Player Illuminated Negativity Killer

Kevin Cochran, Zachery Dunlop, Taylor Grubbs, Noah Thering

University of Central Florida School of Electrical Engineering and Computer Science, Orlando, FL

***Abstract* - The Player Illuminated Negativity Killer is a physical board game designed to provide enjoyment to players of all ages. The project achieves this on a single PCB which hosts 3 microcontrollers, a power regulation system, and an adjustable speaker setup. It's controlled via a custom built software architecture capable of supporting an arbitrary number of games, and 8 games have been prepared for demonstration. Each game supports configuration options and clear rules, allowing new players to explore different games they may like with the peace of mind knowing they can't make an illegal move or be taken advantage of by their opponent.**

Index terms - Game, Game Board, LED, Microcontroller

I. INTRODUCTION

P.I.N.K. was conceived as a way to eliminate the frustrations usually associated with traditional board games, such as losing pieces or being uncertain of legal moves. The board game does not require external pieces to be played, and enforces a ruleset chosen at the start of each game. Eight games are supported by the board, and they are: Tile Flip, American Checkers, Chinese Checkers, Connect 4, Pop-out, Tic Tac Toe, Simon, and Go. Clear acrylic button assemblies hide an electric switch and LED; these components together comprise each tile of the board. The board has similar dimensions to a traditional chess board, being around 15 inches on each side. The board will have an internal battery and support charging over USB. The microcontroller is programmable over an exposed micro USB port to allow for easy addition of games.

The LCD has 2 display states called views, one for displaying game information such as time and score, and another for displaying a menu that can be used to tweak game settings or choose another game.

General operation consists of picking a game from the main menu using the LCD screen, selecting the desired settings for that game also with the LCD screen, and then pressing the LED buttons to play the game!

II. OVERVIEW OF SYSTEM COMPONENTS

A. Microcontrollers

Making the project as user friendly as possible was a major design goal. In order to accomplish this usb communication was necessary to allow the user to add new games to the board. The Atmel ATmega16u2 is used for usb to serial conversion. This allows us to upload code via a usb connection. USB is more convenient for the user to program the Atmel ATmega2560.

The choice of the main microcontroller was the most crucial decision of the project. The main microcontroller needed to have enough input and output to interface will all of the peripheral components. The Atmel ATmega2560 was chosen for this purpose. The Atmel ATmega2560 microcontroller handles the bulk of the code execution, such as Game logic and handling user input. Multiple interrupt pins were employed to properly interface with both the button matix and menu buttons, to ensure that each system would be usable simultaneously. The microcontroller uses 33 digital IO pins to interface with the leds, button matrix, and menu buttons. SPI communication is used for both the SD card and the LCD screen.

An ATTiny13a is used to control the PWM for the speaker, as to avoid interfering with the very sensitive timing of the LED strips while playing a sound and writing LEDs simultaneously. The addition of the ATTiny13a allows for a more pleasing tone generated as an LED write would cause the volume to fluctuate.

B. Button choice

Cherry MX keyboard switches were chosen for the project. The use of keyboard switches simplified the button tile design described later in this document.

C. Button Matrix

The button matrix circuit was needed to reduce the amount of digital data pins needed for the 81 buttons of the board game. Only 18 data pins and 1 interrupt pin were needed to interface the ATmega2560 microcontroller with the button matrix instead of the 81 interrupt pins that would be needed to wire up each button individually. The diode matrix required 171 diodes, 19 resistors, and 81 Cherry MX switches to connect all the buttons using a few data pins as possible.

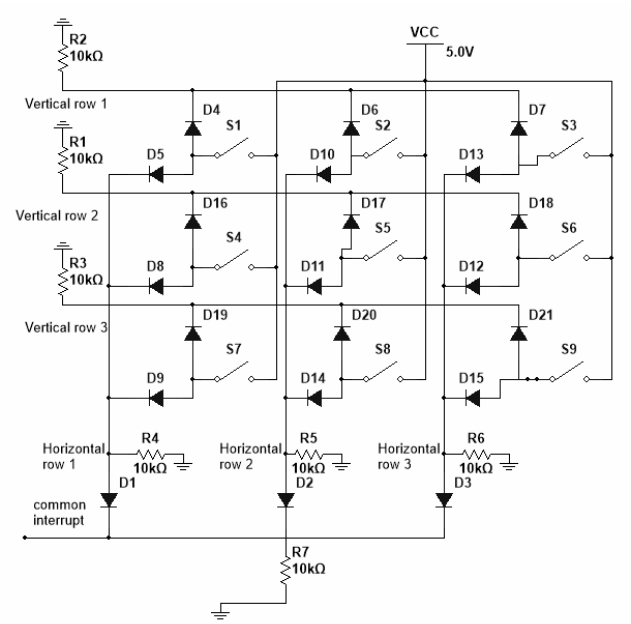


Fig. 1. A 3x3 example of the button matrix circuit

D. PCB Design Process

The PCB was designed in Kicad. The PCB had to contain all the necessary components for the project. One PCB was designed to house the microcontrollers, SD card slot, and the 2 integrated circuits needed for the voltage boosting and battery charging.

Each button had a pcb to support the Cherry MX buttons and the 2 diodes necessary for the matrix circuit. The diodes needed for the common interrupt pin on the button matrix circuit are wired between the button pcbs and the main MCU pcb. The button PCBs are then wired together according to the circuit shown above.

E. LCD

The LCD display is used to display a menu to select games, and a game info panel when the menu is not in use. Once game execution has started the menu can be used to change options pertaining to the active game.

F. Menu Buttons

To control the menu displayed on the LCD 6 buttons are used. Up, down and select are used to navigate the menu. There is a power button to activate the low power state of the board. The extra 1 and extra 2 buttons have different features for different games.

G. Board Buttons

The 9x9 button grid will comprise the board game. The tiles will function as the buttons. Pressing a tile down will activate the button underneath.

H. LED strips

The LED strips run under the tiles of the board. Each tile of the board game will be illuminated by 1 LED. The LEDs mounted under the tiles will signify the piece located at that tile. The RGB LEDs are color addressable. This allows different colors to be used to signify the pieces belong to different players. The LED will also be used to show possible moves in games such as checkers.

I. Speaker

The Piezo Buzzer will be used to warn the player that an illegal move has been attempted. The player will then be allowed to try again to make a legal move.

The above mentioned components help make player interaction with the board game simple and easy, helping eliminate sources of confusion and frustration. Giving proper feedback to the player was necessary for the project to achieve the goal of providing entertainment.

III. PHYSICAL BOARD DESIGN

The board design focuses on delivering tactile and visual feedback while allowing for a broad range of games to be displayed and supported. To this end we've created a 9x9 acrylic lattice, which supports 81 clear acrylic button assemblies that rest on top of 81 cherry MX switches. LED strips run under each button, supplying a single LED diode to each. This allows each button to light up in a variety of different colors while also quickly detecting and responding to input. The clear buttons are sanded to allow light to scatter as it diffuses through their surface, hiding the internal assembly and more fully illuminating the button.

A. Button Lattice Construction

The button lattice is the core of the input/output system of the board, and is responsible for holding the light-up button assembly in place. Made out of black acrylic, it consists of 3 layers that tightly snap together. Multiple revisions to the button lattice design were necessary to achieve the desired result.

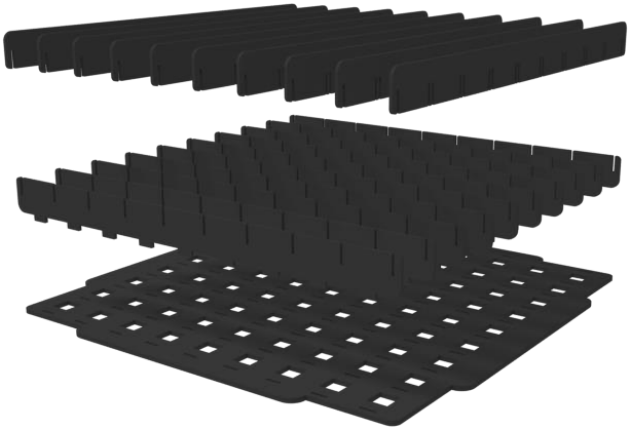


Fig. 2. The 3 main components of the button lattice

The first layer is the bottom sheet, which has holes both for the electric switches and the prongs of the second layer dividers. Once inserted through the holes, the electric switches are held in place by the small PCBs they are soldered to. These PCBs also allow the diodes required for the button detection circuit to be more easily managed, as they are soldered directly to dedicated pathways in said PCBs.

The second and third layers are a 10x10 series of interconnecting dividers of acrylic. The bottom dividers connect to the lower sheet with their bottom prongs, and connect seamlessly with the top set of dividers due to a matching set of cuts in each. The top set also sports slits for the LED strips to be run through, slightly narrower than the corresponding slits on the buttons themselves to ensure the LED strips don't get caught on a button as it's coming down.

B. Button Construction

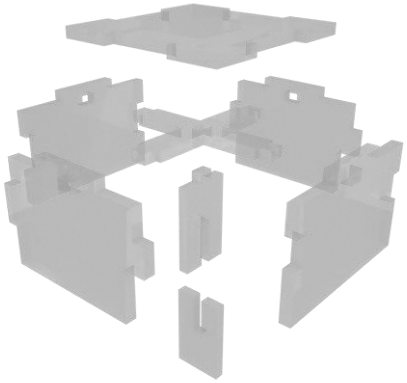


Fig. 3. Exploded view of the individual button assembly

Each button consists of 8 parts, each cut from clear acrylic. A cross piece serves as the center of the construct, and connects to each side as well as a cross pillar. This is both to aid in assembly and to create a more stable final product.

The cross pillar is used to make contact with the electric switch at the bottom of the slot, its plus shaped design being useful in both construction and in ensuring contact is made with the switch even in the event of wobble.

Each button side features prongs to aid it in interlocking with the other button sides, along with an extended cut in select sides to provide space for the LED strip to slide through when the button is pressed down. Every joint is cut with special features to reduce stress and help the pieces interlock. These are mainly pockets cut into inner corners, and small protrusions at the end of each socket.

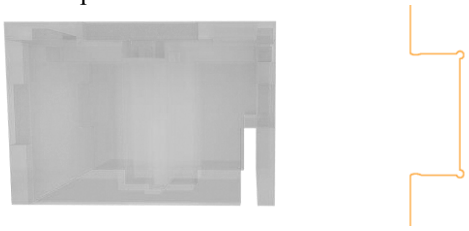


Fig. 4. Button side view with slot, and acrylic joint design.

Finally, the top piece is used to add extra stability to the assembly and of course provide a nice smooth surface for the players to press down on.

C. Exterior Case

The exterior case is what holds the entire button lattice, buttons, LCD Screen, and of course the PCB together in one place. It consists of a bottom and four sides with one side being tilted to house the LCD screen and the necessary buttons with the LCD screen. The top is open to allow the acrylic sheet to fit into the case and allow access to the buttons. It also provides protection for these components and an easy way to transport them.

IV. Electrical System Layout & Design

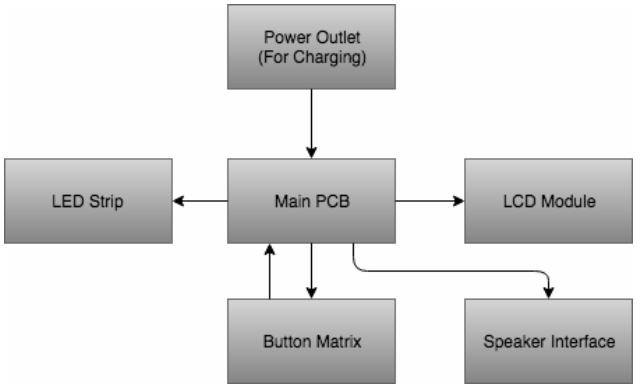


Fig. 5. Physical Layout Overview

A. LED Choice

For the LEDs of the gameboard, an Alitove 16.4ft WS2812B individually addressable LED strip was chosen. The LED strip has the following features [4]:

- RGB LEDs that can implement many multiple colors.
- The strip is able to be cut into strips of 9 LEDs to create 9 rows of 9 LEDs.
- Each LED strip has three connections: power, ground and data simplifying the wiring of the game board.

B. Portable Power Source

One of the key features of the LED game board is its portability. To meet this design requirement, the power source of the project has to be portable. While it would be possible to use several common batteries such as AA, C or D batteries, a lithium polymer battery was chosen for its large power capacity and its rechargeability. The design team chose a 6600mAh battery to meet the power requirements of the project. The 6600mAh battery will have ample capacity to meet the 2 hour battery life needed for this project.

C. Lithium Polymer Battery Charge Protection [2]

For charge protection, the MCP73871 was chosen. This IC uses a thermistor and resistors to monitor the temperature of the battery. The temperature of the battery is proportional to the voltage across the thermistor. The IC is then able to monitor that voltage, and compare it to a predetermined voltage range. If the voltage is outside of this range the IC will shut off and the battery will no longer be charged.

D. Lithium Polymer Battery Voltage Regulation [1]

The chosen lithium polymer battery has an output voltage of 3.7V. The chosen microcontroller requires a input of 5.0V. The TPS61090 switching regulator was chosen to boost the output voltage up to be 5.0V. in addition to boosting the voltage of the battery, the TPS61090 monitors the output voltage from the battery. Once the battery is drained to a predetermined voltage the IC will shut off and stop drawing power from the battery protecting it from draining to an unsafe voltage level that could damage the battery.

E. Logic level shifter

The 74HC4050DCT is the logic level shifter chosen for this project. It converts a 5V logic signal to the appropriate 3.3V needed for both the SD card and the LCD screen.

F. Menu Interface components

The choice for the menu interface is the 18-BIT TFT display with microSD card holder. This LCD screen has dimensions of 2.17” x 1.57” with 320 x 240 color pixels. With the SD card holder, it is possible to display more complex images and pattern on the LCD screen that require more memory to be stored.

V. SOFTWARE ARCHITECTURE

Consisting of over 5000 lines of custom C code, the core software architecture is responsible for linking the physical input/output of the board to the software controlling the desired game behaviors. Ideally, this is to be done efficiently and in such a way that the game creator need not worry about the implementation details of the board itself outside of certain specifications like how many tiles it has. This modular approach allows easy and efficient game creation, which is essential for supporting a large number of games.

A. General system construction

The software is divided into 3 separate layers: the Communication Layer, the Management Layer, and the Game Layer. Each layer serves a different purpose and only communicates with the layers directly above and below it. This modular approach allows a layer to be substituted out if need be.

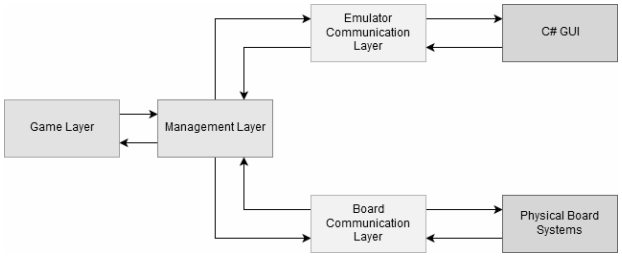


Fig. 6. Main Layers of the Software Architecture

The Communication Layer is responsible for direct communication with the board's physical systems; delivering input in an understood format to the Management Layer and translating the management layer's instructions to physical board output. It also provides an interface layer for basic board systems such as random number generation, as these have unique implementations depending on the platform.

The Management Layer directs the program flow and provides modules that may be of use to the game layer, such as timers and LED control management. It is also home to the event queue, which is responsible for directing which tasks to execute and to ensure that interrupts are separated from the rest of the code base.

The Game Layer is where the actual game code resides, and its behavior is directed through Game Functions and Callbacks. Game Functions are functions the game can call to interact with the board and LCD screen, while Callbacks are functions within the game that get called in response to different events. The game's program flow is entirely directed through these callbacks, which range from OnGameLoaded to OnLCDTimerHitZero. This allows a game to easily respond to different events without worrying about what happens in the meantime.

B. Event Queue

P.I.N.K. heavily relies on interrupts to direct its internal operation, and interrupts come with a few potential downsides, the main one being the possibility for the main execution thread to be interrupted at any time. This introduces the need for unoptimized volatile variables and the ever present consideration of "what will I do if my code is interrupted at this exact moment?". To limit these concerns to one single section of code, every interrupt simply adds an event to the event queue and does nothing else. The event queue holds information about a given event in addition to the event identifier. Processed by the main thread, all code initiated by the event queue can rest easy knowing it will never be touched by an interrupt.

The event queue also supports "unique" events, events that cannot exist in the queue more than once at any given time. This can be used to regulate both input and output, ensuring that any changes to an LED state prior to the LEDs actually getting written out will simply carry over into the current draw call instead of causing an entirely new set of draws.

C. LCD View Management

The LCD screen used for this project supports a wide range of colors and features, but writing out to the entire LCD screen takes a noticeable amount of time, and storing full bitmap images would quickly fill up the on-board memory of the microcontroller. Because of this, and to make things easier to use from the games themselves, the LCD behavior is broken down into "views". Each view consists of a background and a given number of "presets". Each preset is a set of text information such as alignment and background color. By organizing things like this, we can draw the background only when we switch views, from say the game view to the menu view, and otherwise stick to drawing text as it updates in each preset. This cuts down on draw time, to the point where it seems instant.

D. Board Emulator

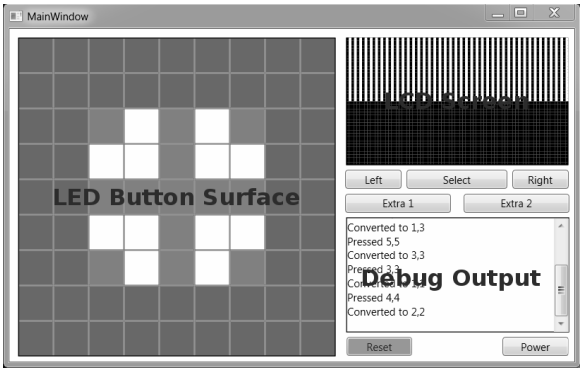


Fig. 7. Board Emulator Window

One advantage to the modular approach in system design is the ability to replace the communication layer for the board with one for a software-based IO device instead. The Management and Game Layers both function as before, but direct their output and take their input from something that can be run on a computer.

This has many advantages that justify the extra development time required to make the emulator. The main one being the ability to develop the software independent of the board itself. This is critical as the amount of game and system software produced for this project was quite large, and being unable to test or properly calibrate it would have severely reduced the amount of time we would have to refine behaviors and fix issues.

Another key benefit lies in bug-testing. If a game software bug shows up in the finished build but does not show up in the emulator, we can narrow down the possible causes considerably. If a bug does show up in the emulator build, it was likely fixed before ever being tested on hardware.

VI. Game Software

In this section, the game software will be discussed. The Game Layer contains all of the logic for the many games we are implementing on our design. Despite being the highest level of software on our chip, there are still many nuances that must be taken into account as well as programming strategies to reduce memory usage. Each

game is coded in Arduino C with the help of our own custom callbacks and functions to promote ease of programming. While there are eight games implemented in our design, Go will be used as the primary example for discussion. This is because Go is our flagship game and is also the most complex, requiring many of the various services provided to the game layer.

A. Using Callbacks and General File Structure

The implementation of callbacks are discussed in detail elsewhere in this document. Here, the usage of these callbacks will be described as well as how they create a veritable structure for each game file. For brevity, this section will only detail the more important callbacks. One of the more important callbacks is OnGameLoaded(){}. As you might guess, any code placed within this callback will be executed when the game is loaded. This is where the structure of every game starts. Generally it is a good practice to limit the amount of code that goes into these callbacks. This is why in Go, the only line of code within OnGameLoaded(){} is calling a local game specific function called InitSetupPhase(){}.

```
GF_PREFIX int GAMEFUNC(OnGameLoaded)()
{
    // Start in the setup phase.
    return InitSetupPhase( 1 ); // Always use a fresh configuration on initial game load.
}
```

Fig. 8. Screencap of OnGameLoaded callback

This external function initializes the LCD menu for the specific game as well as prepares the board itself to play. For example, in Go it creates the necessary menu options, sets the board size to 9 by 9 (our max size), and sets every tile to the neutral tile color. Before it returns, OnGameLoaded(){} must set up everything required to initialize the game. After all the initialization procedures are carried out, the user may select what settings they would like to use for their game. After this is done, they may select "Start Game" from the game menu to start playing. At this point, all the code funnels through the callback OnButtonPressed(int x, int y){}. The code within OnButtonPressed(){} will execute whenever a player presses one of the buttons. This callback receives the x and y coordinates of the specific button pressed. With the practice of limiting the lines of code within these callbacks, OnButtonPressed(){} will usually just call another game specific function. In the code for Go, it simply calls a function called MakeMove(int x, int y) which contains all the logic for handling a tile press. MakeMove will typically have move specific static variables within it, such as the current move counter, in order to contain all relevant player move information.

```
GF_PREFIX void GAMEFUNC(OnButtonPressed)(int x, int y)
{
    MakeMove(x, y);
}
```

Fig. 9. Screencap of OnButtonPressed callback with MakeMove helper function

The remaining callbacks are more situational but still important. Callbacks such as OnLCDButtonPressed(){} are used to take input from the extra buttons, which can be used for input that exists outside of the game-board tiles themselves, such as passing in Go. As for detecting menu usage, OnMenuOptionSelected(){} is all that's needed. OnTimerFinished() is important for setting up events to occur after a given interval in games that require it ;whereas OnLCDTimerHitZero() can be utilized to detect when someone's turn timer runs out in multiplayer games. These callbacks create a unified structure for each game file and assist greatly with programming as well as readability of code. By isolating the code responsible for responding to each event, code becomes much more readable and easy to design.

B. Game Functions

The game functions are another custom addition to the code base designed to assist in programming the games. As expected for a LED board game, the most commonly utilized functions in the code are IlluminateButton(int x, int y, int color), GetButtonColorAtPos(int x, int y), and SetBoardSize(int x, int y). IlluminateButton() simply lights up a tile at the given (x, y) position with the given color. For example, in Go, this function is called every time a player presses a blank tile down to light that tile up with their piece color.

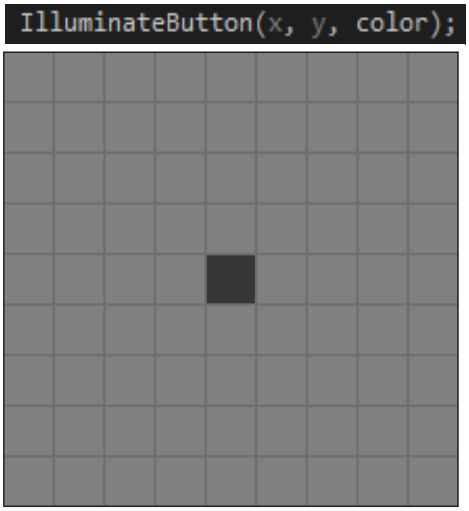


Fig. 10. Emulator example of calling Illuminate Button

GetButtonColorAtPos() is a central part of the logic of each game as it returns the color of a tile at a specific (x, y)

position. With regards to Go, it is used many times to determine which player controls which tiles. For example, if one wanted to determine if a piece was going to be captured in Go, they would call GetButtonColorAtPos() on every adjacent piece around the initial one to see if it is surrounded by enemies. SetBoardSize() is a function that is always called during the initialization of a game. It outlines the playable area of the board in a dull illumination, the size of which is determined by its parameters. In Go, it is called as SetBoardSize(9, 9), which simply activates the entire board for play. Another game example is Four in a Row which initializes as a 7 by 6 board.

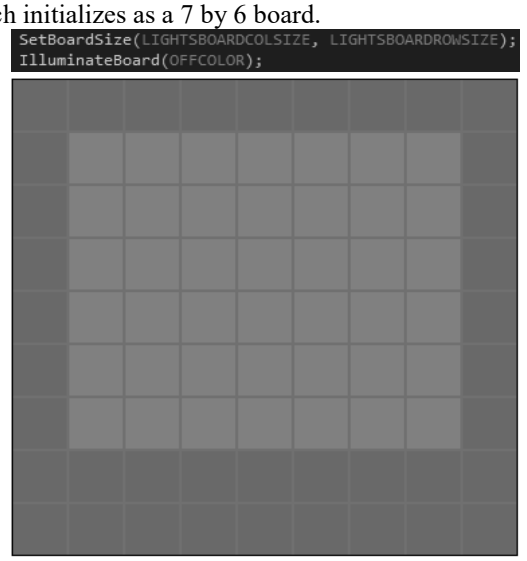


Fig. 11. Example of calling SetBoardSize and IlluminateBoard

C. Programming Strategies:

For different games, different programming strategies must be adapted to make sure the games function properly. Go arguably has the most complex logic but is still possible to implement with well known algorithms. For Go, a graph traversal algorithm had to be implemented in multiple places. An example of this is determining when a player executes a valid capture of an enemy string. A string in Go is when a player has adjacent pieces connected and capturing this string involves completely surrounding it. To determine a valid string, a modified Breadth First Search is utilized. The same goes for territory that is counted at the end of a game. Another traversal is used to determine the captured squares for each player. The difference between this modified Breadth First Search and a traditional one is that for our purposes, we have a custom exit condition. Our traversal checks every single node and once it finds one that is not surrounded, it will break. This is because if one part of the string is not surrounded by pieces, the string is not at risk of capture.

VII. Conclusion

The two semester course of Senior Design has been an invaluable experience for our team. It has taught us, through experience, the design processes and steps involved to develop a concept into a completed and functioning system. By overcoming the challenges and obstacles of designing this LED board game the team has improved their skills massively. We've developed skills such as conducting professional meetings, writing technical documentation, and preparing presentations. After this project we all feel confident that we're ready for our post-academia careers.

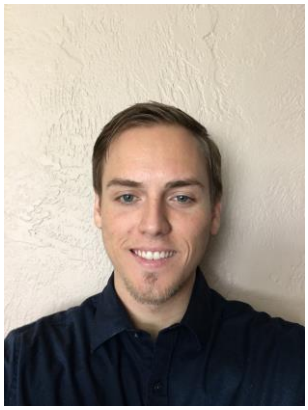
References

[1] Ti.com. (2014). *TPS6109x Synchronous Boost Converter With 2-A Switch*. [online] Available at: http://www.ti.com/lit/ds/symlink/tps61090.pdf [Accessed 13 Jan. 2018].

[2] Ww1.microchip.com. (2008). Stand-Alone System Load Sharing and Li-Ion/Li-Polymer Battery Charge Management Controller. [online] Available at: http://ww1.microchip.com/downloads/en/DeviceDoc/20002090D.pdf [Accessed 13 Jan. 2018].

[3] Industries, Adafruit. "2.2" 18-Bit color TFT LCD display with microSD card breakout." Adafruit Industries, Adafruit Industries, www.adafruit.com/product/1480?gclid=CjwKCAiAr_TQBRB5EiwAC_QCq8JESYsTxaEWM8HsbbtZyx6ngqopMZK7cmOIUrk3ONFdmZUtXNg9rRoCvl8QAvD_BwE.

[4] Amazon. "ALITOVE 16.4ft WS2812B Individually Addressable LED Strip Light 5050 RGB SMD 150 Pixels Dream Color Waterproof IP66 Black PCB 5V DC: Musical, Amazon ,www.amazon.com/ALITOVE-WS2812B-Individually-AddressableWaterproof/dp/B00ZHB9M6A.



Zachery Dunlop is a Senior in Electrical Engineering at the University of Central Florida. He is pursuing a career in the audio engineering industry with a focus on audio electronics.

Kevin Cochran is a Senior at University of Central Florida. He is currently working as an intern at Tetra Tech. After graduating he will continue working at Tetra Tech as a controls engineer.

Taylor Grubbs is a senior Computer Engineering student at the University of Central Florida. After graduating, he will be pursuing a career in software engineering.

Noah Thering is a Senior in Computer Engineering at UCF. Outside of school he keeps himself busy with a variety of personal projects. After graduation he hopes to translate the skills he learned while doing so into a career in embedded systems engineering