input over a wifi connection.
private ObjectDetector detector	The object used to detect objects on the display
private int gameState	Holds the state of the game. 0 mapping to normal play, 1 mapping to combat.
private std::vector<Player> players	A vector of user player objects
private std::vector<Player> npc	A vector of NPC players objects.

6.4.2.2.1 Game Constructor

The game constructor should only be called by the main function when the system is powered on. When the game constructor is called, the first task will be to allocate memory for the object if it does not exist. There should only ever be one instance of the game class running at any given time. Before the memory is allocated, the constructor will check for any instance of the object already existing. If the object does not exist, it will create the object and store it in the private field. The constructor will then continue to setup only if the instance was just generated. Setup will create an instance of the Wifi, Display, Player, and ObjectDetector class. Once all of the necessary fields are stored, the gameplay method will be called so that the game can begin. The gameplay method call indicates the end of the constructor and will finalize the connection of all players for the game. Figure 37 illustrates the flow of the constructor beginning with the power on event and the call from main.

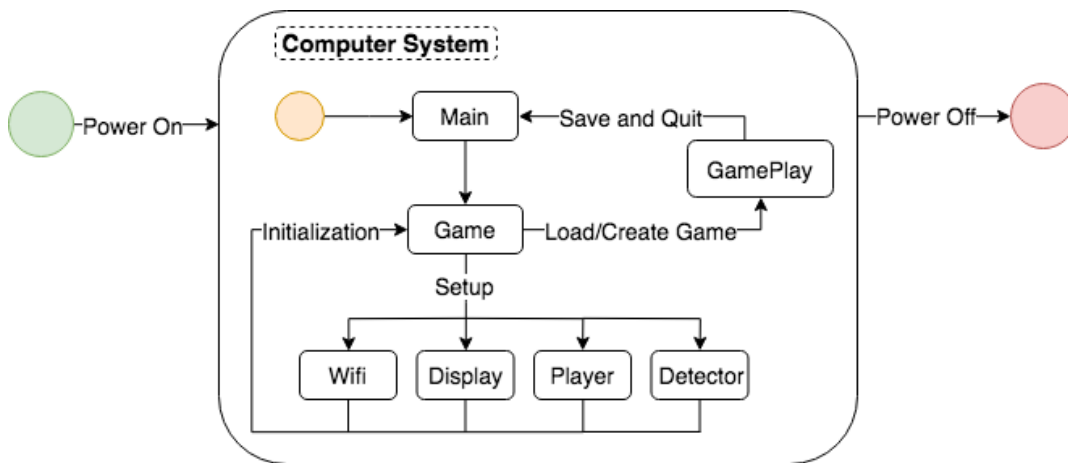


Figure 37: Game Constructor Flowchart

6.4.2.2.2 Gameplay

The gameplay method will be the main functionality of the game. It will maintain ordered player turns and process all user input. At this point, the game will have an instance and all players will be connected. The turn order will be decided by the game master and maintained in this class. In a round, each player will cycle and take a turn. The parts of the turn that will have a digital action are movement and attacks. Both of which will call a Display class method to highlight spaces. The game will then wait for the user to choose a physical space or move their player piece on the

display. The location of the action will be collected by the object detector class. The location will then pass through the validate movement method. If the response of the method is true, then the action was valid, and the turn will continue. If the method returns false, the position is invalid, and the game will request the player to correct the action. Depending on the game state, the player could have multiple actions to take during one turn. If the state is combat, then the attack action can be done by itself, before, or after the movement action. If this is the case, the player will be asked again what action they are going to take, and the process will repeat.

6.4.2.2.3 Validate Movement Method

Since movement and attack locations are user inputted, there must be a verification method for valid input. This will be accomplished by taking the player’s original location, the distance the player can travel or attack, and the new location found from the object detection software. Using the initial position of the player and the distance that they can move or attack, a boundary can be formed. Verifying that the new position is within the boundary can easily be done. Verifying location must go beyond simply checking the location to the boundary formed. The display shows a map that the player pieces or placed on with a grid overlay that will be divided into 1” by 1” squares. Ideally, players will place their pieces perfectly within each of those squares. Realistically, players will not be perfect, and placement verification should not harm the flow of the game if it does not need to. If a piece is slightly overlapping another square, the game should not ask the user to fix the position. However, if the piece is greatly overlapping another square to the point that it cannot distinguish which square was intended, the game must request clarification from the user.

As a way to mitigate the issue of greatly overlapping squares, an algorithm will be in place to decide on the grid space that the piece occupies the most space in. The only assumption this makes is that the user will attempt to place the piece mostly in their desired location. This can only be manipulated if the user intentionally places the piece mostly in the wrong grid space. At that point, the game will just assume that they are in the other grid space and continue operating as expected.

A case to observe is when two player pieces are overlapping the same grid space. This can be seen in the valid placement case of Figure 38. The circles represent player pieces and the shaded region represents the grid space with the most piece area. The circles are shown in green to signify that they are valid placements. Although the two pieces occupy the same space, one piece will have more area than the other and that piece will be the true owner of that space. Specifically, grid space C2 in the figure has both player pieces within the space. As shown by the shading, P2 has a majority of area within C2 making it the owner of the space. It can be generalized that if a player piece has a majority of its area within a grid space it will be the owner of that grid space.

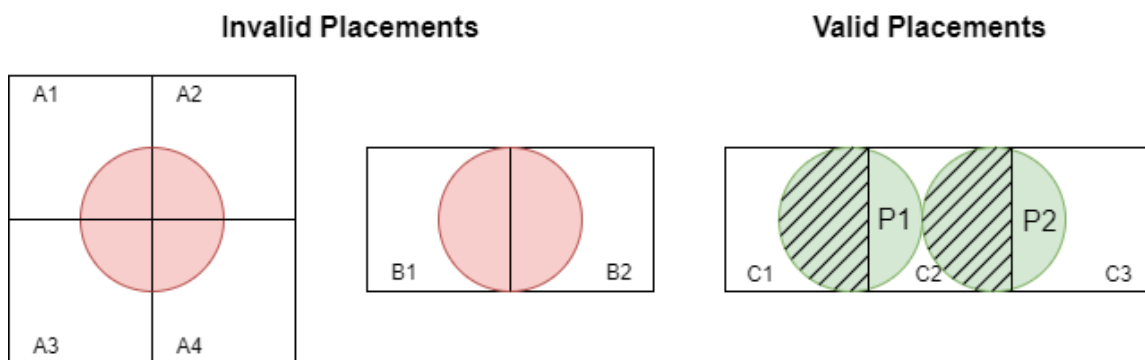


Figure 38: Player Piece Placements

Cases of invalid player piece placement are shown in the invalid placements section of Figure 38. The player pieces are the circles in the figure and are shown in red to signify invalid placement. The placements create a problem programmatically as the area distribution is near equally distributed across multiple grid spaces. In the first instance, the player piece occupies four grid spaces equally. Similarly, in the second instance, the player piece occupies two spaces equally. It is not possible to determine which square was intended by the player using only the piece area because the distribution is even. There also cannot be a deterministic solution to this problem such as always pick the top left most space that the player piece occupies. This assumes more than the software should and will break the flow of the game when the deterministic nature of the solution is wrong. Therefore, in order to maintain proper game flow, all spaces that the piece is overlapping in should be highlighted and the player should be notified that they need to fix the position of their player piece. This avoids any desync between the players and the software where the game software believes the player piece is in a specific location and the player believes that the piece is in a different location.

The case where two player pieces occupy exactly half of a single space can never occur because of the algorithm already discussed. Two player turns will be done sequentially and not simultaneously. When the first player has the option to move their piece, the piece placement will be verified. If the piece is occupying equal area in two adjacent squares, the game will prompt the user to fix their placement. They will then fix the placement of the piece and it will occupy less than 50% of the area in the square that will be shared. Since that square has space for more than 50% of the second player's piece, they can place that piece in the shared square without with a majority of area.

6.4.2.2.4 Create, Save, and Load Game

The game class will also handle the save and load methods. Both methods will be initiated by a request from the game master's mobile app. The Game class will then handle the request. Figure 39 shows the basic process of the save game method. The game will first check to see if the mobile device has a past save data file. If the file exists, it will be overwritten, otherwise it will be created. The different object fields will be iterated through and their contents will be recorded into the save file. This save file will then be transmitted to the game master's device for storage. Each of the objects will be deleted to free memory. Finally, all devices will be disconnected to setup for the next game or shut down.

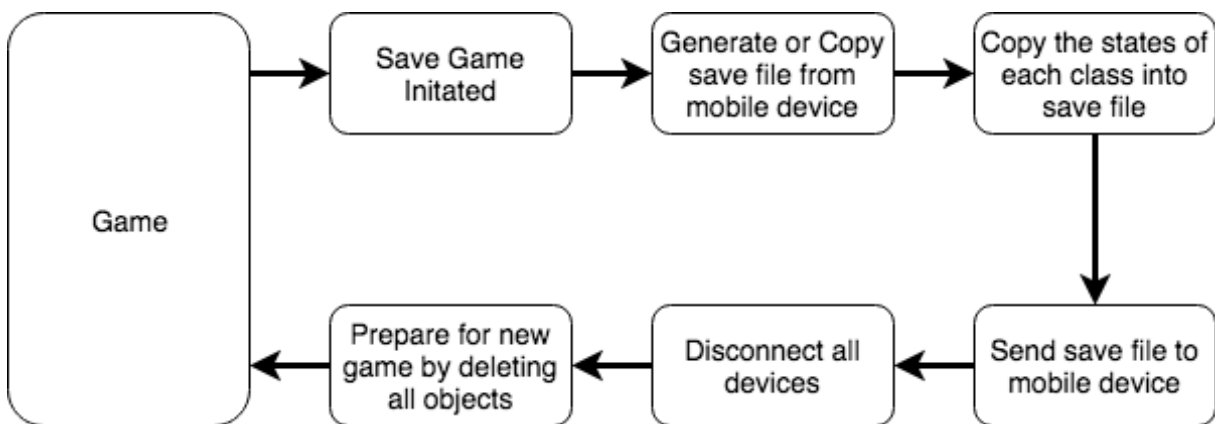


Figure 39: Save Game Flowchart

When loading a game, the save file will be transmitted from the game leader's device and will be processed by the load game method. The file will be parsed and each of the object fields will be populated with the save data. After the objects are populated, all player devices will be connected to the server and the display will begin its setup. After the display setup, the game is ready to resume.

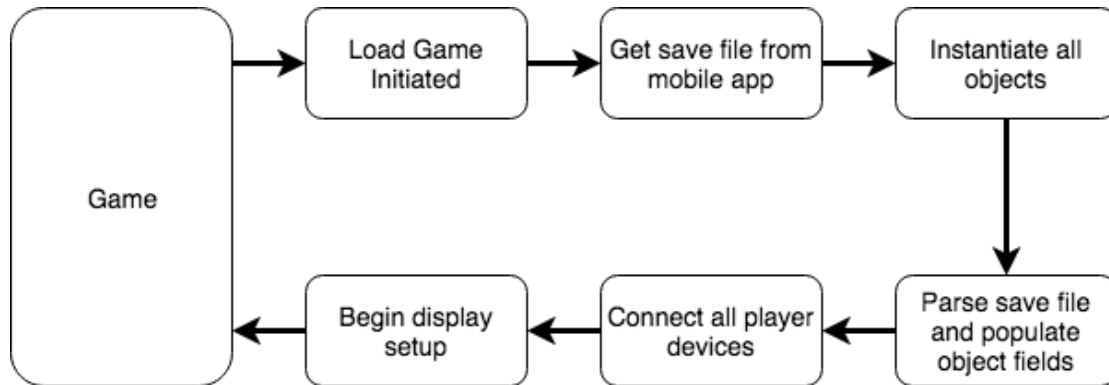


Figure 40: Load Game Flowchart

Create game will act very similar to load game. Besides the obvious difference of no save file, create game will initiate the start of the game. The start of the game will begin by receiving the number of players playing and the map to be displayed from the game leader. The next step will be to setup the character sheet with all character statistics and place pieces in a starting location. While all these steps are occurring, the Game call will be receiving and populating class elements with information about each player to begin tracking gameplay.

6.4.2.3 Display Class

The display class holds the methods required to do all image processing required for gameplay. A third-party image processing library will be used to do the image overlaying portion of the image processing. The third-party library that will be used will be OpenCV [26]. OpenCV is an open source computer vision software library. CCV also uses OpenCV in their object detection protocols. The library has support for all major platforms; Windows, Linux, and Mac OS. The framework has been released in C++, Java, and Python. The free use of OpenCV is allowed under the condition that the copyright notices listed at the top of each file is not to be removed from the files. Also, the library is allowed to be modified as well. Any additions to the software can be made to better fit the cases required in this project's software. Some concerns that have been seen when researching other projects using OpenCV is that C++ has some issues importing and using library methods. As a result, the Display class may need to be developed in a language such as Python. There are ways to port the class over to C++ or a set of API scripts can be written and called from the Display class written in C++.

OpenCV will be used to create the displayed image through their image processing methods. Specifically, the methods to read in, write out, show, and add images together. These methods can be seen in Table 40. These four methods will be able to provide the necessary manipulations required to update and display images for the users. The addWeighted() method will be used to merge two images with a different opacity. The base image will have the higher opacity while the image being overlaid on top will have a lower opacity creating a merging effect.

Table 40: OpenCV Methods

Method	Method Purpose
imshow(String name, InputArray mat): void	Creates a window titled the given name of the image array mat.
imread(String filename, int flags): Mat	Reads in an image into a Mat type using the filename path.
imwrite(String filename, InputArray image, std::vector<int> params): bool	Creates an image file of the input array at the given filename path. Params can be a list of parameters to save the file with. On success a True Boolean is returned.
addWeighted(InputArray src1, double alpha, InputArray src2, double beta, double gamma, InputArray dest, int dtype): void	Merges src1 and src2 into a dest array based on the parameters of alpha, beta, and gamma.

The OpenCV methods will be incorporated in the Display class methods to produce the desired functionality. All methods shown in Table 41 will be implemented. They represent the major functionality of the Display class and give descriptions of the goals of the Display class. Methods requiring a location will require processing from the object detection software to provide accurate pixel location.

Table 41: Display Class Methods

Method	Method Purpose
Display(): Display	Constructor to instantiate display object and return a reference of that object.
displayMap(String URL): void	Creates a map image object from the passed in URL and displays the image to the display.
displayGridLines(): void	Overlays the gridline image object on top of the map image object.
displayDistance(int x, int y, int distance, String Color): void	Creates a set of images of the color passed in. Overlays those images centered around the x and y given to the distance given.
displayNPC(String URL, int x, int y): void	Displays a passed NPC image from the URL at the passed x and y location.
removeNPC(int x, int y): void	Removes an NPC image at an x and y location
removeDistance(int x, int y): void	Removes the distance images surrounding the x and y location.
determineLocations(int x, int y): Rect	Takes in an x and y coordinate and translates it to a Rect object for location of pixels.

Table 42 shows all privately maintained fields for the class. These fields will enable the class to manage the different layers of images and the ability to determine their locations. The map and grid line images will be maintained to avoid corrupting the original files. The PPI will be maintained so that when dynamically creating images, the scale is correct. A list of npc and distance images will be maintained. This will be referenced when creating images.

Table 42: Display Class Fields

Attribute	Descriptions
private Display instance	The instance of the display object .
private Mat mapImage	The map image object.
private Mat gridImage	The gridline image object.
private int PPI	The PPI for the current display
private std::vector<Mat> npcImages	A vector of NPC image objects
private std::vector<Mat> distanceImages	A vector of Distance image objects

6.4.2.3.1 Display Constructor

The display constructor will create memory for the display object. There should only be one instance of this object to play the game. Therefore, when the constructor is called, it will check the private field instance to determine if a display instance already exists. Otherwise it will create an instance, save that instance in the private field and return a reference of that instance for the calling class.

6.4.2.3.2 Methods to Display Images

The map image that will be projected on to the touch display will be provided by the user through the mobile app. To not destroy the original map image, secondary images will be overlaid on top. Therefore, the map image will be the base background image. An important layer that needs to be added to make the map playable is the 1" by 1" grid of spaces. Each space will represent a valid position that a player can place their figure. The 1" by 1" grid must be to scale because the physical game pieces need to fit within them properly.

An image can simply be thought as an area of pixels. A specification required to translate an image's pixels into physical distances is pixels per inch (PPI). This value depends on the native resolution of the projector as well as the length and width of the display region. To find the PPI, let the native resolution of the projector be represented by R and the length, in inches, of the display side be represented by L . Equation 4 shown below will produce the PPI for a projector with the native resolution R projecting on to a square display with a side length of L . For this project, the native resolution of the projector is 720p and the side length of the display is 24". The PPI is 30 pixels per inch and each space will require 900 pixels. It is important to note that the PPI for a projector is not unique. This is because a projector will only output the number of pixels that is equal to its native resolution. The same number of pixels will always be displayed on to any display surface. Therefore, the display surface size is the varying factor for PPI. PPI is the same as DPI (dots per inch); however, the display will be dealing with pixels so thinking in terms of pixels is more useful.

Equation 4: PPI

$$PPI = R/L$$

The grid that is overlaid onto the map image will be a separate image created with all parts transparent except for the gridlines. To accomplish proper scaling, the PPI found above will be the exact dimension in pixels needed for the 1" by 1" grid spaces. The first and last two rows of pixels will be used to create the horizontal grid lines. Likewise, the first and last two columns of pixels will be used to create the vertical grid lines. This will produce a grid with line width equal to four pixels. To determine the width in inches, the inverse of the PPI is taken. Each line will have a width of four times the inverse PPI which will be 0.13".

The ability to distinguish grid spaces is a required property that is needed to properly track gameplay and overlay smaller images. A matrix numbering system is used to identify each space. Specifically, each space is identified by its row and column value with the top left space being position (0,0). The playable region will be 2' by 2' with 1" by 1" spaces creating a total of 576 spaces. The row and column index will vary from 0 to 23. Although the exact pixels are not used, it can be easily determined what pixels are occupying a given space. The exact pixels can be represented by four different values corresponding to each of the four corners of the space; top

left, tl, top right, tr, bottom left, bl, and bottom right, br. Given an arbitrary space (n, m), the four points can be found using Equation 5.

When overlaying smaller images on to the map, the region that the image needs to occupy must be a value passed to the overlaying function. The determine location method will take in a grid space location and apply the equations shown in Equation 5 to create a rectangle object that holds the pixel locations on the display.

Equation 5: Grid Space Coordinates

$$\begin{aligned}
 tl &= (PPI * n, PPI * m) & tr &= (PPI * n, PPI * (m + 1)) \\
 bl &= (PPI * (n + 1), PPI * m) & br &= (PPI * (n + 1), PPI * (m + 1))
 \end{aligned}$$

No padding will need to be done to the images to create buffer along the edge of the image. The table design takes care of the buffer needed for the display and the table. It creates a slight buffer for all edges of the display because the display surface will be resting on a lip of the table. This will relieve any issues that may have occurred had the display been flush with the table.

The actual overlaying will be done by generating a new image each time there is a change. This will be done using library methods that will copy an image on to another image. The general flow will be to create an object for the new image, store that object in local fields, and add the image to the display. This process begins with the function call from the Game class and the flow is depicted in Figure 43.

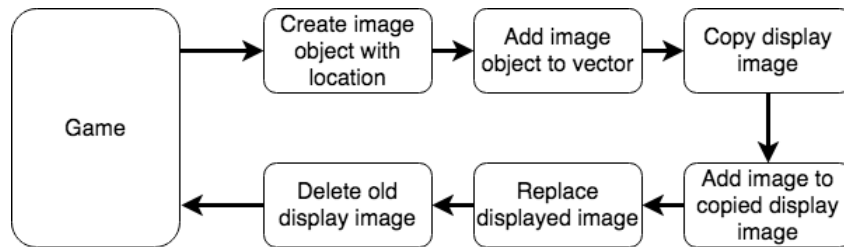


Figure 41: Image Addition Flow

This process will be in such a way that the layers are presented in the correct order. Following Table 43, the gridlines will be added on top of the map image first. Followed by any distance indicator spaces and lastly an NPC's that are currently on the board. This will avoid any NPC's from accidentally hiding under any other image. They are considered players and their locations should be known and visible at all times. As a way to create a more natural look, the distance indicators will not be solid spaces of a single color. Instead, the distance indicator image will be blended with the map to give the map a highlighted affect. This will allow the map image to avoid being completely blocked out by the highlighting, but still remain different enough to be noticeable. The last concern with the image formation is when adding the NPC image and the distance indicator that it blocks the grid lines. This issue will be resolved by only placing the images inside the area that is bounded by the gridlines. Therefore, the NPC and distance indicator images will not occupy the 900-pixel area that each space provides. Instead it will occupy a 676-pixel area because each dimension will be reduced by 4 pixels to bound the area by the grid lines.

Table 43: Display Image Level

Layer	Level
NPC Image	4 : Top
Distance Indicator Images	3
Grid Image	2
Map Image	1 : Base

Figure 42 shows an example of the overlaying levels. The slanted red line represents the map image. Every other layer of the image can clearly be seen on top of the red lines showing that they are the base image. The next layer is the grid lines which are shown by the black lines. The distance indicator shown by the green shaded region is entirely within the grid lines. There is no overlap of the regions. The distance indicator does not have full opacity to allow the behind images to be seen as well. Finally, the NPC image is shown as the blue circle which is on top of every layer.

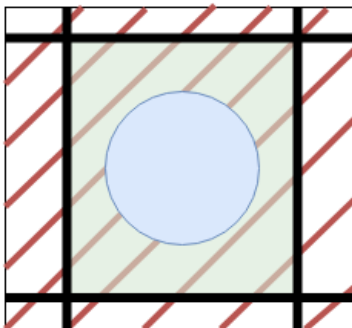


Figure 42: Overlaying Example

It will be important to fully generate the new image before removing the old image. This will avoid any gaps between images that will appear as a flicker. This process will not have smooth transitions and will instead be instantaneous changes from image to image. The majority of the image should remain the same making the actual transition not seem as abrupt. As a way to create some effects, when populating a list of images, periodically the image can be refreshed so that a large chunk of spaces is not modified in a single refresh. For example, the distance indicator spaces will be centered around the player's piece. If the spaces are populated in a circle motion being updated each time the radius expands one space, then there will be an effect of the distance spaces radiating from the player's piece. This can be ensured by creating a small delay between refreshes as well. To mimic that process when removing the distance indicators, it would require more computing as each iteration of a refresh would require the entire image to be rebuilt. Testing will be required to determine if the performance degradation due to repeatedly building the image is insignificantly enough to include the feature.

6.4.2.3.3 Methods to Remove Images

As a way to make removing images more straight forward, when an image is added to the display an object is created that holds all the details about the image and its location on the display. When removing images, information about the image will need to be passed in. Identifying information will be the image's type and its location on the display. From this information, the image object

will be found and freed and a new display image will be generated and updated. This general process can be seen in Figure 43.

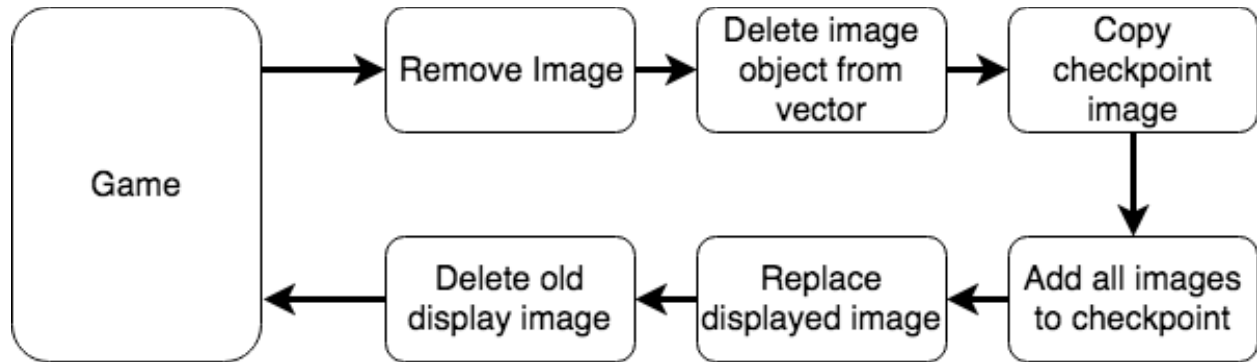


Figure 43: Image Removal Flow

A major concern with this process will be performance time for images that require many layers to be copied. As a way to increase the performance of this process, checkpoints will be made. Images that remain on the display for long periods of time will be saved. For example, the map with gridlines will be the base of every updated image. If a checkpoint image is used, then the steps to generate the map with gridlines will be bypassed. The time to create checkpoints can be generalized for the gameplay. During a turn, the distance indicators will only occur when they attack and move. Once the position is selected, they should be removed. Therefore, the step before adding the distance indicators will be to save the current state of the displayed image and then modify it. Once a space is chosen, then the image can revert back to the state before it showed the distances. The only unavoidable state is during an attack when an NPC must be removed. If there are multiple NPC's on the display, there is no way to know which one will be attacked and removed first, and the state cannot be saved to preemptively save time.

6.4.2.4 Wi-Fi Direct Class

The game software will act as the group leader for all the Wi-Fi direct connections. All mobile devices will connect to the computer and the computer's game software will process and respond to requests made by the mobile devices. To more efficiently develop this functionality, an external library of Wi-Fi direct functions will be obtained. This class will be running on the Windows operating system; therefore, the Windows SDK will be used to implement the functionality of this class. The Windows SDK provides libraries that enable the Wi-Fi direct protocol.

Specifically, the WiFiDirect namespace will be used to provide the API to create the connection. The WiFiDirectDevice class will be used to manage the devices connected to the network. The WiFiDirect namespace includes a method to broadcast the Wi-Fi direct network to nearby device and a method to listen for connection requests. When a device is trying to connect, it will accept that device and create a WiFiDirectDevice object for the newly connected device. The methods required for the project's Wi-Fi direct class will be to initialize the Windows SDK WiFiDirect classes and handle requests from the game as well as communication request from the different player's mobile apps. The WiFiDirectDevice class includes properties for ID and connection status. The ID will make mapping an in-game character to a physical player straight forward as each device will contain a different ID. The class also provides methods to close the connection which will be implemented when the game is being ended or saved. In order to open file transfer capability, there is a method that will return an endpoints object for the two devices.

The endpoints object will then be passed to a socket class to open the communication channel. The socket class will be determined based on the necessary data that will be transmitted. When transmitting data, a stream socket will be instantiated. The stream socket method uses TCP (Transmission Control Protocol) to transmit data. TCP enables flow control and error checking to create fast transmitted data that is accurate. Once transmission is complete the close method can be called to end the channel connection. The socket will be connected whenever files need to be transferred from mobile device to computer or vice versa. Display methods that involve user provided images will form this link to receive the images off the user's mobile device. Also, the save game and load game methods will require a communication link. For save game, the game state will be saved into a file and transmitted to the Game Master's mobile device so that they can carry the game wherever they go. When the game needs to be loaded, the computer will need the load file from the Game Master's mobile device and therefore a socket will be required.

Table 44 lists some of these methods. The constructor will be used to set up the Wi-Fi direct network on the computer's end. Once the constructor's procedure is complete, a broadcast of the network will be in place for nearby Wi-Fi direct enabled devices. The connection of players to the game will occur at the game's bootup. Before any gameplay logic begins, all players will connect to the device. The class will begin a loop scanning for nearby join requests and handle those requests. The scanDevices methods will scan and queue all join requests before connecting each request in a first in first out order. Connection of devices to the computer is not enough to setup a transmission channel. Endpoints will need to be created and maintained for transmission. The createEndpoints method will take the local host's information as well as the device that is passed into the method and create an EndpointPair object that will control the communication. When the game ends, the connection between the computer and the mobile devices will need to be closed. The disconnectDevices method will iterate over all connected devices and close the connection. Once this is done, the class can run the destructor to free memory.

Table 44: Wi-Fi Direct Class Methods

Method	Method Purpose
Wifi(): Wifi	Constructor to instantiate Wifi object and return a reference of that object. The Constructor will turn on the broadcast to Wi-Fi direct enabled devices.
scanDevices(): void	A method to scan for nearby devices requests.
connectDevice(): void	A method to connect a device to the network and add the connection to the list of other connections.
createEndpoints(WifiDirectDevice): void	Creates an endpoint pair object to setup a transmission channel.
disconnectDevices(): void	Runs a procedure to close connection for devices on the network.

The class will need to maintain the devices connected and the communication endpoint pairs. Table 45 shows the fields that will be maintained by the class. The connection field will be a variable length list that holds each of the devices. The order of the list will be based on the order of connection to the server. To pair a device with a player profile, when a player object is being created, the ID associated with the connection will be maintained. Whenever there needs to be a message sent from the game to a specific player, the ID will be compared, and the message will be sent to the device with the matching ID. The endpoints will be similar to the connections. Each

endpoint will be paired to a specific device. For the computer to scan for nearby devices, an instance of the `WiFiDirectConnectionListener` has to exist. Methods that belong to that object will be used to check for connection requests.

Table 45: Wi-Fi Direct Class Fields

Attribute	Descriptions
private std::vector<WiFiDirectDevice> connections	The list of devices connected to the network.
private std::vector<EndpointPair> endpoints	A list of endpoint pairs for connected devices.
private WiFiDirectConnectionListener listener	An object to listen for connection requests.

6.4.3 Programming Language Selection

An obvious choice for the paradigm of programming languages to be used for the project's software is object-oriented. The paradigm offers the ability to use the powerful tool of inheritance. Inheritance allows for classes to be derived from other classes. This allows functionality to pass down from parent classes to subclasses. Function overloading allows parent classes to act as frameworks for subclasses without creating specific methods for functionality. This allows subclasses to be customized to the extent that they are different from the parent. If inheritance or class creation is not required for a specific functionality, the paradigm allows programmers to fall back to an imperative style of programming. The imperative paradigm is a style where statements are listed in sequential order and executed until completion. The C programming language is an example of the paradigm. It is important to note that the programming team is very familiar with the C programming language and that style of programming. The android app subsystem will require a graphical user interface. Therefore, the language choice will either need to implement their own user interface libraries or allow for third-party frameworks. Some popular object-oriented languages that are common industry that will be considered are Java and C++.

6.4.3.1 C++

C++ is an object-oriented programming language with good portability and community support including libraries that help in UI development and object tracking. Besides basic syntax variations, the addition of classes and inheritance is the major difference between the two. In C memory management was completely controlled by the programmer and required constant checks to verify integrity of the memory space to avoid memory leaks and crashes. C++ introduced class constructors and destructors for created objects as a way to relieve some of the requirements put on the programmers for memory management. These two methods work similar to the memory allocation functions, `malloc()` and `calloc()`, and memory freeing functions, `free()`, in C. The difference is that they create and destroy objects and do not required the size of memory to build. The compiler will maintain the different object sizes and when their constructor and destructors are called the memory associated with that object will be allocated or freed. Each class has a constructor and destructor inherently and they can be overloaded to include additional functionality. Some modern languages have custo garbage collection. C++ does not have this built-in feature. The C++ language can be modified to become non-standard, but include additional useful functionality. Windows has their own version of C++ to allow for the windows API to run properly. There are some versions of C++ that implement a garbage collector. These versions will be ignored as in depth research would be required for non-standard builds of C++. Garbage collection algorithms could create performance issues within the software especially being a non-standard algorithm.

C++ was developed before it became popular to include user interface libraries as standard libraries of the language. Therefore, third-party libraries are required to create any graphical interfaces. Some popular libraries are QT and wxWidgets. These libraries enable the ability to construct different interface components through simply function calls. Documentation is available from each of their respective websites. Since the interface will be for the mobile app, documentation is available for porting C++ projects through Android Studio to build an app.

6.4.3.2 Java

Java is a popular language with plenty of documentation available from its creator, Oracle, about built-in libraries and classes. Java runs on the Java virtual machine, the JVM, which adds a level of virtualization to programs. This virtualization allows Java to implement their own garbage collector that will maintain memory management for the programmer. All memory accesses run through the JVM which allows Java to implement their own memory replacement algorithms. The garbage collector can be troublesome in some cases where it will release memory before it should. This potentially can cause some negative performance issues if the memory released was important to the software.

Java was developed to have built-in interface libraries called Swing [27]. Swing aims to be a platform independent framework to allow similar looks across different platforms. There is plenty of documentation on methods and class information all provided on the Oracle website. Also, Java is the native language for android app development. There would be an advantage to developing the app in Java as no porting would be required.

6.4.3.3 Conclusion

After looking at the two languages, C++ will be the chosen language. It will require memory management to left up to the programming team; however, there will be no concern about built-in features taking control of the program. Different options will be explored when dealing with the graphical user interface libraries, but libraries will be available to create a design that can be seen across multiple platforms. If development were to ever move or expand to a different platform, design code will not have to be changed. Lastly, the programming team has experience with the C programming language and the transition to C++ will not be harsh as learning a new language.

6.4.3.4 Visual Studio IDE

For our game running on Windows, we used the Visual Studio IDE to code, test, and finalize our software written with C++. Microsoft has the proprietary rights to Visual Studio which is used to create computer programs, websites, and mobile applications. For our purposes, we did not use Visual Studio to create our mobile application as we had the IDE specific for Android OS, Android Studio for that purpose. Visual Studio has many features that are convenient to a developer to create their own Windows applications.

Visual Studio is integrated with a Microsoft development platform called Windows API that has beneficial features that streamline the development process. Since the platform is so heavily integrated with and built for Windows, it can do some things that other IDEs cannot. Most external IDEs only have access to the base features of Windows such as file management, threads, and processes unless given permission to do otherwise. Windows API allows Visual Studio to handle tasks such as powering on/off the system, creating a background process called a Windows service, and managing user accounts. A Windows service can run in the background regardless of which user is signed-in and is helpful in trying to start our game as soon as we turn our computer instead of loading a file. Windows API helps with built-in functions to ease the UI development process

in allowing graphics to easily be placed on displays. The UI is given functions that control the size of a window, scrolling, and IO which helps in displaying on our screen instead of a regular monitor or TV display. The Windows API also includes features that make feedback directed at players such as loading and toolbars easy to develop.

Visual Studio includes a few convenient programming tools that help us develop our game software with higher quality and simplicity in mind. While all of our team has programmed in C before, C++ has some additional features and upgrades that we needed to get used to while coding our project. The native syntax highlighter and code completion assists new developers in any supported language. While experienced programmers might consider these features as crutches, while getting used to a new programming language these tools will not only help developers fix mistakes but also provide hints as to why certain syntax is not allowed. The IDE has native help guides to get started on creating a new application as well. If all else fails, the community of online programmers has probably experienced a similar issue to an error that a developer may encounter. Stack Overflow [28] is a great website that can help all levels of developers with syntax, logical, or semantic errors.

Visual Studio offers debugging and testing tools for its developmental environment. The debugger offered works as a source-level debugger and a machine-level debugger for any supported Visual Studio programming language. The debugger can be run during the execution of a developer's code which will then assist with variable and stack management. Setting breakpoint allows the debugger to take a break during execution which can then be told to keep running after a developer finishes analyzing the previous results. The debugger also supports step-execution that creates breakpoints after each line of code allowing execution of one line of code at a time. Visual Studio also has tools for easier unit-testing including a test explorer, Microsoft's own unit test framework for C++, and code coverage. The test explorer allows a developer to see all the unit test results with Microsoft's framework. After running all of the unit-tests, the code coverage feature alerts the programmer of code that has not been tested with the current unit-tests. There are more add-ins and plugins designed for Visual Studio that extend the range of capabilities the native IDE supports.

6.4.4 Multithreading

Since we believe that the execution of our game will be taxing on our CPU (Central Processing Unit), we explored the option of multithread programming. Multithreading is the ability to use more than one core of a CPU at one time to break up the load of execution through the distribution of threads. This leads to faster overall execution of the code, as it allows resources not would normally be idle to be utilized. The problem that can be encountered with multithreading is when two or more threads are trying to use the same resources at one time. For example, if two different threads, running on separate CPUs, need the ALU (Arithmetic Logic Unit) then one thread will have to wait until the other is finished before using the resource which can slow down overall execution if the code primarily uses a the hardware unit. There are two main types of multithreading that are worth discussing for our group and they are interleaved multithreading and simultaneous multithreading.

Interleaved multithreading's main purpose is to get rid of data dependencies that can clog the execution of a program. All threads on each CPU should not wait for the output of a thread that is outside the range of the current core. This means that a thread in core one of the CPU will not rely on the output of threads from cores two through four. Every cycle of the CPU there is a context

switch for the code which results in no pipeline flushing. The problems with interleaved multithreading is its performance for single threads, so if there is a long thread then the process will actually take more time to execute than single core processing. Interleaved multithreading also requires specific hardware to run. CPUs need to have multiple register banks and other hardware necessities that are necessary for interleaved multithreading.

Simultaneous multithreading uses parallelisms to superscalar fetch multiple threads on each CPU cycle. The idea for simultaneous multithreading comes from the fact that threads create unused issue slots when normally executed which are then allowed to be used for another thread on the same CPU. In simultaneous multithreading's single-thread mode, the CPU is operating at full performance for that CPU core. This leads to easier handling of pipeline bubbles and long latencies, but simultaneous multithreading is also the most complex to establish in a program. The CPU also has hardware requirements that need to be met in order to use simultaneous multithreading in which the CPU needs to be able to fetch more than one thread at a time for this type of multithreading.

Since we are going to be using C++ as our programming language for our Windows game, we researched what it takes to optimize our game using multithreading. The native C++ language does not have built-in support for multithreaded programs. However, Visual Studio has an external add-in library called MFC (Microsoft Foundation Class), which allows C++ developers to use multithreading. Developers can create their own threads to handle background or maintenance tasks. We can use multithreading in our project by giving each thread that has to deal with object detection to one core, while the game itself can run on another. Although multithreading is useful in speeding up execution, debugging and testing becomes harder the more cores utilized.

The requirements expected to use multithreading are absolutes which cannot be ignored, and if we do not have the correct technology then multithreading will be impossible for table. At the same time, we must also consider that costumers of our gaming table may not have the correct hardware for multithreading, so we would have to program contingencies for single core executions. If we decide to implement multithreading, we can also make hardware requirements for using our table as most modern CPUs, including our team members' PCs, have access to multithreading.

6.5 Wi-Fi Direct Interface

Wi-Fi Direct [29] is also known as Wi-Fi peer-to-peer (P2P). The standard allows devices to directly connect to each other without connecting to a central access point. Devices have dedicated software and hardware to enable connection to other devices without requiring mobile data or internet connection. Devices have the ability create a communication link between one or many devices. This communication link is similar to a Bluetooth connection, but with the improvements to bandwidth and transfer speeds that are provided with Wi-Fi. Wi-Fi direct creates a wider network and faster data transfer for devices in close proximity.

Since Wi-Fi direct requires dedicated hardware and software, not all devices will accommodate this feature. For an android device to accommodate Wi-Fi direct with software, the device must at least be running Android 4.0. Android 4.0 is a version of the operating system that was released in 2011. According to the Android Developers website, only 0.3% of android users are still running a version older than android 4.0. Based off this statistic, Wi-Fi direct will be able to accommodate 99.7% of users. This also assumes that devices have the required hardware. After the software

implementation became standard in the android operating system versions, most android device manufacturers implemented the hardware required to run the standard.

Android's website provides documentation on how to implement this protocol through examples and documentation of class methods. Some specific standards that are required for Wi-Fi direct to work are that the MAC address randomization is required for connection. Although the MAC address is an ideally unique identifier for each device, the address that is used for connection must be a random value that is generated with each new connection.

It is important to note that Wi-Fi direct is disabled for iOS devices. Instead they use their own framework called Multipeer [30] to accomplish peer-to-peer connections. Any code written for Wi-Fi direct will not directly be usable in the iOS environment and must be converted for operation in the Multipeer framework.

For a computer running the Windows operating system, Microsoft offers windows run time libraries. These libraries can be accessed through Microsoft's Visual Studios IDE. These libraries provide classes that will be used to enable and run Wi-Fi direct. Additionally, these libraries will be used to setup the sockets required for file transfer.

Wi-Fi direct connection is created using a server and client. Although the server-client model refers to a different approach to communication, the terms are just reused to indicate the devices hosting the connection link and connecting to a connection link. The steps for the Wi-Fi direct connection process can be seen in Figure 44. The process branches into two separate paths depending on the type of device connecting to a network. The server device hosts the network that the client devices connects to. The sever can also be called the group leader. In this project, the group leader will be the Windows computer and each device will be a different android app.

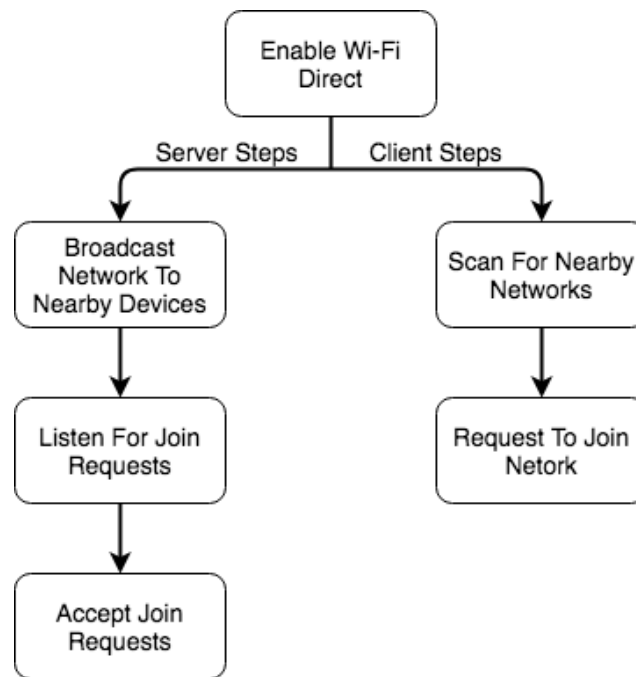


Figure 44: Wi-Fi Direct Connection Steps

Wi-Fi direct on windows only creates the connection point between multiple devices as can be seen in Figure 45. A WebSocket will need to be implemented in order to transfer data. A WebSocket is an agreement between two devices about when and where information will be transferred. They are initiated by the client sending the server a request to form a channel. The server will then respond by accepting the request. The client will then be able to transfer data over the connection. When the client is done the server will acknowledge and close the socket. The protocol can be seen in Figure 45. Since a WebSocket is only an agreement between two devices, each device will need to have a WebSocket initialized with the server to create all of the transfer links.

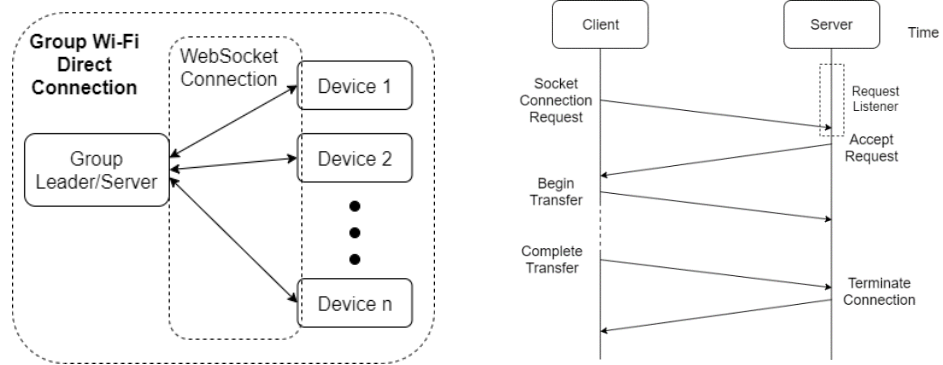


Figure 45: Wi-Fi Direct and WebSocket Protocol

6.6 Windows API

As stated before in this document, when discussing the Visual Studio IDE, Windows API is a Microsoft development platform open to use for personal projects. The Windows API is a collection of other APIs (application programming interfaces) that are collectively used in the Window's operating system. Microsoft offers developer support for Windows API through their software development kit named Microsoft Windows SDK. The Microsoft Windows SDK, specifically the one we'll be using called Windows 10 SDK, contains libraries and functions that we can use for our project. The SDK also includes documentation for describing functions' applications and tools to simplify common tasks and better implement our communication protocol. Windows 10 SDK is available for free for anyone with a legitimate Windows 10 Home, Professional, Education, or Enterprise license which all of our group have access to. below shows the twelve basic categories that all Windows API functions fall into which are collected into groups and their purpose for a developer.

The API modules that are described in

Table 46 mostly work with the interactions of the application and Windows OS. Some APIs work with networking or external devices, but Microsoft has developed different technologies for interactions with different Windows applications. COM (Component Object Model) is the binary-interface that allows inter-process communication object creation. These objects can be used across machines that the object was not created in. This can be useful if we ever need to communicate our application with another computer to host a game. To easier create better and more elegant UI elements, Microsoft has the Universal Windows Platform (UWP). Using the UWP for app design is another option when designing an app instead of using just Windows API.

Some functions and APIs are no longer supported, or the features they implement can be done easier through the Visual Studio IDE or external libraries. APIs such as the User Input or Diagnostics fall into these categories. Audio and some visual functions such as show on screen are also easier thanks to these IDEs. Sensor data collection and interpretation of that data is done with our printed circuit board instead of Windows API. We also do not have a need for the security API as no personal or sensitive data is being accessed, sent, or received. Some of the categories in

Table 46 do not have functions and instead are programs or other recourses such as Windows Installer that creates the installer exe for a client.

Table 46: Windows API Categories

Category	General Purpose
User Interface	Displays output for the user, creates prompts for user input, and handles all other tasks that handle interactions between an application and the user.
Windows Environment (Shell)	Controls the windows of a user application. Can set data from windows user such as audio, images, and time on Windows files from a developer.
User Input and Messaging	Direct Manipulation pre-declares certain behaviors for an application such as pan and zoom. Direct Manipulation also handles putting touch detection on a separate thread than the UI thread.
Data Access and Storage	Handles transferring files and data between a client and server, disk management, and offline file management.
Diagnostics	This API handles troubleshooting applications by debugging, error handling, and network monitoring.
Graphics and Multimedia	Handles the audio, video, and graphics part of the user application through the use of Core Audio and DirectX. Allows developers to integrate their own audio devices such as speakers.
Devices	Sends signals through a distributed bus for controlling peripherals such as a temperature sensors, LEDs, or cameras.
System Services	Gives the developer access to use the recourses of the computer such as memory and threads for their application. It also allows power management to not draw too much power for the application.
Security and Identity	Allows the developer to secure their application through the use of encryption and passwords at logon. Authentication of users is also included in the features in this API for a developer.
Application Installation and Servicing	Windows Installer allows a developer to create an installer that someone can download to install their application. The installer can be set for the application to get updates from the internet.
System Admin and Management	This API extends and enhances the functions given by the Application Installation and Servicing API. The restart manager and task scheduler are two of the tools that a developer has access to.
Networking and Internet	Enables the communication between a network and an application. The communication protocols supported include Wi-Fi, Bluetooth, and Wi-Fi direct.

6.6.1 Application with our Project

In this section, we describe the various functions and methods that are useful to us from the Windows API library. All of these specific functions may not be used as we may be able to find a better alternative in an external library. The functions are still worth discussing since the tasks they perform must still be in our project.

6.6.1.1 User Interface Integration

The UI elements from this API that we are using will be focused on UI and a few error handling message functions as described in Table 47.

Table 47: User Interface API Functions

Function	Description
CreateMenu(): HMENU	Creates an empty menu
InsertMenuItemA(HMENU hmenu, UINT item, BOOL fByPosition, LPCMENUITEMINFO lpmi): BOOL	Inserts a new menu item to the handle identifier for the menu with an identifier for position and information of the new menu item. Returns BOOL true on succeed. Must call DrawMenuBar when changing the window.
DrawMenuBar(HWND hWnd): BOOL	Updates the window specified, true returns if succeed.
EndMenu(): BOOL	Ends the active menu. True returns if succeed
MessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType): int	Creates a modal dialog box that has a set of buttons, an owner window, a title, and text to be displayed. The int that is returned describes which button was pressed.

The menu functions will allow us to set up the initial game with a menu and also to change parameters of the current game. The message boxes will inform the player of any errors that occurs within the game or options that the game master can choose. Animations and images are difficult to make using this API so we will use a different API or the Visual Studio IDE helper to create the other UI elements.

6.6.1.2 Windows Environment (Shell) Integration

The shell elements allow us to create handles for specific icons, unique file names, and loading a specific Windows profile on startup as shown in Table 48.

Table 48: Shell Functions

Function	Description
ExtractIconA(HINSTANCE hInst, LPCSTR pszExeFileName, UINT nIconIndex): HICON	Returns a handle to the icon specified with the string.
LoadUserProfileA(HANDLE hToken, LPPROFILEINFOA lpProfileInfo): USERENVAPI BOOL	Takes a token for the user and a struct for the profile info and loads the specified user profile. True returns if successful.
PathMakeUniqueName(PWSTR pszUniqueName, UINT cchMax, PCWSTR pszTemplate, PCWSTR pszLongPlate, PCWSTR pszDir): BOOL	From a template, creates a unique file path that a developer can use for creating new files. True returns if successful.

Extracting icons can help us manage our non-physical characters such as monster creatures. We will load a profile automatically when starting the computer for our game. Since we will be using the host's PC for storing files with game data, creating unique paths is important to not overwrite files.

6.6.1.3 Data Access and Storage Integration

The functions in Table 49 let us manage offline files since our game will solely be running on the host's PC instead of a database.

Table 49: Data Access and Storage Functions

Function	Description
GetDiskFreeSpaceExA(LPCSTR lpDirectoryName, PULARGE_INTEGER lpFreeBytesAvailableToCaller, PULARGE_INTEGER lpTotalNumberOfBytes, PULARGE_INTEGER lpTotalNumberOfFreeBytes): BOOL	Gets the information required to create a new directory and files on a host computer by accessing the space available to the game. True returns if successful.
CreateDirectoryExA(LPCSTR lpTemplateDirectory, LPCSTR lpNewDirectory, LPSECURITY_ATTRIBUTES lpSecurityAttributes): BOOL	Creates a new directory to store game files with the specified template to create unique directories every time. Returns errors if directory exists already or path not found.
SetCurrentDirectory(LPCTSTR lpPathName): BOOL	Takes the path of directory in the host's computer and changes the current directory for the current process. True returns if successful.
CreateFileA(LPCSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes, DWORD dwCreationDisposition, DWORD dwFlagsAndAttributes, HANDLE hTemplateFile): HANDLE	Creates a new file to write data into using the new file name, access security, and a file template. Returns a handle to the new file.

We need to check if the host's PC has enough storage to handle our game and to create offline files so the game can be run again for a future session if needed.

6.6.1.4 System Services Integration

Alongside the functions shown in Table 50 below, we must use a thread manager in the IDE to give priority to threads.

Table 50: System Services Functions

Function	Description
CreateThread(LPSECURITY_ATTRIBUTES lpThreadAttributes, SIZE_T dwStackSize, LPTHREAD_START_ROUTINE lpStartAddress, __drv_aliasesMem LPVOID lpParameter, DWORD dwCreationFlags, LPDWORD lpThreadId): HANDLE	Creates a thread using the security attributes, initial stack size and address, a pointer to the variable being passed, and a thread id intensifier. Returns a handle to the new thread if success
AttachThreadInput(DWORD idAttach, DWORD idAttachTo, BOOL fAttach): BOOL	Takes an identifier of the thread being attached and the identifier of which thread to attach to. Returns non-zero value if succeeds
GetSystemTime(LPSYSTEMTIME lpSystemTime): void	Takes a pointer to the system time struct, SYSTEMTIME, and puts the current time in that pointer's data)

The first two functions are a part of the thread management that our project must perform to make sure events don't overlap or happen before they are scheduled to. System time can be used as part of our interface for displaying game information such as turn order and options.

6.6.1.5 Networking and Internet Integration

Since we will not be using Wi-Fi or Bluetooth in our project, the functions in Table 51 show the application of Wi-Fi direct that we have described earlier in our document.

Table 51: Wi-Fi Direct Functions

Function	Description
WFDOpenHandle(DWORD dwClientVersion, PDWORD pdwNegotiatedVersion, PHANDLE phClientHandle): DWORD	Opens a handle to the Wi-Fi Direct service. Takes the version used, and a pointer which it updates with the handle. Returns a flag for error handling for the developer.
WFDDot11MacAddressStartOpenSession(HANDLE hClientHandle, PDOT11_MAC_ADDRESS pDeviceAddress, PVOID pvContext, WFD_OPEN_SESSION_COMPLETE_CALLBACK pfnCallback, PHANDLE phSessionHandle): DWORD	Takes the handle from WFDOpenHandle, a pointer to the MAC address of the target Wi-Fi device, and a handle to the current session. Returns a flag for error handling.
WFDCloseHandle(HANDLE hClientHandle): DWORD	Closes a handle taken from WFDOpenHandle. Returns a flag for error handling.

When using the Wi-Fi direct service and the functions in Table 51, we must pair the devices ourselves through the Windows Pairing experience user interface. Our application then needs to open a handle using WFDOpenHandle that is used for the remainder of the session. It is also good practice to close the Wi-Fi direct service to release the resources allocated by the computer to it.

6.7 GitHub

GitHub is a free hosting service provided by Microsoft that we used for developmental software control during our project. GitHub offers collaboration services that are great for development teams including version control, code review, and code merging. Version control allows a developer to see all changes from one version of the code to another. This helps not only keep track of who is contributing to the changes but also to roll back the version to a previously working state if there is a program breaking bug. Code review allows people to purpose changes which will then be either approved or denied for changing the overall code. This can also help with peer review for syntax, logical, or semantic errors. Code merging allows the use of status checks to make sure that branches of code are protected and will work before they are pushed onto the project.

7.0 Test Plan

7.1 Hardware Testing

All components of the multi-touch table are tested independently to ensure functionality before integrating them into larger subsystems. Each subsystem is then tested before being installed in the table and integrated into the overall system. More realistic test cases are generated for the entire system once the table is complete, detailed in section 7.3. These tests are carried out indoors in either air-conditioned homes or university buildings as the table is designed to be used in a consumer's home or similar setting, and some testing equipment is freely available through the UCF Senior Design Lab.

7.1.1 IR Camera

Prior to modifying the camera to record the infrared spectrum, the appropriate driver is installed, and the camera is connected to a Windows 10 PC using a USB 2.0 port to verify that a video feed can be captured with the desired image quality and frame rate. This test is passed if the video feed accurately captures the scene and the feed maintains at least 640 x 480 pixels at 30 frames per second. The original intention was to point the camera at infrared LEDs to ensure that IR light could be detected after the IR filter was removed. Once it was discovered that the IR filter was extremely weak, the filter was instead left in the camera to reduce risk of damage to the rest of the hardware. This test is passed if the camera feed shows a lit LED while human eyes detect no such light. After replacing the IR-blocking filter with a visible spectrum-blocking filter, the camera was pointed again at visible scenes to ensure the visible spectrum-blocking filter was functional. This test is passed if the video feed is a uniform color when no infrared sources are present. Finally, the IR light detection test is repeated to ensure the visible light-blocking filter is not also blocking the infrared light, with the same pass/fail criterion.

7.1.2 IR LEDs

The IR LEDs are powered using a 9 V battery and a breadboard circuit according to their datasheet specifications. The lit LEDs are pointed at a white wall with an area marked out to match the touch surface dimensions. The IR camera is used to verify that the wall is illuminated. This test is passed if the change in illumination when the LEDs are on vs when they are off is distinguishable on the IR camera feed. Adjustments are made to the LED array to illuminate the area as evenly as possible. This test is repeated on the inside of the table once construction is complete.

7.1.3 Surface Material

Test images are projected onto the touch surface/diffusive material to ensure that rear projection will produce a visible image with clarity matching the requirements specification. This test is passed if text 1" tall can be read from 2' from the edge of the table at a 30° angle. The material is then illuminated with the IR LEDs, and objects and fingertips placed on the other side. The IR camera was used to ensure blobs would appear that could be used for touch detection. This test is passed if blobs can be distinguished by the blob-detection software in at least 95% of the test trials.

7.1.4 Power Supply

A multimeter was used to verify that the power supply provided the desired output voltages, current, and wattages with an appropriate test load. This test is passed if the measurements fall within $\pm 10\%$ of the desired values.

7.1.5 Light Sensor

The light sensor is powered using a 9 V battery and a breadboard circuit according to its datasheet specifications and placed under a lamp of adjustable brightness in an otherwise unlit room. The lamp's brightness is varied repeatedly to ensure corresponding changes in current flowing from the sensor in accordance with its datasheet. The test is passed if appropriate changes in current are observed.

7.1.6 Timer

7.1.6.2 Timer Display

The quad seven-segment LED display is tested using a breadboard circuit by first lighting each relevant segment individually. If all segments light as expected, this test is passed. The display is then connected to the Raspberry Pi for program testing. First we display 20 pre-generated random numbers, to ensure the display refreshes at an appropriate rate which avoids flickering, and multiplexing is properly implemented. If the displayed numbers match what has been sent, and no discernible flashing occurs, the test is passed. This test is also used to confirm that functions for commanding various numbers are correct in the software.

7.1.6.2 Timer Buttons

Each button of the timer is tested to ensure that they can generate interrupts for the microcontroller. The buttons are connected to visible LEDs in a breadboard circuit and used to turn the LEDs on and off. The test is passed if the lights turn on and off as expected in all cases. Once the microcontroller is acquired and the timer is programmed, each button is tested to ensure they triggered events as expected on the software side, i.e. the power button turned the timer on and off, the time increment buttons added the appropriate value to the "time" variable, and the clear button reset the "time" variable to 0. Each button is pressed at least 10 times in each of the operating modes. The test is passed if all observed behaviors match expected behaviors. Further, the rollover events triggered by incrementing by 1 minute at 99:xx, incrementing by 1 second at yy:59, and incrementing by 1 second at 99:59 were tested for 10 randomly selected choices of xx and yy, to check that the rollover behavior was well-defined and properly implemented. The test is passed if all rollovers occur as expected.

7.1.7 Speakers

The speakers are connected to a PC and the sound effects chosen for the game are played to verify that the speakers are functional prior to installation in the table. This test is passed if the sound effects are undistorted compared to another functional pair of speakers. Once integrated into the overall system the same sounds effects were played again to ensure nothing was damaged and that the sounds were not muffled and made inaudible by the table. This test is passed if any muffling can be countered by increasing the volume while maintaining little to no discernible distortion.

7.1.8 Accent Lights

The accent lights are connected to the controller and an array of color and effect commands which are to be used in the game are sent to the lights, such as "light orange for 1 second." This test is passed if accurate responses to the commands are verified.

7.1.9 Temperature Sensor and Fans

The fans are powered in accordance with their datasheet to make sure they are not dead on arrival. This test is passed if the fans turn on as expected when powered. The temperature sensor is tested by comparing its readings to those of a traditional thermometer for accuracy. This test is passed if

there is at most 0.1° difference within 10° of the operating range of the table. The temperature sensor was set to trigger a fan response in temperatures exceeding the maximum operating temperature of 32°C. The temperature sensor is then warmed using a hair dryer to ensure the response triggers, then allowed to cool to ensure the fans turn off again. The test is passed if the fans turn off when the temperature sensor crosses the 32°C threshold.

7.1.10 Wireless Communications

The wireless connectivity which allows communication between mobile devices and the table is tested by first ensuring that mobile devices can locate the access point and establish a connection. This test is passed if our Android phones can find the access point. The connection is then tested by storing a lookup table in the table's microcontroller, sending requests for information from the user's phone, and verifying responses were received and were appropriate. This test is passed if the returned values match those in the lookup table.

7.1.11 Table

The physical structure of the table is tested prior to the installation of critical components at risk of damage. First uneven forces are applied to the sides and top of the table in the form of a single individual pushing or pulling on specific areas. The test is passed if the table displays no visible damage and all structural elements remain attached. Maneuverability is verified by having two healthy individuals move the table. Weights are added to simulate the load of the electrical systems. The test is passed if the table is moved at least 10 feet in one trip, and then through a doorway without injury or damage. After installation the security of each critical component's fastening is tested by light pushing or pulling as appropriate. This test is passed if the components remain fastened and do not move significantly unless designed to be removed by such a force. The table maneuverability test was repeated once all systems were integrated to make sure the table could be moved safely without electrical components coming loose.

7.2 Software Testing

The software can be divided into two distinct groups for testing; the mobile app and the computer applications. The mobile app will be built using Android Studio and within Android Studio is the ability emulate the app. This will allow testing to be done locally on a test machine for the app. The computer applications will require some methods to test the results of important functionality.

7.2.1 Mobile App Testing

Mobile app testing will verify the functionality of the user interface on the app. Much of the functionality of the app requires a communication link to the game software, and that will be tested in a later section. The user interface can be tested through the emulation software within Android Studio. Each menu should be executed and traced to verify that each window is appearing when it should and disappearing when its purpose has finished. Local variables tied to each window should be checked to verify that the state of each window is maintained. This can be done by setting constant values for different fields on each window that requires input data. Then cycle through different windows before coming back to each window with inputted data and verify that data is all showing and not lost.

7.2.3 Communication of App to Game Software Testing

The mobile app connects to the computer program to set up a Wi-Fi direct connection. The connection can be tested by looping through the process and outputting the details of the connection when the connection is made. The connection should be maintained throughout the

length of a full game. A testing procedure can be created to initiate the Wi-Fi direct connection and have the computer end output any connection drops during the duration of the test.

The Wi-Fi direct connection is only half of the protocols required for communication. The socket connection will also need to be verified. This can be done by going through the procedures of setting up a socket and outputting values that link to that socket channel. Data should be sent through the channel to verify that each end can send and receive data. Max datagrams for each method should be passed to verify the integrity of files of all sizes. A series of transfers will be taken, and transfer timings will be recorded. These timings can be used to calculate the average transfer speed as well as view the longest transfer time. These will be important metrics when evaluating the functionality of the communication link.

7.2.4 Object Detection Testing

According to the engineering constraints in Table 2, there are three constraints related to the object detection software that will be verified.

- Objects of 0.5” diameter or larger must be able to be detected
- An object location must be within 0.5” of its true location
- Object movement must be accurately detected with 95% or higher certainty

Testing the accuracy of the software requires both physical testing as well as software testing. A testing module will be written to test features independent from the game logic. The game logic constrains the functionality of the software to allow the software to directly relate to what is required. Excluding the game logic allows the software to detect objects across the entire display without any methods for interfering. However, the methods related to coordinate to grid space translation should remain active to test correctness of that method. The testing module will read all objects and output the results into a file that can then be checked for accuracy. Physical movement of objects on the display is required to test the software because the data input is dependent on physical objects.

The playable area of the display cannot have any dead zones where objects are not detectable or incorrectly detected. Detection over the entire display must be verified. To check for any issues with the display, testing can be done by placing objects over the entire display and checking if the software is detecting each object. Verification can be done by running the testing module and verifying the output data with the input object locations. Testing the edge of the playable area will confirm that the range of the camera is correct. An issue that may arise is that the calibration of the software is not accurate enough the translation will be incorrect. If this is the case, all translation data should be off by the same margin. This will be a sign to recalibrate the system and restart testing.

To verify that objects of a certain size are detected, object of the minimum diameter threshold will be placed on the display and the data will be analyzed to see if all objects were detected. Objects smaller than the minimum diameter will also confirm the requirement. If the software can detect objects at the minimum size, it will be able to detect objects of larger sizes. After the coordinates are translated to grid placements, it can be verified if the object is located with the minimum range of its true location by comparing actual placements to detected placements. The final constraint will require movement of objects on the display. This requirement will need to be tested by physically creating movements on the display. To accurately test this condition, at least 100

movements should be tested and at least 95 of those movements need to be detected. Ideally, visible light will be blocked by the camera filter. These constraints should be tested in differing environmental brightnesses to confirm that there will be no noise distortion with the setup.

Once the detection over playable area is confirmed to be functioning as intended and constraints have been verified, then the game logic should be reenabled. Methods related to invalid piece placement checking can be tested by creating a loop that constantly looks for a player piece to be moved. For each iteration of the loop, a piece can be either placed in an invalid location or a valid location and the behavior of the software can be observed. Invalid locations can be defined as pieces placed nearly equal in between two or more grid spaces. Valid locations can be defined as pieces placed within the grid space or at most 40% in another space. The results of these actions should be in line with the protocols defined in the software design. Invalid locations should create a prompt to be fixed and valid locations should be accepted. The tests should be independent of the mobile app; therefore, prompts should be outputted to either the console or an output file. These tests will be conducted over the entire playable area. A specific case to observe the software's behavior would be placing a piece partially out of the playable area.

7.2.5 Save and Load Game Data Testing

There is one engineering constraint from Table 2 that relates to save and load game data.

- There must be at least 20 locations tracked and saved on exit

Since NPCs will be digitally created and tracked, there will be a maximum of 4 physical locations tracked. All other locations will be for the NPCs. Testing will be conducted by placing the save and load game methods within a loop. The software will have access to the expected result files and be able to check on each iteration. The goal is to verify that within the save and load game data methods there are no bugs that cause corruption of the data or save files. By iterating many times in one set time period, it can be observed if there are any issues to the save files that may arise from the code being run in frequent succession.

7.2.6 Display Testing

To test the display software, a testing script will be made to manipulate the displayed image. Different display images will be generated by adding and removing overlaid images. The goal is that the change in display images does not have a high latency and the placement of overlaid images are in the correct locations. To test this, the latency for creating an image with the greatest number of components will be timed. After the image is generated and displayed, the locations of each component will be record and compared to the test data. The latencies will also be timed for adding and removing a single component to verify that the methodology for creating the display image is fast enough. The methods for displaying the move and attack ranges will be tested by running the methods in a loop. After each iteration, the displayed image will be checked for correctness. When updating the map or starting a game, the starting locations for player pieces will be highlighted. These methods will also be tested to verify that the location given to the software matches the location on the display.

7.3 System Testing

To test the overall systems of the table in a comprehensive fashion, several test cases representing possible gameplay scenarios must be generated. The range of parameters possible in the test cases

are given in the following sections. 10 scenarios are generated, each by randomly selecting a parameter value from each relevant category, as each scenario tests a wide number of sub-cases. Unless otherwise noted, all choices are made by random selection with equal probability from the possible options.

7.3.1 Game Options

In each game, there will be exactly one Game Master (GM) and 1 – 4 players. There will also be 1 – 10 maps set up for gameplay. Each player will have 1 – 5 characters to choose from. The number of players and their characters is chosen, then the specific traits of these characters are determined. Then the number of maps is chosen, and a subset of the maps is set up.

7.3.2 Player Character Options

We assign each player character a race and background. We consider 10 race options, some of which have 2 or more subraces to choose from. Some races provide character features to choose between. Each character's race determines their maximum movement each turn. There are also 15 backgrounds to choose from. Each background provides the character with a few skills, items, and features. After race and background selection, the character is assigned 6 ability modifiers in accordance with popular gaming conventions. Each modifier is determined from a corresponding ability score. The ability scores are determined by generating a random integer from a pseudo-normal distribution centered around 10.5 and limited to the range 3-18, then adding race-based modifiers (for example, a tall race may provide an increased strength score). Once a race and background have been selected, a first class for the character is chosen, and the character is given 1-20 levels in that class. Resulting class feature options, including equipment, abilities, and subclass are selected. If the number of levels is great enough, some random ability score increases may also be applied. If the character has fewer than 20 levels in their first class and meets the requirements for another class, there is a 20% chance they get a second class, and a similar process is repeated with the difference that the total levels across all classes must not exceed 20. If the character may qualify for yet another class, there is a 20% chance they get a third class and repeat the process once more. We do not consider cases in which characters have 4 or more classes as they are almost never expected to occur. For each level the character gains, there is a 30% chance of adding a random piece of equipment. Further, for each level the character gains, the amount of currency in their inventory increases. These additions are made on the basis that a high-levelled character has typically been part of a game long enough to have acquired greater resources than a new, low-levelled character.

7.3.3 Map Options

Each game will use between 1 and 3 maps. Each map will be between 10 and 24 squares on the shorter side, and 10 and 32 squares on the longer side. The background for the map is selected at random from 30 images, cropped to the appropriate size given the number of squares and the projected image resolution. Movement restrictions (e.g. walls, cliffs, valleys, etc.) are determined, along with interaction points (e.g. doors, levers, ladders, etc.) based on the map background. Between 1 and 5 additional interactable objects (e.g. books, shovels, torches, etc.) are added to ensure interaction testing can occur. Some of the objects may be removed during the simulation as characters pick them up, and other objects may be added as they are dropped or used by characters (if a character places a torch in a wall sconce, for example).

7.3.4 Game Master Options

For each map, between 1 and 20 non-player characters (NPCs) will be created and controlled by the game master. Each non-player character will be selected at random from a list of approximately 50 options. Each of these 50 options provides the non-player character's ability scores and modifiers, a few skills, and possible actions such as attacks and spells that the character may take on their turn. If the non-player character is a humanoid, rather than another type of creature (e.g. a bear), they may also be given some additional random equipment or abilities. The game master may also choose between 15 music/atmospheric noise tracks to play over the speakers.

7.3.5 Game Simulation

All player and non-player characters will then be placed randomly on the map by first selecting a random x-coordinate and then a random y-coordinate. If the space is already occupied, by another character, an object, or terrain obstacle (e.g. a tree, boulder, or river), the random selection is repeated, until an unoccupied space is chosen. A turn order is randomly assigned to all characters on the map. On their turn, each character takes a random action which is available at their current location, including potential movement to put an interactable object or targetable character in range of their chosen action. This may involve showing the character's movement range, the range of their chosen action/ability, and selecting the area of effect for the action. This may also trigger special effects, such as sounds effects (an explosion as a fireball spell is cast, a clang when a sword is used to attack, a growl as a monster slashes with its claws, etc.) and lighting effects (e.g. colored lighting for player character turn indication, twinkling lights for spells, a warm glow as a fire is lit, etc.). This process repeats for 1-10 iterations, known hereafter as rounds, in which each player character and non-player character takes a turn. Possible actions for each character will include picking up or dropping added usable items to test the ability to modify a character's inventory and available actions during a game. The map is then saved, the table is powered off and on again, and the map is reloaded. 1-3 more rounds are simulated to ensure no unexpected changes to abilities occur during the saving/loading process. At the end of the game simulation, the map is erased from the table and the dungeon master's app, to ensure updates to character abilities and inventories are saved independently.

8.0 Budget

With our design planned, a bill of materials is given in Table 52.

Table 52: Bill of Materials

Item	Unit Cost	Number	Total	Source
OPTIX 30" x 36" x 0.08" Clear Acrylic Sheet	\$18.48	1	\$18.48	Lowe's
30" x 42" x 0.003" Drafting Film Matte, 2-Sided	\$8.16	1	\$8.16	Grafix
EMITTER IR 940NM 65MA 0805	\$0.16	100	\$15.75	Digi-Key
PlayStation Eye	\$8.70	1	\$8.70	Amazon - PlayStation
Floppy Disk	\$0.00	1	\$0.00	Already Owned
CL Eye Platform Driver	\$3.00	1	\$3.00	Code Laboratories
BenQ MX810ST MX713ST DLP Short-Throw Projector	\$178.76	1	\$178.76	eBay - voltarea
12V Fans	\$12.99	1	\$12.99	Pano-Mounts
Speakers	\$0.00	1	\$0.00	Already Owned
Thermistor	\$0.00	1	\$0.00	Already Owned
LTC-4627JF 7-Segment Quad Digit Display	\$3.61	1	\$3.61	Digi-Key
APDS-9306-065 Ambient Light Sensor	\$1.61	1	\$1.61	Digi-Key
RGB LEDs	\$8.99	1	\$8.99	EDGELEC
TLC5940 DIP	\$12.95	1	\$12.95	nooelec
12V 3A AC Adapter	\$11.89	1	\$11.89	IBERLS
Arduino Mega	\$14.99	1	\$14.99	Elegoo
RFP12N10LMOSFETS	\$6.95	1	\$6.95	Riddle Electronics
PCB (x5)	\$40.00	1	\$40.00	JLC PCB
Android fee	\$25.00	1	\$25.00	Google
Table				
Sandeply 1/4" x 4' x 8' Plywood	\$22.92	3	\$68.76	Home Depot
2" x 4" x 12' Lumber	\$6.92	1	\$6.92	Lowe's
Wood glue	\$5.98	1	\$5.98	Lowe's
Screws	\$1.08	1	\$1.08	Lowe's
Hinges (2 pk)	\$3.49	1	\$3.49	Ace Hardware
Caster Wheels (x2)	\$4.99	2	\$9.98	Ace Hardware
Handles	\$3.59	2	\$7.18	Ace Hardware
Paint	\$10.28	1	\$10.28	Lowe's
Total			\$485.50	

This bill of materials does not include small items such as resistors and wires, which we expect to be able to acquire freely from UCF labs or for an insignificant fee. We also exclude tools which may be used in construction such as soldering irons, drills, and paintbrushes, which are already owned by group members or freely available from UCF labs. For online orders, we also exclude shipping and handling fees as those are partly determined by our choice of delivery time and number of items per order, and the fees are typically insignificant. The PCB cost is estimated and may change as we tweak our design. Our original bill of materials, prior to research and decision-making, estimated a total cost of \$487.27, so we are in good agreement with that estimate. Our ideal maximum budget is \$600, so if changes must be made during the prototyping process we have left some flexibility to order additional parts or specialized tools.

9.0 Milestones

The milestones are broken into two semesters, senior design 1 and senior design 2. Research, design, and testing milestones of the project in senior design 1 is shown in Table 53, and integration milestones for senior design 2 are described in Table 54.

Table 53: Senior Design 1 Milestones

Number	Task	Member	Start	End	Status
1	Gather Project Ideas	Group	8/26/2019	9/2/2019	Done
2	Research Viability of Ideas	Group	9/2/2019	9/9/2019	Done
3	Role Assignment and Idea Choice	Group	9/9/2019	9/12/2019	Done
4	Divide and Conquer	Group	9/12/2019	9/20/2019	Done
5	60 Page Draft	Group	9/23/2019	11/1/2019	Done
6	Gather Project Requirements	Erica	9/23/2019	9/30/2019	Done
7	Research Previous Projects	Gabriel	9/23/2019	10/7/2019	Done
8	Research Hardware and Software Standards	Logan/ Gabriel	9/23/2019	10/14/2019	Done
9	Research and Design Initial Software Technologies	Daniel	9/23/2019	10/21/2019	Done
10	Research Touch Detection	Erica	9/30/2019	10/14/2019	Done
11	Research Microcontrollers, Cooling Systems, Timers, and Communication Protocols	Logan	10/7/2019	10/28/2019	Done
12	Research Programing Environment and Tools	Gabriel	10/14/2019	10/28/2019	Done
13	Describe Hardware Prototyping and Testing	Erica	10/14/2019	10/28/2019	Done
14	Create Initial Software Testing Plan	Daniel	10/21/2019	10/28/2019	Done
15	100 Page Draft	Group	11/1/2019	11/15/2019	Done
16	Realistic Design Constraints	Gabriel	11/1/2019	11/12/2019	Done
17	Initial Prototyping and Ordering of Hardware Components	Erica/ Logan	11/1/2019	11/12/2019	Done
18	Expanding on Software Design and Testing Procedures	Daniel	11/1/2019	11/12/2019	Done
19	Initial User Interface Design	Erica	11/7/2019	11/12/2019	Done
20	Final Document Due	Group	11/15/2019	12/4/2019	Ongoing
21	Testing of Hardware Components	Logan/ Erica	11/15/2019	11/22/2019	Done
22	PCB Design Finalization	Logan	11/15/2019	11/28/2019	Done
23	External Software Library Research	Daniel/ Gabriel	11/15/2019	11/28/2019	Done
24	Physical Table Design and Prototype	Erica	11/15/2019	11/28/2019	Done
25	Expansion on Software Design	Daniel	11/15/2019	11/28/2019	Done

Table 54: Senior Design 2 Milestones

Number	Task	Member	Start	End	Status
1	Build and Test Hardware Subsystems	Logan/ Erica	12/5/2019	1/10/2020	Pending
2	Initial Software Writing and Debugging for Windows and Android App	Daniel/ Gabriel	12/5/2019	1/10/2020	Pending
3	Build Physical Table	Group	12/5/2019	1/10/2020	Pending
4	Final Report	Group	1/10/2020	TBA	Pending
5	Combine Subsystems Into Initial Prototype	Group	1/10/2020	1/20/2020	Pending
6	Testing and Subsequent Redesign of Parts if Necessary	Group	1/20/2020	3/16/2020	Pending
7	Finalize Prototype	Group	3/16/2020	3/23/2020	Pending
8	Peer Presentation	Group	TBA	TBA	Pending
9	Final Presentation	Group	TBA	TBA	Pending

Appendices

Appendix A: Copyright Permissions

Game Logo and Backgrounds

Modified from <https://pixabay.com/illustrations/wizard-magic-magician-mystery-1454385/> and other pages on <https://pixabay.com/> in accordance with [Pixabay License](#).

Images and Videos on Pixabay are made available under the Pixabay License on the following terms. Under the Pixabay License you are granted an irrevocable, worldwide, non-exclusive and royalty free right to use, download, copy, modify or adapt the Images and Videos for commercial or non-commercial purposes. Attribution of the photographer or Pixabay is not required but is always appreciated.

The Pixabay License does **not** allow:

- a. sale or distribution of Images or Videos as digital stock photos or as digital wallpapers;
- b. sale or distribution of Images or Videos e.g. as a posters, digital prints or physical products, without adding any additional elements or otherwise adding value;
- c. depiction of identifiable persons in an offensive, pornographic, obscene, immoral, defamatory or libelous way; or
- d. any suggestion that there is an endorsement of products and services by depicted persons, brands, and organisations, unless permission was granted.

Please be aware that while all Images and Videos on Pixabay are free to use for commercial and non-commercial purposes, depicted items in the Images or Videos, such as identifiable people, logos, brands, etc. may be subject to additional copyrights, property rights, privacy rights, trademarks etc. and may require the consent of a third party or the license of these rights - particularly for commercial applications. Pixabay does not represent or warrant that such consents or licenses have been obtained, and expressly disclaims any liability in this respect.

Figure 19

https://commons.wikimedia.org/wiki/File:UART_XBee_Data_format.jpg

This file is licensed under the [Creative Commons Attribution-Share Alike 3.0 Unported](#) license.

You are free:

- **to share** – to copy, distribute and transmit the work
- **to remix** – to adapt the work

Under the following conditions:

- **attribution** – You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **share alike** – If you remix, transform, or build upon the material, you must distribute your contributions under the [same or compatible license](#) as the original.

Figure 20

<https://commons.wikimedia.org/wiki/File:I2C.svg>

This file is licensed under the [Creative Commons Attribution-Share Alike 3.0 Unported](#) license.

You are free:

- **to share** – to copy, distribute and transmit the work

- **to remix** – to adapt the work

Under the following conditions:

- **attribution** – You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **share alike** – If you remix, transform, or build upon the material, you must distribute your contributions under the [same or compatible license](#) as the original.

This licensing tag was added to this file as part of the GFDL [licensing update](#).

Figure 21

https://commons.wikimedia.org/wiki/File:SPI_three_slaves_daisy_chained.svg

This file is licensed under the [Creative Commons Attribution-Share Alike 3.0 Unported](#) license.

You are free:

- **to share** – to copy, distribute and transmit the work
- **to remix** – to adapt the work

Under the following conditions:

- **attribution** – You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **share alike** – If you remix, transform, or build upon the material, you must distribute your contributions under the [same or compatible license](#) as the original.

This licensing tag was added to this file as part of the GFDL [licensing update](#).

Figure 22

https://commons.wikimedia.org/wiki/File:Bluetooth_network_topology.png

This file is licensed under the [Creative Commons Attribution-Share Alike 2.5 Spain](#) license.

You are free:

- **to share** – to copy, distribute and transmit the work
- **to remix** – to adapt the work

Under the following conditions:

- **attribution** – You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **share alike** – If you remix, transform, or build upon the material, you must distribute your contributions under the [same or compatible license](#) as the original.

Figure 23

https://commons.wikimedia.org/wiki/File:Wlan_www.png

This file is licensed under the [Creative Commons Attribution-Share Alike 4.0 International](#) license.

You are free:

- **to share** – to copy, distribute and transmit the work
- **to remix** – to adapt the work

Under the following conditions:

- **attribution** – You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **share alike** – If you remix, transform, or build upon the material, you must distribute your contributions under the [same or compatible license](#) as the original.

Figure 30

https://commons.wikimedia.org/wiki/File:World_Wide_Smartphone_Sales.png

This file is licensed under the [Creative Commons Attribution-Share Alike 3.0 Unported](#) license.

You are free:

- **to share** – to copy, distribute and transmit the work
- **to remix** – to adapt the work

Under the following conditions:

- **attribution** – You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **share alike** – If you remix, transform, or build upon the material, you must distribute your contributions under the [same or compatible license](#) as the original.

Figure 31

<https://developer.android.com/about/dashboards>

For the purposes of licensing, the content of this web site is divided into two categories:

- Documentation content, including both static documentation and content extracted from source code modules, as well as sample code, and
- All other site content

Unless otherwise noted, the documentation on this site, including any code shown in it, is made available to you under the [Apache 2.0 license](#), the preferred license for all parts of the of the Android Open Source Project.

Apache 2.0 is a commercial and open-source-friendly software license. The majority of the Android platform and documentation is licensed under the Apache 2.0 license. While the project strives to adhere to the preferred license, there may be exceptions, such as for documentation (code comments) extracted from a source code module that is licensed under GPLv2 or other license. In those cases, the license covering the source code module will apply to the documentation extracted from it. Source code modules that are used in the generation of documentation and have licenses that require attribution can be found in the [Documentation Licences section](#) below.

Third-party components of this site such as JavaScript libraries are included in the Android Open Source Project under the licenses specified by their authors. For information about these licenses, refer to the source files in the Android Open Source Project.

All other content on this site, except the license documents themselves and as otherwise noted, is licensed under the [Creative Commons Attribution 2.5](#) license.

You may use the content of this site in any way that is consistent with the specific license that applies to the content, as described above. For content licensed under Creative Commons Attribution 2.5, we ask that you give proper [attribution](#).

Figure 32

<https://developer.android.com/studio/write/layout-editor>

For the purposes of licensing, the content of this web site is divided into two categories:

- Documentation content, including both static documentation and content extracted from source code modules, as well as sample code, and
- All other site content

Unless otherwise noted, the documentation on this site, including any code shown in it, is made available to you under the [Apache 2.0 license](#), the preferred license for all parts of the of the Android Open Source Project.

Apache 2.0 is a commercial and open-source-friendly software license. The majority of the Android platform and documentation is licensed under the Apache 2.0 license. While the project strives to adhere to the preferred license, there may be exceptions, such as for documentation (code comments) extracted from a source code module that is licensed under GPLv2 or other license. In those cases, the license covering the source code module will apply to the documentation extracted from it. Source code modules that are used in the generation of documentation and have licenses that require attribution can be found in the [Documentation Licences section](#) below.

Third-party components of this site such as JavaScript libraries are included in the Android Open Source Project under the licenses specified by their authors. For information about these licenses, refer to the source files in the Android Open Source Project.

All other content on this site, except the license documents themselves and as otherwise noted, is licensed under the [Creative Commons Attribution 2.5](#) license.

You may use the content of this site in any way that is consistent with the specific license that applies to the content, as described above. For content licensed under Creative Commons Attribution 2.5, we ask that you give proper [attribution](#).

Works Cited

- [1] "Power Supply Standards, Agencies and Marks | CUI Inc," 2018. [Online]. Available: <https://www.cui.com/catalog/resource/power-supply-safety-standards-agencies-and-marks.pdf>. [Accessed 31 October 2019].
- [2] IEEE SA, "IEEE 802.11-2016 - IEEE Standard for Information technology-- Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PH," 2016.
- [3] "NASA Technical Standard: Soldered Electrical Connections," December 1997. [Online]. Available: <https://nepp.nasa.gov/docuploads/06AA01BA-FC7E-4094-AE829CE371A7B05D/NASA-STD-8739.3.pdf>. [Accessed 31 October 2019].
- [4] "Developer Policy Center," [Online]. Available: <https://play.google.com/about/developer-content-policy/>. [Accessed 31 October 2019].
- [5] "Core App Quality - Android Developers," Android Developers, [Online]. Available: <https://developer.android.com/docs/quality-guidelines/core-app-quality?hl=en>. [Accessed 31 October 2019].
- [6] "Google C++ Style Guide," [Online]. Available: <https://google.github.io/styleguide/cppguide.html>. [Accessed 31 October 2019].
- [7] "Wi-Fi Direct | Wi-Fi Alliance," Wi-Fi Alliance, 2010. [Online]. Available: <https://www.wi-fi.org/wi-fi-direct>. [Accessed 31 October 2019].
- [8] W. A. Initiative, "Web Content Accessibility Guidelines (WCAG) Overview," 22 June 2018. [Online]. Available: <https://www.w3.org/WAI/standards-guidelines/wcag/>. [Accessed 7 November 2019].
- [9] I. (. Wizards of the Coast, "media.wizards.com," 2000. [Online]. Available: https://media.wizards.com/2016/downloads/DND/SRD-OGL_V5.1.pdf. [Accessed 7 November 2019].
- [10] C. Herod, N. Boucher and R. Timones, "UCF Senior Design Group IV: MTPT," 2009. [Online]. Available: <http://www.eecs.ucf.edu/seniordesign/su2009fa2009/g04/>. [Accessed 31 October 2019].
- [11] C. Rodrigue, P. Murphy, J. Lundstrom and R. Mulvaney, "Smart Table - UCF ECE Senior Design Group 34," 2017. [Online]. Available: <http://www.eecs.ucf.edu/seniordesign/fa2016sp2017/g34/index.html>. [Accessed 31 October 2019].

- [12] J. Perardel, "Magic Frame : Turn Everything into a Touch Area," 5 September 2017. [Online]. Available: <https://hackaday.io/project/27155-magic-frame-turn-everything-into-a-touch-area>. [Accessed 31 October 2019].
- [13] "NUI Group Community Wiki - Diffused Illumination (DI)," NUI Group, 6 December 2009. [Online]. Available: http://wiki.nuigroup.com/Diffused_Illumination. [Accessed 31 Oct 2019].
- [14] "NUI Group Community Wiki - Diffused Surface Illumination," NUI Group, 5 January 2009. [Online]. Available: http://wiki.nuigroup.com/Diffused_Surface_Illumination. [Accessed 31 October 2019].
- [15] *How to Control a Ton of RGB LEDs with Arduino & TLC5940 - YouTube*.
- [16] *TLC5940 16-Channel LED Driver w/EEPROM DOT Correction & Grayscale PWM Control / TI.com*.
- [17] *Difference Between Linear Regulator and Switching Regulator | Electronics Basics | ROHM*.
- [18] "Community Core Vision 1.5," NUI Group, [Online]. Available: <http://ccv.nuigroup.com/#about>. [Accessed 31 October 2019].
- [19] L. Bender, "Scene 1.0 - Background subtraction and object tracking with TUIO," [Online]. Available: <http://scene.sourceforge.net>. [Accessed 31 October 2019].
- [20] "TouchLib | A Multi-Touch Development Kit," NUI Group, [Online]. Available: <http://nuigroup.com/touchlib/>. [Accessed 31 October 2019].
- [21] "TUIO Protocol Specification 1.1," [Online]. Available: <https://www.tuio.org/?specification>. [Accessed 31 October 2019].
- [22] "App Store - Support - Apple Developer," Apple Inc., 2019. [Online]. Available: <https://developer.apple.com/support/app-store/>. [Accessed 31 October 2019].
- [23] The QT Company, "QT | Cross-platform software development for embedded and desktop," The QT Company, 2019. [Online]. Available: <https://www.qt.io/>. [Accessed 11 October 2019].
- [24] "wxWidgets: Cross-Platform GUI Library," wxWidgets, 2019. [Online]. Available: <https://www.wxwidgets.org/>. [Accessed 11 October 2019].
- [25] "Build a UI with Layout Editor | Android Developers," Android Developers, [Online]. Available: <https://developer.android.com/studio/write/layout-editor>. [Accessed 31 October 2019].
- [26] "OpenCV," [Online]. Available: <https://opencv.org>. [Accessed 11 November 2019].

- [27] "Java Swing Tutorial," JavaTpoint, 2018. [Online]. Available: <https://www.javatpoint.com/java-swing>. [Accessed 31 October 2019].
- [28] [Online]. Available: <https://stackoverflow.com/>.
- [29] "Wi-Fi Direct | Android Open Source Project," Android Developers, [Online]. Available: <https://developer.android.com/guide/topics/connectivity/wifip2p>. [Accessed 31 October 2019].
- [30] "Multipeer Connectivity | Apple Developer Documentation," Apple Inc., 2019. [Online]. Available: <https://developer.apple.com/documentation/multipeerconnectivity>. [Accessed 31 October 2019].