Gaming Wizard: A Smart Table for Tabletop Gaming

Gabriel Holguin, Daniel Kalley, Erica Lindbeck, Logan Taylor

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — Our project seeks to address the needs of the modern-day tabletop gamer. The Gaming Wizard provides a digital system in the form of an optical touchscreen table and an Android app, which emulate the traditional gaming elements of a battle map and sheets of character information. This system enables users to efficiently set up and play games as part of a standard gaming group and improves gameplay beyond the traditional mediums of play with features such as sound and lighting effects. This project incorporates significant phone application, computer software, and microcontroller programming elements, in addition to numerous core and peripheral hardware systems.**

***Index Terms* — Bluetooth, image processing, infrared imaging, microcontrollers, multithreading.**

I. Introduction

In a 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 map 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 recognizes this and asks for a confirmation. The application saves and remembers character locations even after being closed, which greatly reduces the time it takes to reset a board between meetings.

Additionally, the application tracks character information such as attributes and combat statistics. This creates a paperless game that removes the need for constant erasing and rewriting. Combining this with the location information also greatly reduces 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. Dice rolls can 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 hosts the game from a user's laptop and communicates with players' phones using Bluetooth connections. From their phones, players and game masters (GMs) 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 are 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 are saved to the GM's phone or computer when the table is shut down, allowing the group to recreate the game board quickly in subsequent game sessions. The GM has access to additional options such as map selection and enemy creation that are not accessible to players. The current map, along with additional relevant graphical information are projected onto the table's surface from the interior of the table.

II. Hardware

A. Multi-Touch Surface Scheme

The top of the smart table consists of a multi-touch surface which tracks multiple (primarily stationary) game pieces, in addition to processing touch inputs by users. To achieve this functionality, we implemented an optical

scheme in the form of rear diffused illumination (rear DI). The general methods of this scheme are detailed in Fig. 1 below.

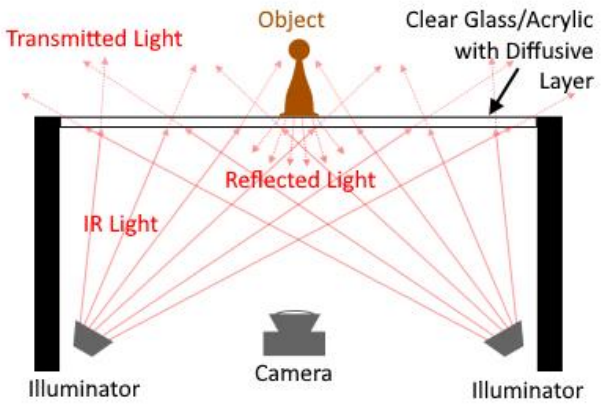


Fig. 1. Generalized rear diffused illumination setup

The surface itself consists of a sheet of clear glass or acrylic, with a diffusive material layered either above or below for image projection and light diffusion. The visual display is generated by a projector underneath the surface (not pictured). Additionally, infrared light is projected upwards and is partially transmitted through the surface. When an object (or user's finger) is placed on the surface, all the infrared light that would be otherwise transmitted through the area covered area is reflected downwards. This creates a bright spot known as a "blob" in the video feed of an infrared-sensitive camera pointed at the surface. Tracking software can then process the blob-filled image to create a map of touch input and object locations.

B. Surface Materials

For our system, we used a clear cast acrylic and a sheet of 0.003" drafting film to make the table surface. The clear cast acrylic provided a stronger structure than glass while remaining lightweight and completely transparent. Drafting film was selected based on effective use in similar hobby projects, and after testing with multiple thicknesses the 0.003" sheet was selected to provide sufficiently high contrast for blob detection.

C. Illumination

To illuminate the screen, we first tested several LEDs within the 850-940 nm spectrum, in several configurations. These configurations were evaluated to determine which arrangement would provide the most even illumination, reducing the chances of hot spots causing false inputs. Upon installing the rest of the multitouch hardware, however, a large array was deemed impractical. Thus, we decided to install infrared illuminators designed for use with night-vision security cameras on opposite ends of the table's interior, with smaller sheets of diffusive material placed across them to improve the evenness of the resulting illumination. These illuminators were selected for their small footprint, included power supplies, and relatively even baseline illumination.

D. Infrared Camera

To record the blobs and thus detect touch inputs to the table, a camera which viewed only the infrared spectrum was required. The PlayStation Eye camera was selected based on price and because its use in previous projects meant there was 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, so it met our specifications nicely. The infrared-light-blocking filter was removed and then replaced by visible-light-blocking material. Floppy disk material was chosen as the visible-light-blocking filter as we had some donated freely to the project and it worked well in testing.

E. Projector

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. Despite the increased cost, a short throw projector was preferred 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, so a used projector was acquired.

A used BenQ MX810ST projector was selected, with a native resolution of 1024 x 768 pixels, a throw ratio of 0.6, and both auto and manual keystoning options up to 30°. This projector also came equipped with an HDMI adapter and a remote, allowing us to easily make adjustments and send an image signal once the projector had been mounted inside the table.

F. Table

The physical structure supporting the rest of the hardware had 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. Few likely candidates for modification could be identified, so eventually a from-scratch design was developed. Due to the lack of experience among the group, this design is simplistic but functional.

This design consists of a 30” x 38” x 36” wooden box, with a 24” x 32” touch surface resting in grooves at the top of the table. In each corner of the table a 2” x 2” leg supports the structure and provides attachment points for the plywood sides. During construction, additional 1” x 4” bracing was placed between the legs at the bottom and top of the table, to provide support and prevent the structure from skewing and collapsing. One 38” side remains open to allow easy access to the interior for component installation and adjustment. The opening was initially intended to be covered by a hinged plywood door. However, in practice a door of this size was unwieldly and inconvenient and was therefore replaced by a light-blocking curtain on a rod placed just inside the opening. Although cosmetic measures such as painting or otherwise decorating the table were considered, they were eventually discarded due to time constraints.

G. Microcontroller

This project uses a custom printed circuit board (PCB) containing an ATmega2560. The ATmega2560 is used for reading a temperature sensor, controlling fans, and prompting LED effects. 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 ATmega2560 was selected for achieving 16 MIPS at 20 MHz, holding 256 KB in flash memory, providing 86 I/O pins, and allowing us to test programs using the Arduino Mega development board. In the final system, a module based on FTDIs FT232RL chip was used to implement USB-to-serial conversion, in order to communicate with the microcontroller once the bootloader had been burned.

Once it had been determined how the MCU would be programmed and how the PC would communicate with the MCU a standalone MCU schematic could be created. This schematic 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

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 is dedicated to uploading programs and the second one is used for receiving UART commands during gameplay. Decoupling capacitors are required to reduce noise from the chip and traces. The digital pin header was originally intended for the 7-segment LED timer and the analog pin header is used for any ADC that may be required. Finally, an LED is connected from VCC to GND and acts as a power indicator.

H. 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 are enclosed within the table. To reduce this heat two fans are be used, one for intake and one for outtake. The fans are attached to opposite sides of the table so air will flow through the table and cool the enclosure. The chosen fans operate at 12 V with a current draw of 0.25A and run at a speed of 1600 rpm. Each has an estimated air flow of 73CFM which is acceptable for this size of enclosure. The size of each fan is 120x120x25mm (4.72x4.72x4.13in) which easily fits within the side of the table.

Controlling the fans so they only run while the enclosure is above a certain temperature gets rid of unnecessary power usage. A thermistor is used to estimate the temperature inside the table using a voltage divider circuit as an analog input to the MCU. 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. From the thermistor datasheet we know that the resistance of the thermistor at 30 degrees Celsius is 8k. A 10k resistor is used in the voltage divider for the second resistor, and 5V is used for voltage source, so 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, 569. As a MOSFET was used to supply current to the fans, the bit controlling the

digital pin going to the gate of the MOSFET was configured so that it would be set high if the value of the analog read function became greater than or equal to 570.

I. Timer

A timer was added to the table for use in timed skill challenges and limiting turn durations. A quad seven-segment display is used to display the time, and four buttons provide necessary inputs to increment the time by minutes and second, start and stop the time, and turn the timer on and off. 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: Off; On, Setting Time; On, Counting Down; and On, Timing Complete. A prototype timer was created 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 to be adapted and programmed into the microcontroller. However, the COVID-19 crisis introduced some severe delays in assembling peripherals to our system, so the Raspberry Pi system was used in the final prototype.

J. Special Effect Lighting and Sounds

Lighting effects are used for player actions such as attacking and casting spells. 20 RGB LEDs line the edges of the table and are controlled by the MCU. When an action is done on the player's phone a signal is first sent through Bluetooth to the single board computer. The computer then sends a signal to the MCU that selects which function the MCU will perform and a preprogrammed LED effect will occur. A TLC 5940 LED driver is used to control the effects. This IC reduces 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. Sound effects complement the LED effects, but speakers are controlled by software on a PC rather than the MCU due to memory limitations. These speakers are connected to the PC's 3.5mm jack.

K. Power Supply

Many elements of the table are powered by a standard surge protector, as they came with their own power supply and are very expensive, such as the projector and a host PC. However, the microcontroller, LED drivers, and fans require separate power supplies. For safety purposes, an AC adapter was purchased to match the highest DC requirement of these components, 12 V. A power supply with a maximum current draw of 2 A was selected based on usage estimates for our individual components. A TPS565201 IC was used to construct a step-down regulator was added to provide 5 V for the components requiring a lower input voltage.

III. SOFTWARE

The tabletop game software is comprised of three different subsystems. These subsystems are the object detection, game software, and the android app. The game and object detection subsystems are both located on the computer system. The different subsystems run on different threads to allow for simultaneous execution. Specifically, this enables the object detection subsystem to generate locations while the game software handles requests from the mobile app subsystem. The game subsystem receives player piece location data from the object detection subsystem, and it receives explicit user commands through the connection with the app. The game software must 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 faces the bottom of the display and looks for blobs that contrast with background illumination to distinguish objects. The user app is the endpoint where users can directly communicate with the software. The app acts as a replacement for the character sheet which is traditionally a physical piece of paper storing all the character information. Communication between the app and computer systems is carried out using the Bluetooth protocol.

A. Object and Touch Detection

Fig. 2 displays the use case for the object detection subsystem. The key feature for this subsystem is the ability to track user pieces on a display. This can be accomplished by using third-party open source object detection framework. The framework can track cursors and blobs. For each tracking object the framework must be able to transmit the data readings. The input to this subsystem is a camera feed and the output is the object detector class. This class is 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.

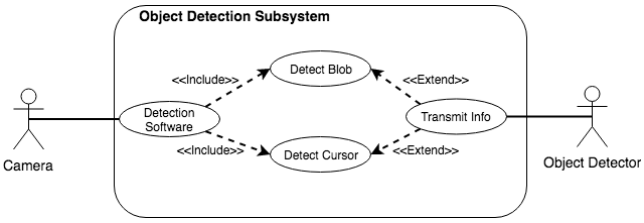


Fig. 2. Object detection use case diagram

The CCV (Community Core Vision) [1] framework was developed by a group called The Natural User Interface Group. The current version was released in November of 2014. Ideally, the software locates white blobs on a black background in a video input stream with no background noise. This type of video stream can be closely matched with our rear DI scheme.

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

The TUIO (Tangible User Interface Objects) [2] protocol is an open source framework, free under the minimally restrictive L-GPL license. The CCV software utilizes this framework to package and transmit information about detected objects. This format provides dimensional values that describe an ellipse that is created within the bounding box of the blob. The bounding box allows 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 provided with respect to the top left corner of the image. After the bounding box is created, an ellipse is inscribed, and the area of this ellipse approximates the area of the blob. The bounding box is oriented based on the direction that the object is placed on the display. An angle is associated with this bounding box. The (x,y) coordinates for the center point of the blob as well as the ellipse dimensions are normalized and returned as floating-point values. The (x, y) coordinates are used to determine what grid space the center of the blob is located inThe high-level flow of communication can be seen in Figure 3. The transfer of blobs begins with the video stream input and ends with the GetLocation() function implemented in the ObjectDetector class.

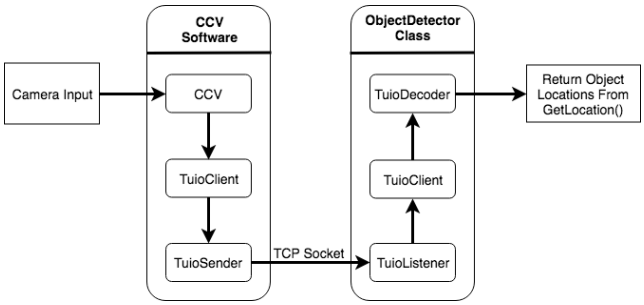


Fig. 3. CCV and ObjectDetector Communication

B. Mobile App

The mobile app software is the input endpoint point for a player's gameplay decisions. The decisions are collected through a series of menus that are displayed depending on the current state of the game. While at the main menu, a device only has an instance of the basic app user interface. The methods and fields can be split into three different classes. The parent class is the overall app user interface with two subclasses for the GM and the player. Regardless of the type, namely being a player or GM, each user can interact with the main menu. Only after the game has been started do the menus differ. At that point, the different subclasses are instantiated. The differences in the menus is accompanied with difference in functionality. The Game master has 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 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. Some functions are 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. The app is styled to be reminiscent of the paper and pencil format of the original game.

We decided to develop our mobile app for Android OS for several reasons. Emulation in Android Studio allowed for extensive unit testing during development. Further, Android OS sales currently outpace iOS. Finally, the lack

of a MAC and the high annual developer fee for Apple iOS would have made iOS development costly.

Because the app was built in C++, a third-party open source library was required for the interface. Although the project only developed 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. QT was selected as it is free to use and provides cross platform support.

C. Game Software Overview

The game software can be broken up into different classes that encapsulate the different aspects of the program. Those classes are the player class, game class, Bluetooth class, and display class.

D. Player Class

The Player class oversees managing player actions. These player actions come with values associated with either hit points or range of action. When performing an attack action, the attack range is based on player stats and so is the hit point damage applied during that attack. Similarly, when performing a move action, the player has a set range based on their player stats to move. Players have the option to perform both actions on any given turn assuming NPCs are present on the board. If the game state is not currently in combat, then the attack action is not an option. Players also have the choice of just performing a single action. Since there are a few options, the player needs to select what action they would like to perform through the mobile app. The Game class must then to ask for the distance in order to display the range and continue gameplay. NPCs are controlled by the GM through the mobile app. For all purposes they act as a normal Player. A subclass is not required as no different functionality is required.

E. Game Class

The Game class is the starting point for the program. When the program is launched, an instance of the Game class is instantiated, and the game setup begins. The key steps to setup are to connect users to the game, display the map, and either restore player locations or begin tracking player locations. The constructor for this class is used for setup. The class is also in charge of validating player actions on the display and processing any input from a player's mobile app. Since the game class maintains the major functionality of the game, there are private field elements for each of the objects required for player and display interactions. The players and NPCs are also maintained in this class. A vector for each is the data structure to hold the values. Both are maintained as instances of the Player class.

When the game constructor is called, it checks for any instance of the object already existing. If the object does not exist, it creates the object and stores it in the private field. Setup creates an instance of the Bluetooth, Display, Player, and ObjectDetector class. Once all of the necessary fields are stored, the gameplay method is called so that the game can begin. The gameplay method call indicates the end of the constructor and finalizes the connection of all players for the game.

The gameplay method is the main functionality of the game. It maintains ordered player turns and processes all user input. At this point, the game has an instance and all players are connected. The turn order is decided by the GM and maintained in this class. In a round, each player takes a turn. The parts of the turn that have a digital action are movement and attacks, which call a Display class method to highlight spaces. The game then waits for the user to choose a physical space or move their player piece on the display. The location of the action is collected by the object detector class. The location then passes through the validate movement method. If the response of the method is true, then the action is valid, and the turn continues. If the method returns false, the position is invalid, and the game requests 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 is asked again what action they are going to take, and the process repeats.

The game class also handles the save and load methods. Both methods are initiated by a request from the GM's mobile app. The Game class handles such requests. For a save request, the game first checks to see if the mobile device has a past save data file. If the file exists, it is overwritten, otherwise it is created. The different object fields are iterated through and their contents are recorded into the save file. This save file is then transmitted to the game master's device for storage. Each of the objects is deleted to free memory. Finally, all devices are disconnected to setup for the next game or shut down.

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

Create game acts very similar to load game. Besides the obvious difference of no save file, create game initiates

the start of the game, receiving the number of players playing and the map to be displayed from the game leader. Character sheet with all character statistics are created and pieces are placed in a starting location. While all these steps are occurring, the Game call receives and populates class elements with information about each player to begin tracking gameplay.

F. Display Class

The Display class holds the methods required to do all image processing required for gameplay. A third-party image processing library is used to do the image overlaying portion of the image processing. This third-party library is OpenCV [3]. Specifically, the methods to read in, write out, show, and add images together are used. These four methods provide the necessary manipulations required to update and display images for the users. The addWeighted() method is used to merge two images with a different opacity. The base image has the higher opacity while the image being overlaid on top has a lower opacity creating a merging effect.

Privately maintained fields enable the class to manage the different layers of images and provide the ability to determine their locations. The map and grid line images are maintained to avoid corrupting the original files. The PPI is maintained so that when dynamically creating images, the scale is correct. A list of NPC and distance images is maintained. This is referenced when creating images.

The map image that is projected on to the touch display is provided by the user through the mobile app. To not destroy the original map image, secondary images are overlaid on top. The grid that is overlaid onto the map image is a separate image created with all parts transparent except for the gridlines. To accomplish proper scaling, the PPI found above is the exact dimension in pixels needed for the 1" by 1" grid spaces. The first and last two rows of pixels are used to create the horizontal grid lines. Likewise, the first and last two columns of pixels are used to create the vertical gird lines. When overlaying smaller images onto the map, the region that the image needs to occupy must be a value passed to the overlaying function. The determine location method takes in a grid space location and creates a rectangle object that holds the pixel locations on the display.

The actual overlaying is done by generating a new image each time there is a change. This is done using library methods that copy an image on to another image. The general flow is 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 Fig. 4.

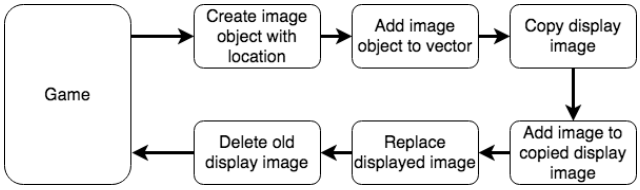


Fig. 4. Image Addition Flow

This process occurs in such a way that the layers are presented in the correct order. The gridlines are added on top of the map image first, followed by any distance indicator spaces and lastly any NPCs that are currently on the board. This avoids any NPCs accidently hiding under any other image. They are considered players and their locations should always be known and visible. To create a more natural look, the distance indicators are not solid spaces of a single color. Instead, the distance indicator image is blended with the map to give the map a highlighted affect.

To make removing images more straightforward, 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 needs to be passed in. Identifying information includes the image's type and its location on the display. From this information, the image object is found and freed and a new display image is generated and updated.

G. Bluetooth Class

The Bluetooth class maintains information about all connected devices. The connections 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. Two lists of character arrays are used to hold the data being sent and received. Whenever there needs to be a message sent from the game to a specific player, the ID will be used to fill that players send buffer. Likewise, whenever the game receives a message from a specific player, their receive buffer is filled.

The game software acts as the server for all the Bluetooth communications. Specifically, the Winsock2 library was used to create the server. The game controls communication with two different procedures, a server and client procedure. The server procedure handles accepting Bluetooth connections and disconnecting Bluetooth connections. The client procedure handles read, write and close actions. The messages sent between devices follow the JSON format.

H. Multithreading

Since we believed that the execution of our game would be taxing on our CPU (Central Processing Unit), we explored the option of multithread programming. Microsoft Visual Studio has an external add-in library called MFC (Microsoft Foundation Class), which allows C++ developers to use multithreading. We were able to use multithreading in our project by giving each thread that has to deal with object detection to one core, while the game itself could run on another. Although multithreading was useful in speeding up execution, debugging and testing did become harder as more cores were utilized.

VI. Conclusions

The Gaming Wizard system provides a suite of features which improve the tabletop gaming experience. The smart table preserved the most beloved elements of the traditional battle map, namely the use of user-chosen landscapes and character figurines, while improving efficiency by saving character locations and automatically calculating distances. The smartphone application makes saving and looking up character details easy and allows the system to automatically calculate the results of any action by simulating dice rolls with appropriate modifiers. Additional features that would be difficult to implement independently of such a comprehensive system increase immersion and improve the gameplay beyond what traditional media could accomplish.

Acknowledgements

The authors wish to acknowledge the assistance and support of our faculty advisor Dr. Samuel Richie in developing the scope of our project and maintaining our timeline. We would also like to thank Graeme Lindbeck for his woodworking knowledge and assistance in table construction. Finally, we thank the committee of professors who kindly agreed to review our work on this project.

References

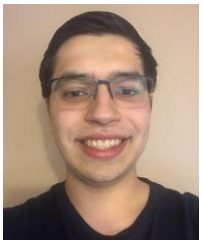
[1] "Community Core Vision 1.5," NUI Group, [Online]. Available: *http://ccv.nuigroup.com/#about*. [Accessed 31 October 2019].

[2] "TUIO Protocol Specification 1.1," [Online]. Available: *https://www.tuio.org/?specification.* [Accessed 31 October 2019].

[3] "OpenCV," [Online]. Available: *https://opencv.org*. [Accessed 11 November 2019].

Biography

Gabriel Holguin is a senior Computer Engineering major graduating in Summer 2020. He will be pursuing a career as a software developer in the industry after graduation.



Daniel Kalley is a senior Computer Engineering major graduating in Spring 2020. He will be pursuing a master's degree in computer engineering.

Erica Lindbeck is a senior Electrical Engineering and Mathematics double major graduating in Spring 2020. She will be pursuing a PhD in Electrical Engineering with a focus on signal processing at the University of Florida starting in Fall 2020.

Logan Taylor is a senior Electrical Engineering major graduating Spring 2020. He will be pursuing a career in the Orlando area after graduation.