Group 12 - Summer 2012 - Fall 2012

December 7, 2012

Shenmin Lo, EE
Joseph Lunder, CpE
Siarhei Traskouski, EE
Robert Wadsworth II, EE

TABLE OF CONTENTS

Index of Figures

Index of Tables

1 EXECUTIVE SUMMARY

Chess is a game of strategy that has been played for hundreds of years, even in modern day chess is a popular game. Two particular trends have developed in the world of chess, correspondence chess and advanced chess playing algorithms run by computers. Correspondence chess is a form of chess that began years ago and has only grown more popular in the modern day. In correspondence chess two players play a game against one another while separated by a distance, sometimes halfway around the world. The moves in correspondence chess are placed by contacting the opposing player through some form of long-distance communication. In the modern day this form of contact has shifted from sending letters through the postal service to sending emails and using correspondence chess servers which cater to the trend. At the same time that correspondence chess was changing chess algorithms were being created. These programs which are run on the computer are aimed at finding a solution to the game of chess. At first they were not very adept at playing versus a live human but in the past few decades these programs have evolved greatly. There are now algorithms that can beat almost anyone one the planet.

At UCF there is a project known as Deep RGB named after the famous chess playing computer Deep Blue. The aim of this project is to develop something unique in the chess world which is a device that combines the two aforementioned trends into one physical product. This item is a chessboard unlike any other it will allow two players to play on it like a standard board. However it also allows a single player to either play a chess computer or to play correspondence chess over the internet. The board itself operates the opponent's pieces whether it is the chess algorithm or an opponent across the country. The board uses magnets to manipulate the pieces moving them when a new move is received from the computer or the opponent and correcting mistakes made by the player, detecting illegal moves etc. It allows multiple users to use one board by up-linking to a server in order to access stored user information.

This user information contains a wealth of preferences allowing the player to set up the board in his or her own special way. It allows them to set anything from their preferred color of the LED array stored beneath the playing surface, to the type of music they would like to listen to while playing. Deep RGB does not just use the audio system to play music selected by the user but also to alert players when certain events occur. Such as when a player is placed in check or when an opponent has made a move, alerting a correspondence chess player that it is their move. It helps those who are new to the game by illuminating potential moves of pieces with the LED array as well as moving pieces to their previous position if an illegal move is made. The board is able to reset the board for another game with the press of a button using magnets under the surface and placed within the pieces to move them to their positions. The server stores saved games so that the board can be set to the last saved state of the user's game. This in tandem

with the many features that are not listed here make Deep RGB one of the greatest products to affect the chess community.

2 PROJECT DESCRIPTION

2.1 MOTIVATION

To a few of the members of the group, the game of chess has been a present, but non-overwhelming, part of life for years. Chess is the ultimate in training to think in strategic and flexible ways and is therefore taught to many children who show even a slight aptitude for the game. Though none of the members of the group developed the focus for chess that is required to become a true master of the game, each of them can enjoy a nice game every once in a while. This casual interest in the game of chess, added to a healthy obsession with robotics, is what allowed for the formation of the idea behind DeepRGB.

Chess is a game that can take a long time to play and if the players involved are not actively participating in a sit-down game, each move of the game can take hours, if not days, to make. Correspondence chess is a very popular way to play the game that involves two players who potentially never meet each other in person, yet compete in a slow, but determined manner. Chess games have been played remotely through the use of standard mail, telegraph, and email, and people more recently have even moved to connecting to remote servers to play games via a graphical user interface program. Though this last method reaches the epitome of convenience in remote, lag-free play, it loses a crucial part of the feel of playing chess, that of being able to lift a piece in your hand and decisively place it in its new position.

Chess, a game that is easy to explain and play but difficult to master, is a perfect match for creation of an automated system to run it. The rules of chess are fairly clear-cut and therefore easy to implement on a small microcontroller, and that same microcontroller allows for communication out to a server to handle the problem-space search and return a new move.

2.2 OBJECTIVES

The objectives for Deep RGB were decided among all the members of the group over the course of several meetings and are intended to support functionality that is believed to be required for a game of chess and also that which will make Deep RGB stand out among both other chess board systems and other senior design projects.

- Main Control Unit
    - Receive information from Hall Effect Sensor Grid

    - Processing of grid for chess piece locations
    - Process possible locations of lifted pieces for LED system
    - Communication with LED and Sound systems for events
    - Communication with Wi-Fi system for sending and receiving board state
- Power Supply
    - Provide stable power to the MCU and each of the subsystems that communicate with it
    - Provide power continuously for the duration of the game of chess
- Hall Effect Sensor Grid
    - Scanned regularly for positions of pieces removed and placed on the board
    - Scanned for pieces not located centrally within the squares
- Magnetic Piece-Moving System
    - Manage grabbing a single chess piece via magnet, moving it, and leaving in place without disrupting any other piece on the board
    - Handle correction of piece location when a human player places a piece towards an edge of a square instead of in the middle.
    - Pieces mounted with small magnets allowing for triggering Hall Effect sensors
    - Magnets in pieces strong enough to be grabbed easily, but not so strong as to affect pieces around them.
- Audio System
    - Receive information on events from MCU
    - Read requested song information from internal SD card and play over integrated speakers
- LED System
    - Light each players pieces in their preferred color
    - Listen for chess piece movement events from MCU and light calculated possible move squares
    - Simulate combat when a piece is taken by varying color of LED in contested space
- Board-based Wi-Fi System
    - Connect to a public or private Wi-Fi network to facilitate communication with server

- Handle making requests to the server to change or view the current state of the board

- Server-based Software
  - Listen for connections from board or web interface and process updates to game states
  - Allow for secure connections from the board and web interface.
  - Store the game state and players for a large number of possible games
  - Recognize when a game is waiting for a computer player and send the board state to the chess engine for processing

2.3 REQUIREMENTS

The requirements for each section were decided upon by the group as a whole after suggestion by the individual group member in charge of that section. The requirements for the body were decided by the entire group together.

Table 1 - Body Requirements

Title	Requirement
BOD01	Weight of entire unit no more than 5kg.
BOD02	Dimensions of playing field no larger than 40cm by 40cm.
BOD03	Smooth playing surface that obfuscates the LED and Hall Effect sensor grid.

The requirements for the Main Control Unit and Wi-Fi System were decided upon by Shenmin Lo and were agreed upon by the other members of the group.

Table 2 - Shenmin Requirements

Title	Requirement
MCU01	Maximum 7 volts input required.
MCU02	Minimum 16KB flash memory space.
MCU03	Minimum 16MHz clock frequency.
MCU04	Minimum 1 SPI and 1 UART bus.
MCU05	Open source schematics and libraries preferable.
MCU06	Minimum 25 digital input and outputs.
MCU07	Free Integrated Development Environment.

WLS01	Minimum 0.1 Mbps transfer rate.
WLS02	Minimum 15 meter range.
WLS03	UART interfacing.
WLS04	Maximum 5 volts input.
WLS05	Low power usage possible.

The requirements for the LCD Display, Magnetic Piece Positioning System, and Power Supply were submitted by Siarhei Traskouski and were agreed upon by the other members of the group.

Table 3 - Siarhei Requirements

Title	Requirement
LCD01	Will display relevant information from the system.
MPP01	Grabber capable of accessing every square of the play area, including the board and graveyard.
MPP02	While moving, the grabber will not affect other pieces on the board.
PSS01	Takes input of American standard 60Hz, 120V AC.
PSS02	Provides stable output power of 3.3V, 5V, and 12V DC.

Robert Wadsworth decided on the requirements for the LED Matrix, Hall Effect Sensor Grid, and Audio System. They were discussed and agreed upon by the other members of the group.

Table 4 - Robert Requirements

Title	Requirement
LED01	8 by 8 grid underneath each square of the chess board.
LED02	Each LED is programmable in 255 steps of each red, green, and blue.
HES01	Coverage of 8 by 8 chess board and two 2 by 8 graveyards for recognition of movement of chess pieces.
HES02	Capable of measuring magnetic forces of up to 5 Tesla.

HES03	Hall Effect sensors in use will not suffer damage from the magnetic fields of the various chess pieces.
HES04	Linear Analog output to Main Control Unit.
AUD01	Play .wav files of 6-36KHz sample rates.
AUD02	Play audio through speaker of at least 8 Ohm.

The requirements for the server software, including the Management Application, Multiplayer Support, Saved Game Support, and Chess Algorithm were submitted by Joseph Lunder and were agreed upon by the group as a whole.

Table 5 - Joseph Requirements

Title	Requirement
MNG01	Allow communication between server and board.
MNG02	Allow for secure, individual access to the server and related games.
MNG03	Respond to move requests within 5 seconds, discounting network lag.
MPS01	Handle connections for both sides of the chess game as well as observer positions.
MPS02	Authenticate users via password.
SGS01	Store up to 100 million individual, in-progress games.
SGS02	Allow for mid-game state save of in-progress games.
CHS01	Interact with one or more chess algorithms to process moves for in-progress games.
CHS02	Run chess algorithm on maximum number of cores available to the system.
CHS03	Run chess algorithm only when processing a move for an in-progress game.
CHS04	Engine will read from Opening Books for initial moves.
CHS05	Engine will utilize Endgame Table Bases for optimum efficiency in endgame conditions.

Title	Requirement
CHS06	Engine will allow for multiple difficulty levels to match various player skill levels.

3 RESEARCH

3.1 SUBSYSTEM RESEARCH

3.1.1 MICROCONTROLLER UNIT

When it comes to microcontrollers, the world of electronics is teeming with various devices that are capable of providing the power and efficiency needed in this project. One of the most crucial decision factors when it comes to selecting an appropriate microcontroller is to opt for the most affordable overall price of the embedded system while satisfying the specifications needed in order to make the DeepRGB functional, efficient and reliable. Another key criterion is the flexibility offered within its programming language. The preferable programming language needed for this project would be C/C++ due to its simple structure and ability to easily interface with libraries needed to control the components of the chess board. Libraries are prewritten subroutines of computer code that help simplify the coding process and ensure that all components are being utilized at the full potential. Custom made libraries take a great deal of time, effort and money to create and would be in our best interest if these were available to us in an open source environment.

3.1.1.1 INTEGRATED DEVELOPMENT ENVIRONMENT COMPARISON

There were numerous microcontrollers capable of handling the functions needed for this project and each came with their own Integrated Development Environment (IDE). Having an open source IDE ensured that the cost remained low while providing us with the flexibility to integrate and modify the libraries needed for this project. Below is a comparison of the most popular IDEs used

3.1.1.1.1 ARDUINO IDE

Keeping all the criterions explained in section 3.2.1 in mind, one of the most popular and user friendly was the Arduino IDE. Arduino's IDE was used to develop applications for ATmega microcontrollers and had an enormous community providing new and innovative open source libraries that could be applied to solve the needs of this project. It was available for free on the internet and equipped with many debugging tools. The

IDE came with many libraries used to control stepper motors and templates were available that show how to receive instructions from wireless devices. Modified versions of it were available that added functionality to the programming software. The IDE used a programming language especially similar to C++ called Wiring capable of running on multiple operating system platforms such as Windows, Linux and OSX.

3.1.1.1.2 MICROCHIP - MPLAB X IDE

The MPLAB X IDE was used to build applications for Microchip microcontrollers. This award winning IDE also had a large community creating massive amounts of libraries open to public use. Its coding language was defaulted to C, but can also be modified to accept assembly code as well. The compiler had a built in debugger and came with multiple examples. It was compatible with multiple operating systems such as the ones described above and was available for free.

3.1.1.1.3 TI MSP430 - IAR EMBEDDED WORKBENCH

There were various IDEs available to code TI's MSP430, the IAR Embedded Workbench was one. It utilized a C and C++ compiler and like most compilers came with a debugger. The IDE had few examples and templates, but none that were relative to our project. The library used to control a stepper motor or receive wireless data from a device would get very complicated to build ourselves and we would prefer an open source alternative. The IDE also had problems installing on other operating systems and no support for Linux nor OSX computers was available.

3.1.1.2 MICROCONTROLLER UNIT SELECTION PROCESS

Taking into consideration the information gained from comparing the different microcontroller IDEs available, we advanced to comparing their hardware counterparts to the needed specifications. With all the features in DeepRGB, the microcontroller needed to be able to execute many instructions per second and also provide many analog and digital inputs and outputs. The ratiometric linear Hall Effect sensors output an analog signal and would need to be converted to digital in to be quantifiable by the microcontroller's software. This means that the microcontroller needed to have analog to digital converters present.

Many of the other components in Deep RGB also required a Clock, Pulse Width Modulation (PWM) and Transfer/Receive (TX, RX) pin-outs to accommodate a wireless data transmission device. The goal was to have the exact amount of pin-outs needed on the microcontroller board, too few would prevent the microcontroller from performing its needed tasks, and too many would unnecessarily increase the cost of the entire project.

A crucial part of the selection process was the availability of the development board's schematic. Having the schematic for the development board would allow us to gain a deeper understanding of the microcontroller's pin-outs and the components needed to assist all the functions present on the development board e.g. power managements, status LEDs, etc.

To prevent DeepRGB from wasting power while not in use, it needed to exhibit low power consumption and the possibility of a standby option that would switch off components that were not being used after a certain amount of time. Having a low power microcontroller would allow the final product to have a small power supply that would not radiate excessive heat. Below are a few microcontroller unit choices selected for comparison.

3.1.1.2.1 ATMEL CORPORATION ATMEGA 2560

Most if not all Arduino compatible microcontrollers designs were available on the Arduino website and are created under a Creative Commons license. Having this wealth of information at our hands allowed us to custom make our own Printed Circuit Boards (PCB)s with all components integrated onto a single board. Having one microcontroller in charge of the other components in DeepRGB was ideal for dependability and cost reasons, but daisy chaining is possible with the majority of microcontroller boards.

The price of most Arduino compatible development boards was affordable and provided us with the ability to purchase these premade boards to setup the initial development process and later integrate this onto our own custom PCB. With the piece positioning system taking up most of the space inside the chess board, the size of the Arduino PCB was not much of a concern.

The ATmega 2560 R3 shown in Figure 1, was one of Arduino's largest microcontrollers available on the market and was the prime example when choosing the correct microcontroller development board. Other Arduino based microcontrollers were basically scaled down versions of the Arduino Mega 2560 R3. Posessing a development board allowed us to start programming DeepRGB and ensure all specifications were met before moving on to create our own PCBs.

The microcontroller development board itself was based on the Atmel Corporation ATmega2560 and utilized Arduino's open source IDE, this allowed it to be backwards compatible with other Arduino microcontrollers. With its open source schematic, we gained a greater understanding of its components and allowed us to remanufacture our own version of the microcontroller. Since the Arduino Mega 2560 was capable of providing us with many digital input and output ports, it could be modified to suit our specific amount of ports needed, whereas to add more ports to a microcontroller would have posed a greater challenge. It also had the convenience of uploading code to it via USB whereas other microcontrollers used a serial port. This would allow us to program the board from virtually any computer available.

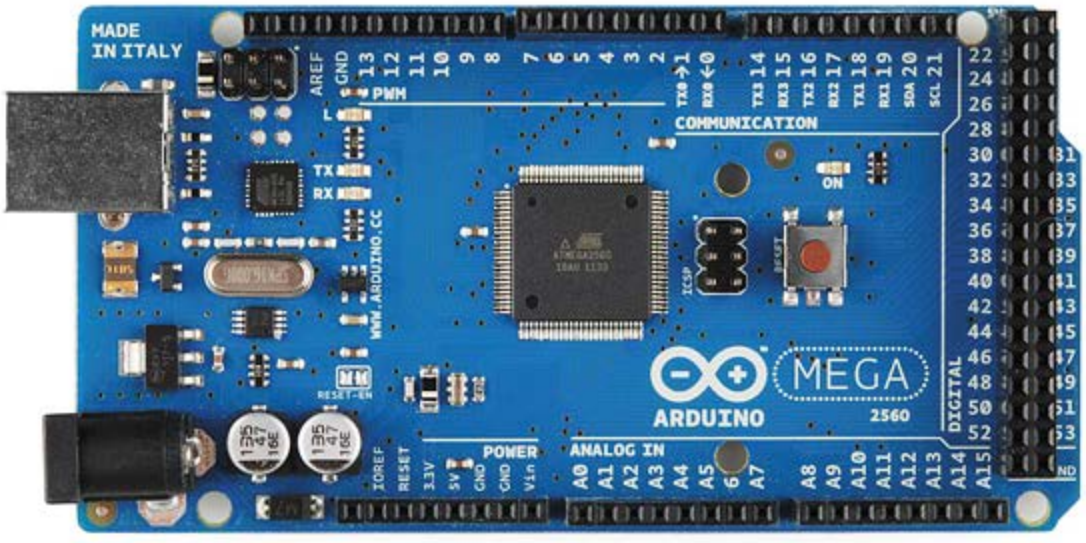


Figure 1 – The Arduino Mega 2560 development board reprinted with permission from Sparkfun.

The microcontroller had a total of 54 digital input and output pins of which 14 could be utilized as PWM outputs if necessary. It also had the capability of reading 16 analog inputs and receiving or transmitting data from 4 hardware serial ports.

Featured below, is a general summary of the Arduino Mega 2560 specifications and characteristics as well as its open source datasheet.

- Microcontroller — ATmega2560
- Operating Voltage — 5V
- Input Voltage (recommended) — 7-12V
- Input Voltage (limits) — 6-20V
- Digital I/O Pins — 54 (of which 14 provide PWM output)
- Analog Input Pins — 16
- DC Current per I/O Pin — 40 mA
- DC Current for 3.3V Pin — 50 mA
- Flash Memory — 256 KB of which 8 KB used by bootloader
- SRAM — 8 KB

- EEPROM 4 KB
- Clock Speed 16 MHz

Most Arduino microcontrollers were either powered by an AC-to-DC adapter, external power source or via the USB cable used to program the board which was automatically selected when the board receives power. If the AC-to-DC adapter option were chosen, it would be plugged into the board via its equipped 2.1mm center-positive power plug to provide the board with the necessary 7-12 volts needed. DeepRGB utilized an AC-to-DC adapter, but the idea of battery power did arise as a valid means of powering the system. This would be done by connecting the leads of the battery to the Gnd and Vin pin headers on the board. The recommended input voltage range for the Arduino Mega 2560 was 7-12 volts, using an adapter would supply the board with the constant regulated input voltage recommended while using a battery could have caused this voltage to drop over time and cause the microcontroller to become unstable if the input voltage dropped below 7 volts. Using a battery would have made DeepRGB more mobile, but would have made the entire enclosure larger in size, heavier and would require the user to plug it into a wall socket for charging every couple of hours.

The ATmega2560 had a built in flash memory of 256 KB used for storing code and the bootloader. The bootloader, a small piece of software (8 KB) that is preloaded onto the microcontroller, allowed us to upload our code without the need of extra hardware. The bootloader would only be active during the first few seconds after the microcontroller is reset and provides a quick startup time. The code that gets uploaded onto the flash memory of the Arduino is called a sketch; these sketches are coded and compiled in Arduino's open source compiler. The compiler was written in Java and based on Processing, avr-gcc and additional open source software.

The 54 on board digital pins could be utilized as both input and output ports using the following built in functions: pinMode(), digitalRead() and digitalWrite() where pinMode() assigns a specific pin to be used as an input or output, digitalRead() requests a digital input from a specified pin and digitalWrite() outputs a digital signal to a pin. The pins operated at 5 volts and were able to supply or receive up to 40 mA. They're also fitted with a pull-up resistor ranging from twenty to fifty Kilo Ohms.

The Arduino Mega 2560 also provided us with many analog inputs; these were extremely crucial in order to make DeepRGB aware of the positioning of chess pieces and provided 10 bits of resolution which allowed us to have 1024 values while measuring individual Hall effect sensors. Along with the default digital pins, the Arduino Mega 2560 had designated pins with specialized functions that provided added functionality and flexibility such as:

- Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).
    - Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.
- External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).
    - These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 2 to 13 and 44 to 46.
    - Provide 8-bit PWM output with the analogWrite() function.
- SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).
    - These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.
- LED: 13. There is a built-in LED connected to digital pin 13.
    - When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- TWI: 20 (SDA) and 21 (SCL). Support TWI communication using the Wire library.

AREF. Reference voltage for the analog inputs.

Used with analogReference().

Reset. Bring this line LOW to reset the microcontroller.

Typically used to add a reset button to shields which block the one on the board.

Most Arduino boards were capable of communicating with other microcontrollers and devices. The Arduino Mega 2560 came equipped with four hardware Universal Asynchronous Receiver/Transmitters (UART) that could be used for Transistor-Transistor Logic (TTL) serial communications. They were crucial for DeepRGB, since we made use of a wireless data transmission device in order to transmit and receive chess piece positions from a server. This data needed to be transmitted and processed by the Arduino and saved in an array. The wireless data transmission device will utilized these UARTs in order to gain access to the Arduino's flash memory. The UARTs were also equipped with an LED that flickers when data is transmitted to and from the Arduino, this helped us with troubleshooting any wireless transmission problems that arose.

Almost all Arduinos came with some sort of overcurrent protection system to prevent any damage from occurring while programming. The Arduino Mega 2560 was equipped with a resettable polyfuse that shielded the computer's USB port from any short circuits that occured. If the current in the USB port exceeded 500mA, the fuse would automatically sever the connection from the computer until the short circuit or overload was removed from the Arduino.

3.1.1.2.2 MICROCHIP PIC18F46K80

The PIC family of microcontrollers designed and built by Microchip were known to be exceptionally efficient and very popular amongst embedded systems developers. These microcontrollers were known for their low cost, broad accessibility, enormous user base and its IDE. There were many examples of projects similar to DeepRGB that used a version of the PIC microcontroller. Microchip's PIC18F46K80 microcontroller was selected to be considered for the project due to the following specifications:

- Program Memory Type — Flash
- Program Memory (KB) — 64
- CPU Speed (MIPS) — 16
- RAM Bytes — 3,648
- Data EEPROM (bytes) — 1024
- Digital Communication Peripherals — 2-A/E/USART, 1-SSP(SPI/I2C)
- Capture/Compare/PWM Peripherals — 4 CCP, 1 ECCP
- Timers — 2 x 8-bit, 3 x 16-bit
- ADC — 11 ch, 12-bit
- Comparators — 2
- CAN — 1 ECAN
- Temperature Range (C) — -40 to 125
- Operating Voltage Range (V) — 1.8 to 5.5
- Pin Count — 44
- XLP — Yes
- Cap Touch Channels — 11

The PIC18F5K80 had many core features that allowed it to be more power efficient. The microcontroller was able to switch between many types of modes to save power and wake back up when more power was required. The microcontroller was capable of running in idle mode that disabled its CPU core but allowed peripherals to maintain an active state. The microcontroller also offered a sleep mode called nanoWatt XLP ideal for low power applications and required only 20nA during this state. The microcontroller also provided 64KB for the application code and was rated for up to 10,000 erase/write cycles while having data retention for up to 20 years.

The development board used to develop and prototype with a PIC 18 series microcontroller was known as the Explorer Board. The development board was capable of handling the entire PIC18 MCU family of processors and would have come in handy if we ever needed to replace a faulty processor or decided to upgrade to a better processor. The PIC18 Explorer Board featured:

- Multiple PIC18 processors, both a PIC18F8722 on board (128KB Flash, 80 pins, superset of traditional PIC18 family), and a PIC18F87J11 Plug-In Module (128KB

Flash, 80-pins, superset of J-series, PIM adjusts to accommodate 3V device). A switch selects the desired processor.
- Supports many other PIC18 devices with Plug-In Modules, supporting 28 to 80-pin PIC18 devices
- PICtail ™ daughter board connector for connection to standard expansion boards such as Ethernet, speech playback, and the many different sensors
- Expansion connector accesses full device pin-out and breadboard prototype area
- Convenient connection for MPLAB PICkit 3, ICD 3 or REAL ICE for in-circuit programming and debugging
- Alpha-numeric LCD display
- USB interface for USB to RS-232 communication
- 25LC256 SPI EEPROM
- Crystal oscillator
- Potentiometer (connected to 10-bit A/D, analog input channel)
- Analog output temperature sensor
- LEDs
- RS-232 port
- Power supply connector and programmable voltage regulator, capable of operation from 2.0V to 5.5V
- Demo software including temperature sensor demo included (illustrates Microchip's analog temperature sensor MPC9701A) and 32 kHz crystal for Real Time Clock demonstration

This microcontroller had a documented example available of a wireless transmission device connected to its UART communication port. The microcontroller was capable of connecting wirelessly, but its 3,648 bytes of RAM hinders this microcontroller and wouldn't allow us to enable all the features needed to make DeepRGB.

3.1.1.2.3 TEXAS INSTRUMENTS MSP430FR5739

The Texas Instruments MSP430FR5739 low-power microcontrollers was made up of several embedded devices with qualities such as FRAM nonvolatile memory, a 16-bit CPU, and different built in peripherals directed for several types of projects. These features in conjunction with seven low-power modes would have been able to achieve a low power solution to the project. This microcontroller featured:

- Embedded Microcontroller
    - 16-Bit RISC Architecture up to 24-MHz Clock
    - Wide Supply Voltage Range (2 V to 3.6 V)
    - -40°C to 85°C Operation
- Optimized Ultra-Low Power Modes
- Ultra-Low Power Ferroelectric RAM
    - Up to 16KB Nonvolatile Memory
    - Ultra-Low Power Writes

    - Fast Write at 125 ns per Word (16KB in 1 ms)
    - Built in Error Coding and Correction (ECC) and Memory Protection Unit (MPU)
    - Designed to Support Energy-Harvesting Applications
    - Universal Memory = Program + Data + Storage
    - 1015 Write Cycle Endurance
    - Radiation Resistant and Nonmagnetic
- Intelligent Digital Peripherals
    - 32-Bit Hardware Multiplier (MPY)
    - Three-Channel Internal DMA
    - Real-Time Clock With Calendar and Alarm Functions
    - Five 16-Bit Timers With up to Three Capture/Compare
    - 16-Bit Cyclic Redundancy Checker (CRC)
- High-Performance Analog
    - 16-Channel Analog Comparator With Voltage Reference and Programmable Hysteresis
    - 14-Channel 10-Bit Analog-to-Digital Converter (ADC) With Internal Reference and Sample-and-Hold 200 ksps at 100-µA Consumption
- Enhanced Serial Communication
    - eUSCI_A0 and eUSCI_A1 Support: UART With Automatic Baud-Rate Detection, IrDA Encode and Decode, SPI at Rates up to 10 Mbps
    - eUSCI_B0 Supports: I2C With Multi-Slave Addressing, SPI at Rates up to 10 Mbps
- Power Management System
    - Fully Integrated LDO
    - Supply Voltage Supervisor for Core and Supply Voltages With Reset Capability
    - Always-On Zero-Power Brownout Detection
    - Serial On-Board Programming With No External Voltage Needed
- Flexible Clock System
    - Fixed-Frequency DCO With Six Selectable Factory-Trimmed Frequencies (Device Dependent)
    - Low-Power Low-Frequency Internal Clock Source (VLO)
    - 32-kHz Crystals (LFXT)
    - High-Frequency Crystals (HFXT)

The development board used to develop and prototype with a MSP430FR5739 microcontroller was known as the MSP-EXP430FR5739 Experimenter Board. The development board was capable of handling the entire MSP430FR57xx microcontroller family of processors and would have came in handy if we ever needed to replace a faulty processor or decided to upgrade to a better processor. The MSP-EXP430FR5739 Experimenter Board featured:

- Integrated MSP430FR5739 :

    - 16KB FRAM / 1KB SRAM
    - 16-Bit RISC Architecture up to 8-MHz
    - 2x Timer_A Blocks, 3x Timer_B Block
    - 1x USCI (UART/SPI/IrDA/I2C) Blocks, 16Ch 10-Bit ADC12_B, 16Ch Comp_D, 32 I/Os
- 3 axis accelerometer
- NTC Thermister
- 8 Display LED's
- Footprint for additional through-hole LDR sensor
- 2 User input Switches
- Connections
    - Connection to MSP-EXP430F5438
    - Connection to most Wireless Daughter Cards (CCxxxx RF)

As seen in the specifications of the developer board, this microcontroller had a documented example on how to connect it with a wireless device. Since DeepRGB needed to be wireless, this board would have been up to the task.

3.1.1.3 MICROCONTROLLER UNIT SELECTION

After researching various components needed for the other features of DeepRGB, Table 6 was made in order to represent the pins needed to control every component in the system. This table will be cross-referenced with a selection of microcontrollers in the list below and will help choose a correct unit.

Table 6 – The pins and busses needed to connect the peripherals.

DeepRGB Device	Control pins needed	Pin type
Liquid Crystal display (LCD)	1 Register Select (RS) 1 Enable (EN) 4 Digital pin outputs to interface with the LCD	Digital I/O pins
Red, Green, Blue Light emitting diode matrix	1 Serial clock (CLK) 1 Chip select (CS1) 1 Master output, slave input (MOSI)	SPI I/O pins
Hall effect sensor matrix	1 Serial clock (CLK) 1 Chip select (CS2) 1 Master output, slave input (MOSI) 1 Analog input	SPI I/O pins Analog read pin
Wireless data transmission device	1 Transmit pin 1 Receive pin	UART I/O pins
XY grid stepper motors	4 direction control	Digital output pins
Electromagnet	1 Enabler 1 Potentiometer control	Digital I/O pin PWM pin
Audio-Sound Module	1 Serial clock (CLK) 1 Chip select (CS3) 1 Master output, slave input (MOSI)	SPI I/O pins

	1 Master input, slave output (MISO)	

With all the features of the microcontroller obtained, a comparison of the three microcontrollers is shown in Table 7:

Table 7 – Comparing the three microcontrollers in order to make a final decision.

	PIC18F46K80	MSP430FR5739	Atmel Corporation ATmega 2560
Operating voltage	1.8 - 5.5 V	2 - 3.6 V	2.7 - 5.5 V
Digital I/O pins	35	33	54
Anlaog input pins	11	14	16
UART & SPI busses	3	3	4
Program memory	64 KB	16 KB	256KB
Clock speed	64 MHz	24 MHz	16 MHz
Experience with product	None	None	Very experienced
Price per microcontroller	$4.30	$6.35	$17.97
Price per development board	$165.00	$29.00	$58.95

The PIC18F46K80 seemed like the logical choice due to its high clock speed, but due to the high cost of its development environment and small RAM size it the least desirable amongst the three. The MSP430FR5739 came in second place due to its small program memory, a lack of an open source schematic and insufficient experience with the product. A MSP430 LaunchPad was bought in order to try to gain more experience with the product and further research lead to the discovery of the absence of libraries needed to control all of the devices for this project. This lead to the final conclusion of using the Atmel Corporation ATmega 2560 microcontroller and its Arduino ATmega 2560 development board.

3.1.2 HALL EFFECT PIECE DETECTION SYSTEM

When Deep RGB began it was unanimously decided that the pieces on the board would be moved via a magnet under the board, as to avoid the clumsiness that a robotic arm would bring to playing a game of chess. With this decision came a new challenge in the form of a detection system that must be integrated into the board in order for it to be able to move the pieces. The board could not simply "remember" where the pieces are by storing the positions in memory since a human would be moving some of the pieces themselves. Therefore the board had to have a way to actively detect where pieces were. There are several ways this could have been done including RFI, the pieces would have transmitters installed and the board could be equipped to triangulate their

position. This however would not only have been expensive but difficult since each piece would have had to broadcast a specific signal.

After further thought it was decided that the board could remember what pieces were where by using Hall Effect sensors to detect which pieces were moved and where. This would also not only be an appropriate way for the board to keep track of the user input and continue the game, but would allow the board to add new features. The main feature that became available was the ability to correct for human error and allow the board to move pieces the user had moved previously. The board could re-center pieces that were placed too close to the edge of a square or move them back to their previous position if the user placed them on a line and the appropriate square could not be determined. It added the ability to move user pieces to the graveyard when they were taken by a piece on the side of the computer. This made Deep RGB that much sleeker since it wasn't hassling the user to remove pieces from the board and the user no longer had to be careful about placing pieces in the exact center of the tile since Deep RGB could take care of that for them.

Several types of Hall Effect sensors were researched such as switches, latches and ratio metrics. Other variations existed but these three were rather common. To start switches, like their name sounds switch on when detecting a magnetic field. This would have hardly been useful for a chessboard since it was either on or off and while telling us that there was definitely a piece in the area of the sensor or that there is none around whatsoever it would not tell us much more than that. A latch turns on in the presence of a magnetic field but will stay on until an opposite magnetic field is applied. Like the switch the latch stays on at a constant voltage so all it could tell us is whether or not a piece had moved by it and if a piece moved by it on the opposite side. This like the switch was useless for the chessboard. A ratio metric Hall Effect sensor not only turns on when in the presence of a magnetic field it also changes the voltage output to correspond with the distance of the field's source from the sensor. This was exactly what Deep RGB needed to monitor the pieces on the board.

If a net of ratio metric Hall Effect sensors were placed in/under the board they could be used to calculate the position of every piece on the board. This also needed to be implemented in the graveyards so that the board could place taken pieces appropriately, making an automatic board reset and the promotion of pawns possible.

Since the project is rather large it was more cost effective to create the net with through-hole components eliminating the need for a costly PCB manufacture. The next parameter of concern was the sensitivity of the sensor. It could not be too sensitive for it would sense fields from too far away and reach maximum output before the piece was in position. This could also have been solved by using a component with a wider operating range so that the maximum output is higher and therefore it could sense larger/stronger magnetic fields. The magnets that were being considered at the time

could create a magnetic field of about 3,498 Gauss at the surface of the magnet. Most Hall Effect sensors would reach maximum output before the magnet was close to the sensor.

This problem had been considered and two solutions were devised both are appropriate and only require a small amount of recoding to the detection software to switch from one to the other. The program could have been written with both algorithms and then all that would be needed is a simple input to switch from one to the other.

3.1.2.1 TRIANGLE ARRANGEMENT

The first solution to the problem of maxing out the sensors was triangulation which created a space about half an inch or more between the sensors and the magnets when the pieces were centered in the tile. With this arrangement there would be three sensors for each tile on the board to facilitate triangulation. It would be possible to use fewer sensors and simply triangulate on a wider scale but this would raise problems, the magnets would need to be much stronger which could affect how the movement of the pieces worked. The positioning program would also need to calculate whether or not the piece was in the center of the appropriate square using not only less information but less accurate information and doing so would have slowed down the game play. It would be much simpler to have three sensors equal distances from the center that way the program would have only had to make sure each sensor is supplying outputs that are similar to each other. The three sensors would never have read the same field strength since the polarity relative to the face of each sensor would be different. The main flaw with the triangulation is the fact that there would have been a very large amount of sensors used. This would not only cost more money but would cost more power as well and create the challenge of finding a way to supply the data to the MCU without using a vast amount of I/O ports. A situation where this method should have been used is when the sensors used would be damaged by the magnetic fields produced by the pieces and the movement device because they wouldn't be in the direct path of a magnet.

3.1.2.2 ALL CORNERS ARRANGEMENT

The second solution to the sensor problem was using one sensor for every corner created by the tiles. This created four data points per tile and therefore supplied the program with more data from which to extrapolate the information needed about each piece's position. However if a sensor was placed at every corner the typical sensor could detect anywhere from 0 to 4 pieces at one time. In light of this fact a register with the previous data taken from each sensor would have to be implemented so that the program could test the change in readings. One major benefit of this design was that less sensors were used, reducing cost and need for input ports on the MCU. This method would be used when the sensors available would not be damaged by the high magnetic field created when a piece is moved down the line to a new position. For

reference, the approximate field strength of our magnets at various distances is available in Table 8.

Table 8 - A table showing the approximate magnetic field strength from the magnet being considered at different distances.

Distance From Center of Magnet(inches)	Magnetic Field Strength(Gauss)
0.0	3,498
0.5	1,707.69
.75	505.982
1.0	213.461

In addition to the field (playing area), there would be two graveyards which raised the total number of tiles to 96. If one of the two arrangements of sensors detailed above were to be used it would require 117 to 288 sensors for the all corners and triangulation methods respectively. To solve the problem of using too many I/O pins analog multiplexers could be used as switches to allow a single sensor signal through at one time. This could be done for both arrangements and the setup would have been quite similar the primary difference would have been that it would require fewer registers for the corners method. Every sensor's anode in a row would need to be connected to a bus, for the triangle arrangement only sensors of the same position i.e. top, left, right would be connected together resulting in three busses per row. The cathodes of each sensor in a column was connected to a bus, each bar will feed into a different input on an analog MUX. For the triangle arrangement each position would go to its own MUX so there would be a top MUX, as well as a left and right MUX. Each anode bar fed into a different input on an analog MUX. By doing this each sensor could be turned on by selecting the right input on the corresponding cathode MUX and the anode MUX. This could have been done quickly as the sensor power up time was about 30 microseconds. Every output would feed into the same analog input on the MCU since the sensors will only be on one at a time there was no need for more than one input.

However even though there was only one analog input needed to read the data there could have been either 8 or 13 digital outputs needed to control the MUXs depending on the arrangement used. This was reduced by using shift registers which needed only 3 digital outputs to be controlled bringing the total I/O pins used by either arrangement down to 4.

3.1.3 MAGNETIC PIECE-MOVING SYSTEM

We began to research our project by investigating X and Y positioning tables. The XY rails were to be located underneath the chess board and capable of moving from one X-Y coordinate to another simultaneously in order to move certain chess pieces. This was done by integrating a magnet into the base of each piece and using a strong magnet underneath the board to move the pieces. Choosing the right positioning system was a crucial part of this project and is what made this project seem so magical. We

considered three different approaches to creating the complex positioning system. The first step was choosing the type of magnets used in the chess pieces, the strong magnet to attach to the XY rail system and motors that were capable of providing the accuracy we required.

Most of the XY rail systems we were researching had similar components: they all had two motors and gear system which transferred the rotational movement of motors into linear movement in the X and Y directions. The first approach, as seen in Figure 2, we considered for moving the system was based on an internal ring gear. Motors with cylindrical shaped gears attached to them interacted with a rack gear on the board. When the motors spin they moved the system back and forth along the racks to move the system to any one point under the board.

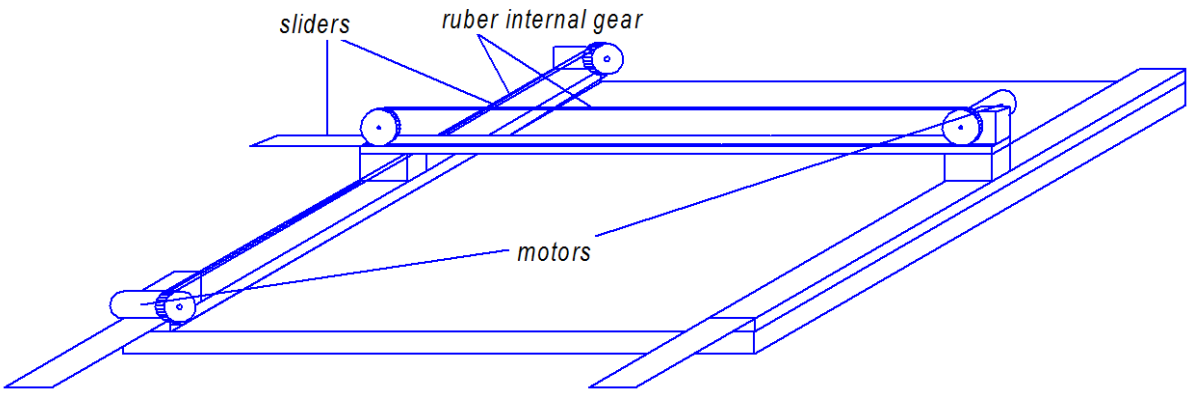


Figure 2 - The first choice for the system

The second approach to the movement problem was the worm gear system shown in Figure 3. This system used two sets of aluminum rods arranged in the X and Y directions in order to guide the motor assembly. A car would ride along each rod and support a worm gear between them, the motors would use these worm gears to move back and forth along their axis and achieve mobility in the X and Y directions.

The final approach was a system based on racks and gears. It had a pair of glides in Y direction and one glide in X direction as depicted in Figure 4. The pair of glides in the Y direction was connected by an arm under the middle of which a motor was mounted which used a gear to move along a rack. Atop the arm another rack and glide were affixed allowing a second motor to move the system in the X direction.

The most important properties for us to consider when deciding on which movement system to use were cost, level of difficulty during the installation, reliability and level of noise made during operation. The first system had many moving parts and used an internal ring gear made from rubber which could slip off or break, so reliability of that system was less than that of other systems. As far as noise level, the first system had higher level of noise as well because of the many moving parts. The first system was not

too expensive since it did not require many advanced parts however the fact that it required more parts slightly offset this fact making it the second most expensive of the systems. The rack and gear system was more reliable because it had the least amount of moving parts and used rack gears instead of a ring. The noise level of the rack and gear system was also lower than the first system. Since the rack and gear system was simple and made of relatively few parts it was the cheapest of the three systems. The most reliable of all the systems was the second system with a worm gear because it used aluminum guide rods and worm gears in the middle which made the system more balanced compared to all other systems and also produced the lowest level of noise. Cost wise however, the second system was the most expensive because it consisted of technologically advanced parts such as linear bearing rings which were quite expensive and aluminum rods that were hard to manipulate. A comparison of the positioning system possibilities is shown in Table 9.

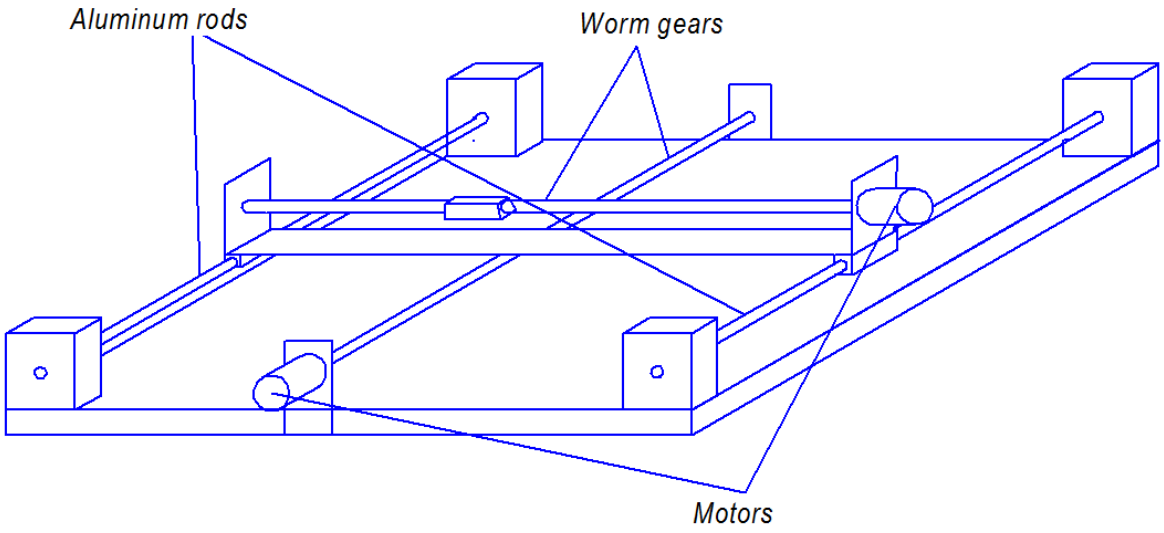


Figure 3 - The second choice for the system.

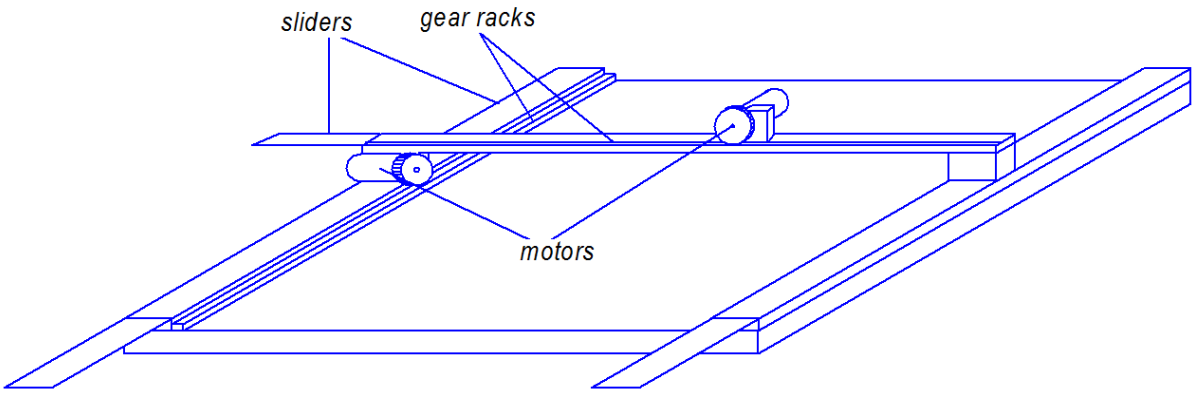


Figure 4 - The third choice for the system

Table 9 - Summary Characteristics of XY Positioning Tables

	Noise level	Difficulty of installation	Reliability	Cost of the system
Internal gear	High	Middle	Low	Middle
Worm gear	Low	High	High	High
Rack gear	Middle	Low	Middle	Low

The rack gear system had the best set of qualities for our project that is why we decided to move forward with this system in our project. The only big disadvantage of a rack gear system compared to a worm gear system is the glides. When the motor moved the positioning magnet to the end of the platform the glides slide out, this would take a lot of extra space when we tried to install this XY table inside a box. During our discussions with the group we decided to consider creation of the hybrid system which includes best qualities of worm gear system and rack gear system.

On the top of X axis we needed to install the magnet system to attract the magnets in the bottom of each chess piece. This would allow us to move each figure by moving our magnet system underneath the board. One way to create this magnet system was to use a servo motor which raised and lowered a powerful magnet, this approach required the purchase of an additional motor and increased the total cost of the system. The extra motor would need extra inputs from a microcontroller and in our case when we had a lot of sensors actively reading, it would reduce the reaction time of the microcontroller. Such a system would also require additional height to move the rod with the magnet. Another approach was to create a magnet system where an electromagnet was attached to the top of the X axis. In this case we would not need to use a servo to raise the magnet; we just needed to position the electromagnet under the right chess piece and then pull the piece into the next X and Y position and release it by turning off the electromagnet. In addition this approach saved us extra outputs on the microcontroller because we only needed to send a simple signal to turn the electromagnet on and off.

Electromagnets typically came in two types; Horizontal Electromagnets and Vertical Electromagnets. Since we needed to attract chess Pieces from above the electromagnet we needed to use a vertical electromagnet. Vertical electromagnets could have different shapes such as cylindrical, rectangular or cubical they also could have opposite poles and parallel poles. Due to the fact that base of chess pieces had a round form; we needed to look into using a cylindrical shape electromagnet so it would not attract neighboring pieces.

The diameter of our electromagnet was limited to no larger than 3.81cm and no smaller than 2.54cm. Our first choice was the R-1207-12 produced by the “Magnetech Corporation”. This electromagnet had a small diameter of 1-1/4” and a big holding value of 45 lb although holding value was calculated with no air gap between the magnet and the surface it was attracting we could still attract other magnets at a distance because the magnetic force did not dissipate immediately. The rule of thumb for practical

magnetic fields dictated that the electromagnet diameter and field distance was 4 to 1, or the field distance was one fourth of the electromagnet diameter. Since the electromagnet's diameter was 3.175cm, our magnetic field distance was approximately 0.95cm. Due to the fact that we had to move magnets on the top of the chess board our magnetic field distance needed to be bigger than 0.95 cm.

The EM 137 manufactured by "APW Company" was our second choice. This electromagnet had 3.47cm diameter and 33-44lb holding value and the voltage supply for this magnet was 12V. Also it used standard through-hole mounting and included brackets to hold it in place.

Our last choice was the ER2-103 made by "Industrial Magnetic Inc." The magnet had the same diameter as our previous two choices but needed a 24 V supply and had smaller holding force value. This electromagnet was twice as expensive as the other electromagnets. We found during our research that there were not many companies selling small, powerful and cheap electromagnets. We thought the reason was that not many DIY projects involved small and strong electromagnets. But on the other hand there were a lot of inquiries on the Internet from people about how to create electromagnets at home. A comparison of the electromagnets compared was made in Table 10.

Table 10 - Electromagnet Specification Comparison

Model of electromagnet	**R-1207-12**	**EM 137**	**ER2-103**
Voltage, V	12	12	24
Current	DC	DC	DC
Duty	Continuous	Continuous	Continuous
Watts	3.3	5	4.2
Amps	0.28	0.41	n/a
Holding Force, Lbs	45	33	22
Weight, Lbs	0.24	0.24	0.4
Diameter, cm	3.175	3.493	3.175
Height, cm	1.905	2.06	n/a
Price, $	40	29.24	76.06

The size of the positioning system was also one of the important factors to us. We didn't want to create big and bulky chess board. Based on specifications the board could not be bigger than 70cm by 70cm with a height no larger than 30cm. Based on this data we needed to choose optimal size and strength of the motors.

In addition to the electromagnet we needed to choose some magnets which were to be installed in the chess pieces. After some research, we realized that there are many places where we could acquire magnets. The smallest magnets and strongest ones were neodymium magnets. Those magnets were very strong and the prices for them were

almost the same in every store we checked. Finally we chose to purchase the magnets from the website amazon.com.

Stepper Motor

We came to the point where we needed to research motors and motor controllers for our positioning system. The block diagram below (Figure 5) showed the basic path from sensors to motors in order to move the electromagnet to a specific location. Positioning sensors registered the chess piece position on the board and sent this information to the MCU. After it has processed the data from sensors, the MCU sent feed back to the motor controller in the form of different voltage and current levels. The motor controller then sent certain signals to the motors in order to move the electromagnet to a specific location.

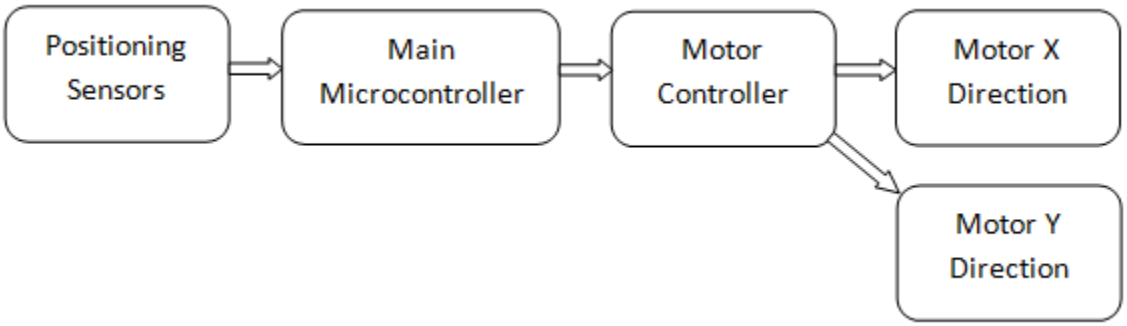


Figure 5 - Block diagram about motor controlling

There were many different types of motors used for different applications. There were also many motor controllers used to control many different types of motors. In our research we decided to consider all possible motor types we could use for our design in order to find the best solution for Deep RGB. Of all the different motor variations, there were basically two main motor types, DC and AC motors. Our research included AC motors, liner actuators, DC brushed, DC brushless and stepper motors.

The first type of motors we considered was the AC motor. AC motors were divided into two main categories the synchronous type and the induction type. Synchronous AC motors synchronized the rotations of an electromagnet with the frequency of the AC power to run the motor. In this type of motors the magnetic field was motionlessly relative to the rotor. Synchronism with line frequency allowed the motors to run very smoothly and quietly since the sine waves allowed smooth transition from one phase to another. Due to the fact that the AC synchronous motors span at the same rate as the line frequency they had a low torque value. As a result these motors usually came with a gear box to increase the torque. The presence of the gear box increased the cost of the system. Figure 6 shows how to control an AC motor with a PIC microcontroller

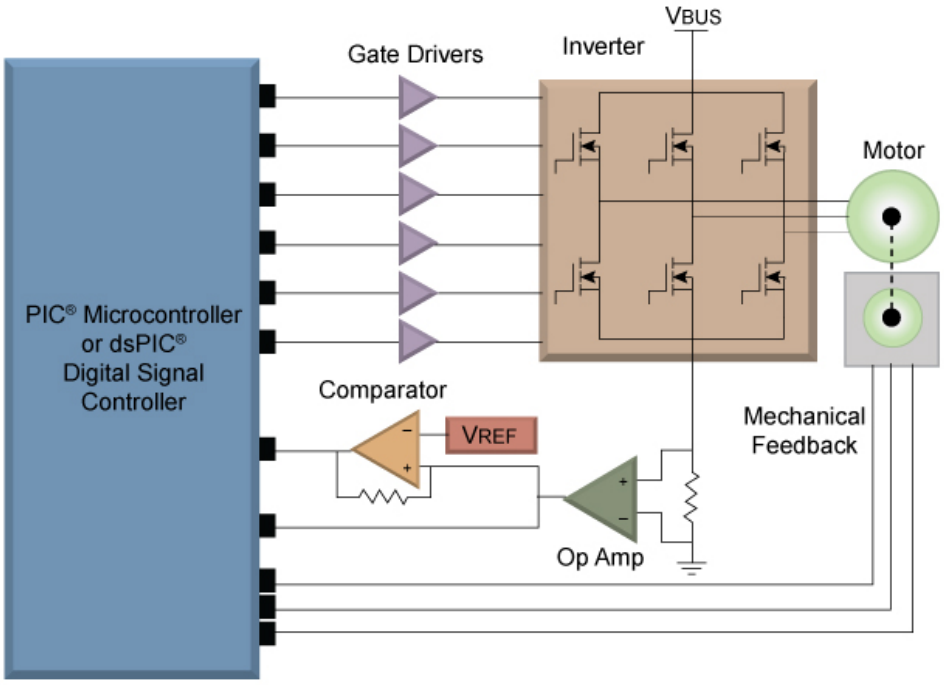


Figure 6 - Block diagram of controlling AC motor by microcontroller

Induction type AC motors were cheaper compared to synchronous motors due to a less complicated production process. This type of motor acted as an asynchronous AC motor where the electrical power was transferred to the rotor from the stator by an electromagnetic field. The noise level of these motors was lower compared to asynchronous motors and they were easy to start. However induction AC motors were less power efficient than synchronous motors. There were signals from MCU sent to the power inverter which converted them into AC power to supply the three phases of the motor. To control motor movement the MCU sent two currents with two different phases and information from a tachometer about the physical movement and position of the AC motor. During our research we decided to connect only one motor, therefore we needed approximately five inputs and five outputs from the MCU. Since we needed at least two motors the amount of I/O pins needed from the MCU would double. Overall AC motors were not very popular to use in small electronic projects. There were also not many motor drivers available for such motors.

The next type of motors we researched was linear actuators; linear actuators transferred different energy such as pneumatic, hydraulic, and electrical into the movement. In our case we were interested in the usage of DC power to transfer energy of rotation to the linear movement. DC linear actuators were quiet, reliable and fast. They tended to already come with preinstalled gear boxes and used a lead screw. They also tended to be driven by a permanent magnet DC motor. The Basic actuator came only with two wires one for negative and one for positive voltage. Different types of

actuators needed 5, 6, 12 or 24 volts from the power supply. To start an actuator all you needed to do was just provide power to its wires and the actuator would start moving.

When the power was disconnected an actuator would hold its position, this option might have been useful for our project because we needed our positioning system to stay in a given position until we wanted to move the electromagnet to the next position. The speed of the linear actuator was regulated by applying different voltage level so in case we needed to decrease speed we would lower the voltage supply. To do so we could have used a PWM adjusted to a required level so we would have the speed we needed for our application.

Due to the fact that linear actuators have wide use in a positioning system tables we could have used them for our movement system. After we did our research on the actuators pricing, the cheapest actuator we found was from www.Firgelli.com at the price of $80. Since we needed two of them for our XY positioning table total cost of actuators will double so this became too expensive for our budget. Another disadvantage is that price rose dramatically when the actuator length increased.

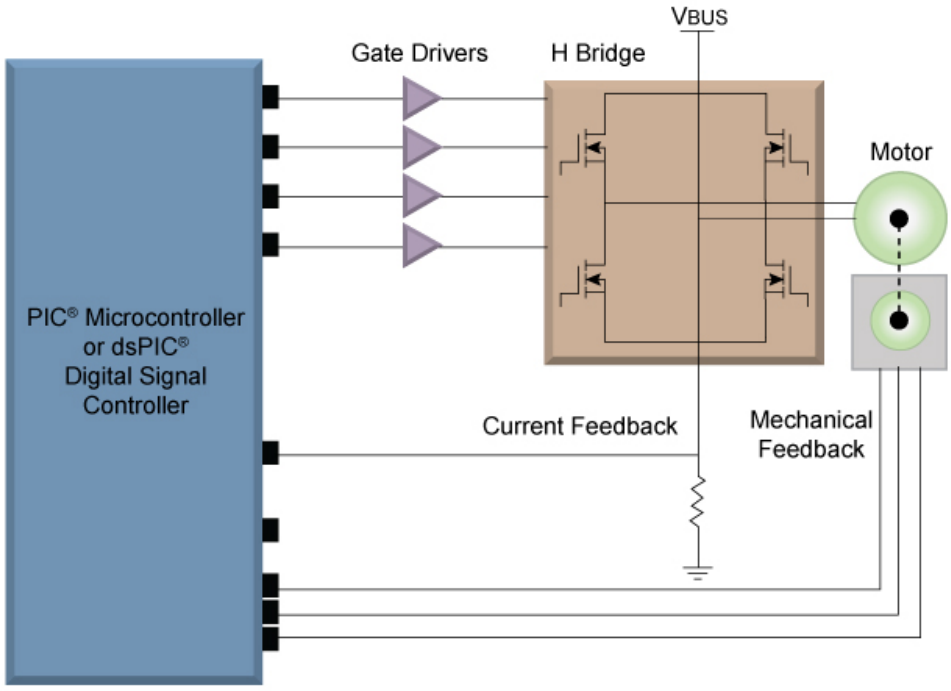


Figure 7 - DC motor control block diagram

DC motors came in two main categories: brushed DC motors and brushless DC motors. Both motor types used DC current to run but since DC has a constant voltage, we needed to make use of the MCU's PWM pins. We needed to know the exact position of

the DC motor, to be able to get this information, we needed to have feedback from the motor to the motor control system so that microcontroller could adjust position of the DC motor based on received information from sensors and feedback. We mapped out the basic connection of DC Brushed motor from Figure 7.

Brushed DC motors have permanent magnets installed inside them which act on the rotor through brushes. On the other hand, brushless motors did not have brushes because the rotor was made of permanent magnets while the stator was made of electromagnets controlled by an IC which eliminated the need of the brushes. Typically, Hall Effect sensors are used to activate one stator's coil after another. For our project we compared brushed and brushless motors to decide which type is more suitable for us. We found that each type of motors had its own advantages and disadvantages. We created a table were we described advantages and disadvantages of those motors. Table 11 is where we first describe brushed DC motor.

Table 11 - Advantages and disadvantages of brushed and brushless DC motors

Advantages	Disadvantages
Brushed DC Motors	
Brushes are replaceable	Maintenance required
Low construction cost	Poor heat dissipation
For fixed speeds not required controller	Lower speed range
Two wire control	Higher electronic noise
Simple and inexpensive control	
Brushless DC Motors	
No voltage drop across brushes	Complex control
Small size and high output	Higher construction cost
Require much less maintenance since there are no bushes	
Higher speed range	
Low electronic noise	
Commutation based on Hall Sensors	

The two most commonly used DC motors for projects like ours were stepper motors and servo motors. We compared two types of motors to choose the motor type which would suit our project the best. Servo motors were more expensive compared to stepper motors of the same size and power level. Versatility wise, stepper motors were better because they were used in anything from clocks to scanners while servo motors were used more often in larger projects. Servo motors came in wider variety of frame sizes but stepper motors could still be found in small and medium size frames. Stepper motors were easier to setup, all you needed to do was connect wires from a motor to the stepper motor driver. The noise levels were better in servo motors; however our motors would be inside of the chess board which reduces noise level. Moreover,

stepper motors were mostly used in a direct drive mode so you could just attach the shaft of the motor directly while servo motors had high RPM and usually requires more gearing ratios compared to stepper motors. As we could see stepper motors were better cost, easier to setup, we could connect them directly, they are very accurate in positioning of their shifts and they had optimal proportionality of power to size which was enough for our project. Because of all of those characteristics we decided to use stepper motors for our project.

Stepper motors did not have brushes like servo motors. Full rotation of the motor was divided into a number of steps. Most common stepper motors were manufactured with 180, 200 or 400 steps per revolution which created stepping angles of 2, 1.8 and 0.9 degree per step. For our project we chose to use a motor with 200 steps per revolution because it had a better combination of the number of steps and cost. The motor could also be driven with or without a stepper motor driver. A stepper motor driver allowed us to increase the manufactured amount of steps by dividing the step number into smaller portions. We decided to use a stepper motor driver because we needed to have the ability to be very precise in positioning our electromagnet. There were three types of stepper motors: permanent magnet stepper, variable reluctance stepper and hybrid step motor. The hybrid is a combination of two other types and it gave us better performance with respect to power, speed and a number of steps. Thus the hybrid stepper motor was chosen for our project.

There were two basic types of stepper motors based on the coil connection. One was the unipolar motor and the other bipolar. Hybrid step unipolar motors had 5 or 6 wires which were wired to the end of two windings with a center tap on each of them. Hybrid step bipolar motors had 4 wires which were wired to the end of two windings. Based on this, bipolar motors offered bidirectional current flow while unipolar motors allowed having the current flow in half of windings since torque was directly related to winding current bipolar motors could create much more torque compared to unipolar motors. Wires in unipolar motors were also thinner and this directly related to increasing heat loss in unipolar motors and also required more wires, while on the other side bipolar motors required more complicated circuitry, typically with an H-bridge connection. Since stepper motor drivers were quite cheap using bipolar motors did not increase the cost of our moving system. All these facts led us to the decision that we needed to use bipolar stepper motors for our moving system design. Figure 8 illustrates the basic connection of the stepper motor with a microcontroller.

From this figure we can see that to connect the stepper motor we did not need to have a feedback loop due to the specifics of the construction of the stepper which were discussed earlier in the paper.

During our research about bipolar stepper motors our first choice was to acquire a Stepper Motor with 200 steps/rev, 12V 350mA. This motor had many features that we were looking for. It had 4-wire bipolar stepper with 1.8 degree per step for more precise

positioning and had a good holding torque at 28oz-in. Max current was 350mA, suitable for our motor driver and could be easily supplied by a power adapter

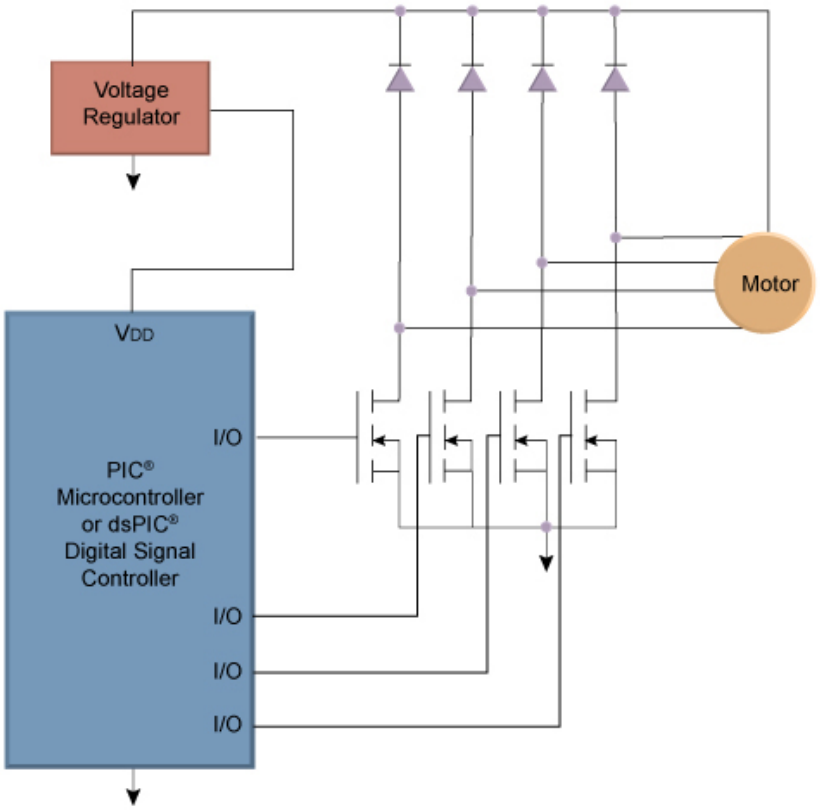


Figure 8 - Stepper motor control block Diagram

Our second choice for the motor was a Unipolar/Bipolar, 200 steps/rev, 4V 1200mA. This step motor was a hybrid and could be used as bipolar or unipolar stepper motor. It had 6 wires and a 1.8 degree step angle. Every phase consume 1200mA at 4V which gave a high holding torque at 3170g-cm (44oz-in). This motor had a bigger frame size (42mmx42mm) but it also had a quite large holding torque which was almost twice as much as our first choice motor and frame size was only little bit bigger.

Another potential motor was Applied Motion – 5017 – 009 Bipolar Stepper Motor. This motor had a medium level of torque at 31.4oz-in compared to previous two motors, 6 leads which allowed us to use this motor as bipolar or unipolar also it had a frame size of NEMA 17 (42mmx42mm). The motor consumed 570mA current per phase that gave it medium power consumption. Weight of motors was not as important a factor in our project as a frame size, torque, and power consumption.

The Mercury ROB-09238 Bipolar Stepper Motor was our next choice. This motor had only 4- wires which made it simple and powerful at the same time. It had 1.8 degree step angle, 2 phases and it consumed 330mA current and 12V voltage. Holding torque was 2300g-cm (31.9oz-in), frame size was standard. Parameters for this motor were

similar to a previous motor the difference between them was that this motor had 4 lead wires instead of 6 so it could only be used as a bipolar motor where the other motor could be used as unipolar as well as bipolar. We displayed data of all stepper motors we have to choose from in Table 12 below.

Table 12 - Motor Specifications Comparison

Motor Model #	Stepper Motor -200 steps/rev	Unipolar/Bipolar, 200 steps/rev	Applied Motion –5017–009 Bipolar Stepper Motor	Mercury ROB-09238 Bipolar Stepper Motor
Motor type	Bipolar	Unipolar/Bipolar	Unipolar/Bipolar	Bipolar
Step Angel, degree	1.8	1.8	1.8	1.8
# of Wire Leads	4	6	6	4
Leads length, mm	230	300	305	1200
Drive Shaft Diameter, mm	5	5	5	5
Rated Voltage, V	12	4	6	12
Rated Current, mA	350	1200	570	330
Holding Torque, oz-in	28	44	31.4	31.9
Winding Resistance, Ohm	34	3.3	15	34
Frame Size, mm	42.3 x 42.3	42.3 x 42.3	42.3 x 42.3	42.3 x 42.3
Weights, g	200	350	n/a	200
Price, $	14	19.95	12.95	14.95

We chose to use a stepper motor for our movement system for its accuracy. For example our stepper motor had 1.8 degree per step so it made 200 steps for a full turn of the motor rotor. To move the motor a certain number of steps the controller would send the same number of impulses to the motor and that regulated the speed of the rotation through the frequency of pulses from the controller.

Stepper Motor Controller

During our research we realized that there were many different motor controllers available on the market. Due to the variety of controllers that were available it was hard for us to choose a controller for our project. However, after we realized that all stepper motor controllers can be divided into simple categories it simplified the process. Motor controllers could be divided into two basic categories.

The first type of controllers had a constant voltage. These controllers supplied a small voltage to each coil. In order to make the motor run they would turn on or turn off voltage from specific coils using transistors. It created certain limitations on using such controllers during high speed since when a motor started turning it created a counter voltage which could be as high as the supply voltage. However the price of these controllers was very low and they could be used in applications where the speed of the stepper is no greater than 2 or 3 turns per second. This speed was relatively low for our project but it still was considered since we were planning to use cylindrical shaped gear attached to the stepper motor shaft which increase length of one turn.

The second type of controllers had constant current. In this case the stepper motor used a much higher voltage compared to the constant voltage controller. Due to the higher voltage, the motor also received a much higher current which could have ruined the coils; one way to protect coils was to use large resistors. Another way was to turn the voltage quickly on and off by using pulse width modulation (PWM). This technique was more complicated compared to usage of large resistors since we needed to adjust current to low or high levels. However the advantage of this method was a low power leakage.

Table 13 - One step voltage sequence is applied to the bipolar stepper motor.

Sequence	Coil 1	Coil 2
1	High+	Low
2	Low	High+
3	High+	Low
4	Low	High-

The simple way to control a bipolar stepper motor was to provide a specific impulse sequence from the microcontroller to the motor. To do so we needed to supply a specific voltage to each coil so the coils got energized and not energized in a specific way. In Table 13, we show the sequence of voltage needed to be applied to bipolar stepper motor coils to perform full step.

We could provide the sequence from Table 13 to the microcontroller through the H-bridge. The H-bridge worked as a set of switches which turned coils on and off and even reversed polarity so the motor could move in a reverse direction as needed.

The usage of a controller built on an H-bridge was under serious consideration by our group. One of the biggest reasons was that it was the most inexpensive motor controller that could be built from simple electronic parts; one coil required a full H-bridge for control. It consisted of a set of 4 transistors and resistors to limit the current. It could also be built by using MOSFETs instead of transistors. We provide a simple diagram of H-bridge on the Figure 9. Since we had two coils in the bipolar stepper motor to control it we needed two full H- bridges. We needed to have a total of 16 transistors and 16 resistors to control two motors. Since building motor controller was not our priority task in this project we decided to search for available solutions on the market.

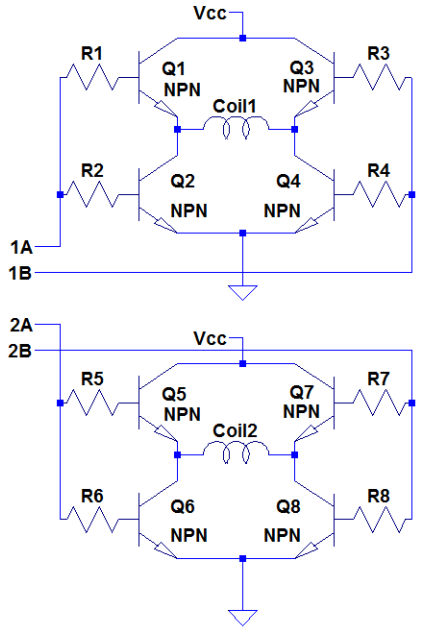


Figure 9 - Full H-bridge to control the stepper motor

During our research we found out that the L293D dual H-bridge chip had two build in H-bridges. Texas Instruments is one of the companies that produced this chip. According to their datasheet it had a wide voltage supply range from 4.5V to 36V. It was a good option since it gave us ability to use it with motors that required a different level of voltage supply. Another option was to use an output current that could be up to 600mA which might not allow us to use full range of permitted current level for stepper motor. Presence of thermal protection in the chip protected the controller from overheating was considered by our group as a big advantage compared to building a controller from scratch since we could accidently burn one of the transistors during fine-tuning the controller.

After we checked the same type of chip from the other companies we found out that they all had the same basic construction and data specifications. For example the chip from the company STMicroelectronics L239D had a supply voltage 4.5V to 36V and also had the same output current 600mA as a chip from Texas Instruments.

The next chip we were researching was L298 Dual H-bridge. This chip had the same basic structure compared to the L239 the main difference was larger power dissipation. It had an operating supply voltage up to 46V and output current per channel went up to 2A which was double the current from L239. Higher current level allowed us to use stronger stepper motors or have higher torque at the same speed.

Our next step was to find a motor controller based on L298 chip. After we did our research, we found three main motor driver opponents based on L298. The first motor driver we researched was ROB-09670 Motor Driver 2A Dual L298 H-Bridge. It had four directional LED's, eight Schottky EMF-protection diodes. The motor driver required a 6V

to 35V voltage supply and had up to 2A output current per channel. An important feature was the presence of the heat sink due to high operating temperature of the L298. Another convenient option was the presence of 5V voltage regulator with power output which allowed us to supply additional logic component from this driver. One disadvantage of this driver was the price of $34. Since we needed two of those drivers total price will be up to $70 which was too high for our budget.

The next opponent was Solarbotics L298 Compact Motor Driver from solarbotics.com. This motor driver had very similar characteristics. The price made a huge difference between two of them. The motor driver from Solarbotics cost only $18.95, this would save us about $30 if we bought two of those motor drivers. But a big disadvantage of this driver was that it did not have heat sink attached to the chip. This could lead to overheating problem during motor driver operation. One of the ways to avoid overheating was to buy a heat sink separately and attach it to the motor driver but it would create additional expense and extra work. We chose to continue our research for a better motor driver.

Last opponent which was based on the L298 was L298N Dual h-Bridge DC Stepper Motor Controller Module for Arduino from oddwires.com. The motor driver from this company had the same parameters as two other motor drivers. But it combined the two best futures from the other two motor drivers. It had a built in heat sink which prevented the motor driver from overheating and cost $12.95 which was the lowest price compared to its opponents.

Bipolar stepper motor had 4 wires which needed to be connected to the motor driver. In the Figure 10 wires from a microcontroller were also required to be connected to the motor driver so we could send sequential signals individually to every wire for a stepper motor. Two bipolar motors required 8 I/O ports from the microcontroller. Since we have many other sensors to be connected to the microcontroller we were looking for a better solution to reduced amount of I/O ports required to connect microcontrollers and motor controller. After we did our research we found circuit created by Sebastian Gassner depicted by Figure 10 which was based on the idea that during every step sequence two wires always have opposite polarities.

This allowed us to create a circuit that included extra switches. Those switches switched polarities of the two wires during every step sequence and it reduced the required amount of I/O ports from a microcontroller for our project from 8 to 4.

The two wires to control stepper motor were definitely a big advantage, however the usage of a motor driver based on L298 chip allowed us to use a stepper motor only in full step mode. In Figure 11, we illustrated a full step operation.

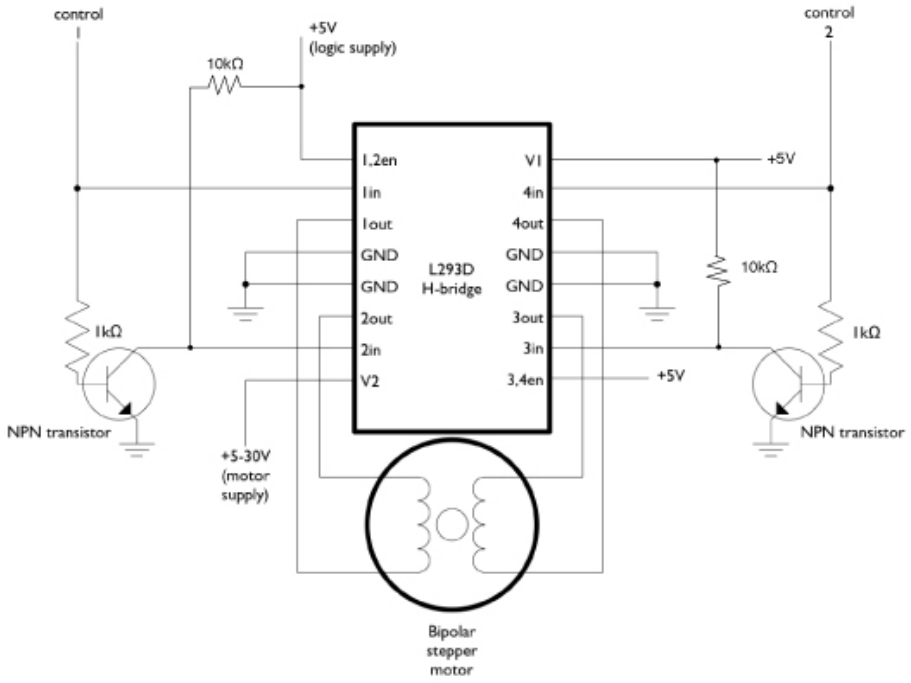


Figure 10 - Two wire bipolar stepper motor connection

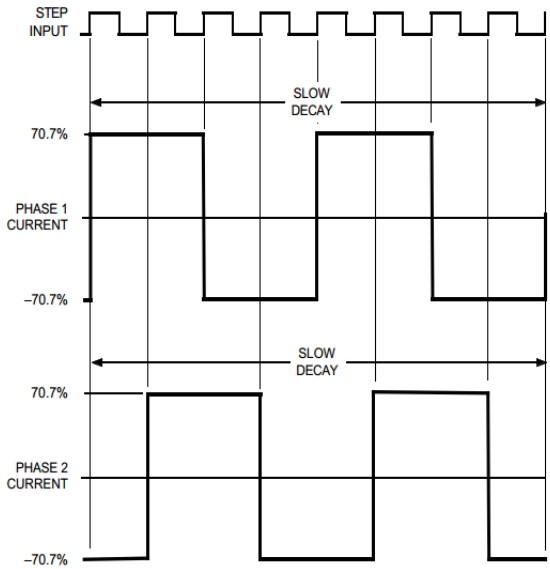


Figure 11 - Full step operation

A stepper motor operated in full step mode during high speed created certain drawbacks including resonance and vibration. The controller sent a high impulse to one of the coils of the stepper motor and made rotor move one step. Then the rotor would stop and wait for the next impulse to move to the next step. But the rotor couldn't stop

immediately and would move back and forth before it stops. Such motor behavior resulted in vibration. Resonance was created when the input motor frequencies from the motor controller matched the natural frequencies of the motor. As a result those drawbacks created extra heat in a stepper motor, and made the motor skip steps during high speed as well as create noise while the motor was in operational mode. During our research, our group looked for a way to eliminate such negative phenomenon of the stepper motor movement. One of the ways was to increase the load on motors to reduce vibration; as a result the rotor had less extra energy to move back and forth. Another more sufficient way to reduce the negative consequences of stepper motor operation was to use microstepping. Eight step microsteps are shown in Figure 12.

Usage of micro-steps allowed us to have better positioning in our movement system. Due to the fact that the rotor received impulses with a sequence shape much closer to a sin wave compared to the full step mode. It created a smooth transition from one coil to another. The rotor of the step motor did not make jumps and as a result reduced significantly the noise level and vibration of the motor. This increased accuracy of our positioning system was an important factor for our project. Due to this fact we decided to search for a motor driver that had a micro-step feature incorporated into the motor driver.

The first simple stepper motor driver we found was EasyDriver Stepper Motor Driver from the sparkfun.com. This stepper motor driver was capable of driving one 4 wire bipolar stepper motor or a 6/8 wired unipolar stepper motor connected as a bipolar. It had a micro-stepping option in which it could drive motor in full, half, quarter and eighth micro-steps. It was based on A3967 chip from Allegro MicroSystems, Inc. and required power supply from 7V to 30V. Finally this stepper motor driver could supply from 150mA to 750mA per phase.

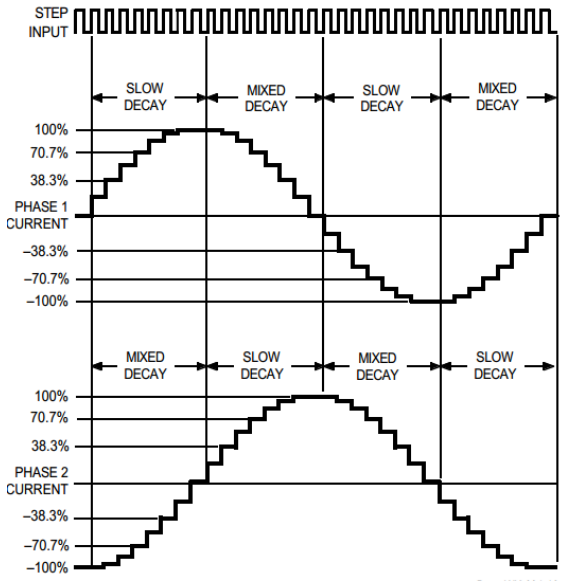


Figure 12 - The 8 microstep operation

The second stepper motor driver we found during our research was Big Easy Driver from the same company sparkfun.com as our first driver. This driver was the latest version of the EasyDriver and it could be used for stepper motors that required more power. Motor drivers became very hot while they were operating and using a driver which is suitable for higher operating current allowed us to operate at lower temperatures compared to the driver with lower operating currents. Current control for this motor driver was up to 2A per phase which was twice as big as a EasyDriver. It could also drive at 16th micro-step operating mode and was based on newer chip A4983. Even though the price of this motor driver was higher, availability of it features makes Big Easy Driver preferable to EasyDriver based on our requirements for the stepper motor driver.

The final motor driver we were considering during our research was A4988 Stepper Driver Carrier with Voltage Regulators from the website pololu.com. This motor driver was based on the newest chip, the A4988. Characteristics of this driver were almost the same as the Big Easy Driver. It operated from 8V to 35V and could deliver current up to 2A per coil. It also had voltage regulators which allowed us to supply any logic device which requires 3.3V or 5V power supply. There were still some disadvantages of this motor driver compared to the Big Easy Driver. Firstly, it did not have an open source schematic like the Big Easy Driver. Secondly, it did not have mounting holes. However it did have same advantages compared to the Big Easy Driver. One of the advantages was the size of the board was smaller. Another advantage was the price A4988 Stepper Driver Carrier cost $19.95 compared to Big Easy Driver which cost $22.95. In Table 14, we compared all three stepper motor drivers together.

Table 14 - Compares stepper motor drivers

	Easy Driver	Big Easy Driver	A4988 Stepper Motor Driver
Power supply range, V	From 7 to 30	From 8 to 35	From 8 to 35
Output power, V	3.3/5	3.3/5	3.3/5
Microsteps	1/2; 1/4; 1/8	1/2; 1/4; 1/8; 1/16	1/2; 1/4; 1/8; 1/16
Chip	A3967	A4983	A4988
Current control per phase, mA	Up to 750	Up to 2000	Up to 2000
Board size, in	0.6 x 1.9	1.25 x 1.75	0.7 x 1.4
Source	Open	Open	n/a
Price, $	14.95	22.95	19.95

Based on the information from Table 14, we made our decision of what stepper motor driver we are going to use in our project.

3.1.4 AUDIO SYSTEM

When the project Deep RGB was first coming together the idea was tossed around that the player should be able to listen to music while playing. This system would not only play music it would be used to send alert sounds for different situations such as check and illegal moves. The system would also allow the user to set preferences as to what music they liked and the board would play the preferences when the user logged in. Another idea for a feature that arose was the ability for a user to plug in their IPod, iphone or other listening device and play music directly from there.

Since the MCU was more than likely not capable of mp3 decoding a peripheral device had to be used to make the audio system possible. The MCU that was going to be used was an Arduino Mega 2560 a common MCU used by hobbyists for a wide variety of applications including sound control. In fact the Mega 2560 was so popular for this application that a special audio version of it had been created to make a powerful controller for many audio applications. Since the MCU being used did not have an mp3 decoder on it a shield must be used to convert the information into sound. This shield took data from the MCU and decoded mp3 files outputting the sound through various I/Os usually a 3.5mm headphone jack at the very least. The audio shield would not be able to play the sounds on its own and therefore had to be connected to powered standalone speakers or to an amplifier that would then drive speakers. If the project were better funded integrated speakers and a small amplifier would have been used for the best results.

In order to play music directly from an external device like an IPod an input like a 3.5mm jack would need to be used and the MCU would need to be able to switch the input to the speakers from the audio shield to the input port. Had Deep RGB been produced for commercial use this feature could be scaled to include all manner of input ports such as USB, serial connections, SD cards and even blue tooth. For the purposes of this project a simple 3.5mm jack would suffice for the input as to keep it simple. Even so this feature was not of the utmost importance and therefore would not make it into the final product if time or money ran short.

The audio shield had to be able to receive interrupts from the MCU so that it could stop itself in the middle of a sound playback to play an alert sound. If the auxiliary input was included in the final project then the MCU would have to switch the connection to the speakers from the external device back over the mp3 shield then if necessary return the connection to the external device.

3.1.5 LED SYSTEM

To begin designing the LED subsystem that will be in our chessboard we had to decide what type of LED to use. The decision boiled down to two different options, single color

LEDs or RGB LEDs. The other variables such as mounting, size and power costs all hinged on this first decision.

The benefits of a single color LED were that it had only one input and output; it could be turned on with a single bit of data. However there were some downsides to it such as the fact that it had only one color limits the information that it could give out. It was always the same color for every application it was used for; this made it a bit less desirable for a chessboard where multiple lines of information needed to be presented to the player. Such information as moves available, check indications and move errors.

The benefits of an RGB LED included the fact that it could be changed variably to include a wide range of colors. This was a useful asset in a chessboard since each type of information could have its own color assigned to it. When each type of information broadcast to the player was color coded it created the opportunity to make a hierarchy of information sent. For example if a player was in check it could show up as a red line between the piece that has them in check and their king, while moves available to the piece being raised could show up as blue. The downside to having this ability are the extra programming that would go into it, the program will have to run through several more operations to calculate the LEDS that needed to be lit and light them in the right color. The other downside was that RGB LEDs require three inputs which made it difficult to wire to a small board.

After considering the pros and cons of both types of LEDs the group decided that the RGB was the way to go. It would make the board look that much more appealing to the user and would add an extra bit of flare to set Deep RGB apart from the competition. As an added bonus this decision was what gave the project its catchy name.

This decision led to a few new challenges and questions that needed to be answered. To start off what type of RGB would be used surface mount pin mount etc. Also how would they be powered and would the power be variable to reach the maximum potential of the LEDs or would it be fixed. Finally how would these LEDs be controlled would there need to be multiple boards or just one.

To start off there had to be a decision made on the type of LED used. Pin mounts were inherently bigger than surface mounts which could be a problem with applications that need to be smaller. However they are easier to use than surface mounted units since they can be routed in many ways and did not necessarily need to be connected to a PCB.

Surface mounted components were neater and provided a more refined look to a project. They were also thinner which could be a benefit when trying to move objects with magnets through a space. However they had to be connected directly to a board to use and that created a whole new challenge of creating a PCB.

In the end the decision was made to use through-hole components. The issue wasn't aesthetic since the components will be hidden, it all boiled down to cost. The vastness

of project Deep RGB would have made it very costly to create a PCB that would contain all the components necessary to run the LEDs. The larger component size was deemed irrelevant since the overall size of an LED is about 5mm and the magnets being used would have no problem reaching through that amount of space. As stated this decision was made primarily on a fiscal basis and if finances permitted a PCB could be used to make a cleaner product.

RGB LEDs had three inputs which could be made variable to create multiple colors. The three inputs typically had different max voltages as well; usually the voltage controlling red was about half the max voltage as green and blue. There were two possible ways to treat the LED, as a variable component or as a fixed component. Thus there had to be a decision as to whether the LEDs should change color or simply be turned on or off.

The RGB LEDs did not have to be completely variable to work in Deep RGB. There were only three possible uses for the LEDs; movement indication, check calls, and errors. Thus the LEDs could be fed three voltages and by turning each one off and on seven colors could be made. This was sufficient for a chessboard but lacked the prestige of having a chessboard that can display a greater variety of colors.

The problem with making an LED change its color was that it was in fact impossible with a regular output from a MCU. Since the MCU was digital and the LED input needed to be analog this created a problem. There was a rather elegant solution to this problem called pulse width modulation or PWM. To do this a PWM output had to be used from the MCU so that it could change the duty cycle of the inputs of the LEDs. Consideration had to be made for the clock speed of the MCU that would be used due to the fact that PWM worked better when the controller was able to change the output values quicker. The faster the controller could change the outputs the more control it had over the duty cycle and thus it could create more colors than a slower MCU.

After considering both ways of powering the LEDs it was decided that Deep RGB deserved to be a bit flashier and therefore it should have variable colors. Some alternative applications of that ability that had been discussed but not officially added were the ability for players to choose their own colors so as to diverge from the ordinary black and white. Another possible use of the ability was to create a randomizer that would change the colors that were displayed when showing possible moves so as to liven the board up a little.

It should be noted that since the decision had been made to use RGB LEDs the power consumption of the board was greater. An average LED had a max forward current of thirty milliamps and the RGB essentially had three LEDs in it so its max power consumption was ninety milliamps. This was accounted for when deciding on a power source which was covered in the corresponding section of this report. To gain a better grasp on the power used by the LEDs a chart was provided below the values were taken from a blue LED (since blue requires more power) model YSL-R1047B5D-D2 and an RGB model YSL-R596CR3G4B5C-C10. Both LEDS were made by the China Young Sun LED

Technology Corporation to help ensure that both LEDS being compared are similar in both quality of manufacturing and materials used. Looking at Table 15 it became apparent how much of a difference in power consumption there was when an RGB LED was used over a single color LED.

Table 15 - Power Consumption of LED types

LED	V1	A1	V2	A2	V3	A3	Watts
Monochromatic	3.4V	30mA	N/A	N/A	N/A	N/A	102mW
RGB	3.4V	30mA	3.4V	30mA	2.2V	30mA	270mW

This was not a great amount of power but the cumulative effects of multiple LEDs could be significant as shown in Table 16 below. Another issue was that the MCU could not supply this much power to the LEDs so one of two things had to happen. The LEDs could be multiplexed so that only one was turned on at a time and the display was quickly written over and over at such a rate that the human eye cannot notice. The other solution would be to include transistors to amplify the power from the MCU and thus power all the LEDs at once.

Table 16 - Example Power Ratings for all LEDs in Project

	Power Per Unit	Total Power For Project
Monochromatic LED	.102W	6.528W
RGB LED	.270W	17.28W
Difference (RGB- Mono.)	.168W	10.752W

One of the more challenging problems of creating an 8X8 array of RGB LEDs was how to control them all. Since each LED has 4 inputs that turns comes to 256 inputs in all. This was a problem since the average MCU did not have that many output pins. This could be remedied by using multiple MCUs but that would have raised the cost of the project greatly and frankly was a rather inelegant solution to the problem. The best way to do this was to use shift registers to reduce the code to either three or four digital outputs.

To control the display with three bits of code from the MCU four shift registers were required. There was one register each for red, green, and blue as well as one for the anodes. This means it required a total of thirty two bits of information to light the entire LED display. The actual design was covered in the hardware section of this report. Using these shift registers the information could be passed down each one from a single output and stored in the appropriate place. The other two outputs needed from the MCU were the shift clock and the store clock. The shift clock controlled when the information was shifted down while the store clock controlled when the information was stored and thus outputted. There was one flaw with this design, the PWM was controlled by the store clock which meant every time the display refreshed, about fifty times a second, the registers were told to store the information again. This cut down on the need to use an enable display bit but created the risk of having data errors, if the data output inadvertently changes so will the display.

There was a way to subvert this problem by creating another output that would control the LED display. This output would be an output-enable which like the name sounds enabled the output and thus power the LED display. PWM was controlled via this output which would turn the display on and off fifty times a second creating the illusion that the LEDs were on continuously. The difference however was that it was not telling the registers to collect new data so if the data input should become corrupted in any way it would not affect the current display.

3.1.6 LCD DISPLAY SYSTEM

Another subsystem researched for our project was LCD displays to satisfy our need for conveying data to the user. The basic purpose of the LCD in our chess board was to supply the user with all the possible information he or she needed. This information included displaying user name and password during the connection to the internet and login to the server. It could also display connection status such as if the system was connected to the server or not. We could also display information about previous or current moves of chess pieces. Last but not the least the display could show the user the current status of the game being played. Criteria for choosing the right LCD included the cost of the unit, the ease of use and how easy it was to set up.

One of the ways to display information was to use LED arrays with different colors. Each color could inform the user about the process going on in the system. For example, red color could represent that the chess board system was connected to a power supply; the light could stay on if there was power or off if there is no power. To show the connection of the board to the server we could have used a green colored LED. This LED would stay blank when the board was not connected, flash with the constant time frame when chess board was connecting to a server and stay green when the board had established connection with the server. We could have used blue color LED to inform the user about different errors in the chess board by using different flashing intervals for each separate error.

The simplicity of an LED display system was a big advantage and big disadvantage of this system at the same time. The advantages were that this display required low technical level and short time to setup; this display also had the lowest cost level. On the other hand, it would have lacked the ability to show some significant information to the user such as password, user name or displaying different game information and game status. To summarize the LED system display we used Table 17.

Table 17 - Advantages and disadvantages of LED system displays

Advantages	Disadvantages
Low cost	Lack of ability to display characters
Low power consumption	Need to study first to be able to understand error massages
Easy to implement	

High level of brightness	
Reliability	

Based on this information, the display we needed be able to show the system status and display some text information about game status. The next option for the displaying text characters was an LCD display. Our research showed that based on the complexity of the crystal arrangement of in LCDs; there were three main types of LCDs to be considered. Each feature made them different from each other and enabled them serve different applications.

The first type of LCD considered was a segment LCD. After we researched this type of display we found that it had a few key disadvantages when compared to other types of LCDs. Since each character was implemented by using 7 to 14 or 16 segments, which created a limitation in displaying all information our user would need. Some of them were used only to image numerical information which was definitely not enough to meet our requirements. The other could display numerical, alphabetical and symbolic information but those displays could only a predetermined and limited amount of symbolic information for example the display in a thermostat for air condition. There was the possibility of ordering a customized segment LCD but it would have increased the cost of the display significantly. Since our project we had certain budget limitations the cost of a customized display was too expensive for our project. The advantage of a segmented uncustomized display was easy implementation in the system, low power consumption and the lowest cost of the different types of LCD displays. We collected the characteristics of segment LCDs in Table 18.

Table 18 - Advantages and disadvantages of segmented LCD displays

Advantages	Disadvantages
Simple programming process	Lack of characters display ability
Low cost	Small amount of symbols per line
Low power consumption	Lack of preinstalled controller
Reliability	
Easy to integrate	

The next type of LCD we researched was a dot matrix LCD. The versatility of this type of display was great compared to segment LCD because dot matrix LCDs used blocks of 5 by 7 dots to create a character which could be a number, an alphabetic letter or any symbol that could be drawn in that small of a space with the dots. Dot matrix displays also had built in controllers, in most of the cases it was the Hitachi HD44780. Wide usage of this controller in different projects allowed us to find all necessary information on how to implement the display in the Deep RGB project. This gave an advantage to the display when compared to the other options. Even though there was plenty information on how to setup other types of displays, usage of dot matrix display would integrate better within Deep RGB.

The power supply for most of dot matrix displays was regularly 5V. This beneficial aspect made it very convenient to implement dot matrix displays together with a microcontroller because they both ran on the same voltage level. We also did not plan to use Deep RGB in any harsh environments; the board we were creating would be operated only indoors. However taking Deep RGB to play outside wouldn't harm a dot matrix display since they had an operating temperature range from 0 to 50C. Another useful feature for us that could be found in dot matrix displays was backlights, which increased readability of display in harsh light environment such as bright light or when there was not enough light to read display. We posted in Table 19 the advantages and disadvantages of dot matrix LCD displays

A graphic LCD was another option for Deep RGB project. This type of display had significantly increased connection complexity compared to the previous types of displays we considered. It consisted of literally thousands of pixels which were used to create images in monochrome or color. This type of display was typically used in high tech projects which required high resolution graphics and images. It was very popular and as a result, graphic LCD displays had a lot of available information on how to incorporate them with most popular microcontrollers. However the cost of graphic displays was significantly higher compared to the dot matrix displays even if they were the same physical size. There was also a big difference in power consumption for graphic display compared to character based displays. Since graphic display had more components than character based LCDs it used more power than alphanumeric displays. Just like alphanumerical displays, graphic displays had a backlight; most of graphic displays needed a 5V power supply which also made them easy to use with microcontrollers. Operating temperature for most graphic displays was similar to operating temperature of alphanumerical displays ranging from 0 to 5C. Information about graphic displays was summarized in Table 20.

Table 19 - 3.2.8.3 Advantages and disadvantages of dot matrix LCD displays

Advantages	Disadvantages
Low power consumption	Unable to use for graphics
Low cost	Limitation on dot matrix size
Integrated controller	Display size limitation
Preinstalled characters	
Ability to create own characters	
Wide variety of sizes and colors	
Easy to find information on how to incorporate display with microcontroller	
Backlights	

Table 20 - Advantages and disadvantages of graphic LCD display

Advantages	Disadvantages

Ability to display graphics and images	High cost
Wide variety of different sizes	High power consumption
Reliable	Hard to program
Backlight	

We didn't even consider touch screen types of LCDs since usage of such display would unnecessarily increase complexity of incorporating the display in Deep RGB project. Because the LCD was a small addition to the main project concept, it was decided that it should have been as simple to implement as possible and did not need many extra features. The cost of this display was also much higher than the amount we allocated for the display in our budget.

After we reviewed all of the information collected on the different types of display systems we decided to use a dot matrix LCD in the project. The dot matrixes were not the latest and greatest in LCD technology but they were capable of displaying a simple message which suited our needs. Even though character-based modules had certain limitations they could still be found in wide variety of lengths and widths. Standard line sizes were 8, 16, 20, 24, 32 and 40 characters with 1, 2, or 4 lines. Simpler one line displays wouldn't be enough for our project because the minimum requirement for Deep RGB display dictated that it had to display the previous line as well. As a result for our project we needed to have at least a 2 line LCD with a minimum of 16 characters long due to the size of the information we needed to display.

The first part we picked for our display was a 16 x 2 Character LCD (Parallel Interface) from pololu.com. It was a simple two line display with 5V power supply which used a HD44780 interface. One negative side of this display was that it didn't have a backlight which could have been a big issue for us since we had RGB backlighting on a chess board and users might try to play in a dark which could make the display uncomfortable to use. The price for this LCD display was a low $9.95. The same size and type display with a backlight cost a couple dollars more (total price $12.95) but it gave us the ability to fully use it in the dark. The supply current for an LCD without backlight was also only 2mA while versions that included a backlight only needed an extra 120mA in order to operate.

Next we chose the LCD16 x 2BL-3V3 from futurlec.com. This was a regular 16 x 2 LCD display with a backlight. It had a built in controller, low 3.3 V power supply and 1/16 duty cycle. The price for this LCD was $8.90 almost 30% cheaper than our previous choice. The size of display, 16 x 2, met our minimum requirements for our LCD module which was more desirable than a larger display since it kept the cost of the system low. The power supply for the display was also 3.3V which was different from the 5V power supply used by most popular microcontrollers.

Our third choice was the BLUELCD20x4BL a 20 x 4 character display which made the display more than double size of the two previous choices. This display looked more modern since it had blue colored characters. It also included a backlight and had a 5V

power supply. Cost of this display was twice that of our second choice and it cost $16.90.

Finally after doing more research and having discussions the team decided to use a smaller LCD which would include the same blue color as a previous 20 by 4 characters display, had a lower cost, backlight and needed 5V power supply so we could easily incorporate it with our microcontroller. It was the BLUELCD16x2BL 16 characters per line display with two lines. The color of this display was blue and it had a backlight. It also needed 5 V from the power supply. Finally its operating temperature ranged from 0C to +50C which was completely suitable for indoor environment.

3.1.7 POWER SUPPLY SYSTEM

One of the last phases of our research was dedicated to the power supply module. Our team saw many available solutions and options offered on the market. The goal of the power supply research was to determine the best ratio of quality, reliability and price. During our power supply research, we identified two main avenues that needed to be investigated; one was the different ways to supply power to the board and the other was distributing energy between different board components.

In the first part we considered different options to supply power to our chess board. One of the options was to supply power to the board through batteries; the ability to use batteries had certain advantages. It would allow us to use the board in any place where there was not a wall socket available. Two types of batteries were considered; NiMH (Nickel Metal Hydride) and Lithium (Li-ion). These two types of batteries were mostly used in small and medium size projects.

NiMH batteries were popular a couple of years before Deep RGB and because this technology had just passed its mature stage in its lifespan, we knew very well the benefits and the shortcomings of them. One of the advantages of the NiMH batteries was that you could recharge them as many times as you wanted. Their price compared to Lithium batteries was also cheaper and it was possible to get one AA size battery supplying 1.25V and storing 2500 mAh of energy for around $2. The disadvantages of NiMH batteries were that they had a high rate of discharge while they were not in use and it could take them about 10 hours to charge depending on different conditions which would create a limitation on the time the chess board could be used powered solely by those batteries. The website www.pololu.com was found to sell NiMH batteries; those batteries had 1.2V voltage output, 2200 mAh capacity current rate. For our project we needed at least 10 of those batteries to create 12V power output for our stepper motors. The rest of information about that battery we entered in the Table 21.

Lithium-ion batteries became more popular for use in medium size projects compared to NiMH in the years shortly before Deep RGB. This happened because they had great advantages over the NiMH batteries. One of the Lithium-ion batteries advantages was that they had greater power density. Another advantage was that lithium-ion batteries

could store the same amount of energy as a NiMH battery while weighing 20%-30% less. Even though the price of the lithium-ion batteries was twice as much as NiMH batteries, the ability to use a battery that was much lighter was considered by our team as a big plus for lithium-ion batteries. They were also environmentally friendly since they did not use toxic materials that NiMH batteries contain. One of the lithium-ion batteries researched was the Polymer Lithium-ion Battery from the web site sparkfun.com. This battery had a current capacity of 2000 mAh and it had a nominal output of 3.7V. To create the minimum 12V output we needed to have at least 3 of those batteries. The rest of the information we put in the comparable Table 21.

Table 21 - Compare of NiMH and Polymer Lithium

	NiMH	Polymer Lithium
Nominal capacity, mAh	2200	2000
Nominal voltage, V	1.2	3.7
Minimum amount of batteries to supply 12V	10	3
Weight of each battery, oz	0.97	1.27
Total batteries weight, oz	97	3.81
Price for each battery, $	2.15	16.95
Total price, $	21.5	50.85

From Table 21, we could see that the usage of NiMH batteries gave us more power and was about half as expensive as using lithium-ion batteries. On the other hand using lithium-ion batteries added much less weight to our chess board compared to NiMH batteries which would have made the board heavy and therefore less. Overall, using batteries in our design could have created an extra degree of portability for our chess board project. However our first prototype would not be very portable and the incorporation of batteries in our project would have added extra weight to the chess board and also take significant amount of time and money from our budget. Due to all these facts, we decided to not incorporate batteries in our project and keep this idea as a future upgrade option for the next generation of Deep RGB systems.

The next option to supply power to the chess board was to incorporate a generator in Deep RGB system. The generator would have generated electrical power by transferring mechanical energy from the rotation of pedals to electrical power. We lobbied this idea while trying to improve the quality of time people had while using Deep RGB by allowing them to work out while they played.

During our research on the power generation idea we found a couple of available solutions on the market and also some information on how to create power generating machines by ourselves. One product we found was the Powerplus Gazelle – Pedal powered PowerBank Generator from the web site amazon.com. This generator cost $238.99 and could supply 12V DC electrical power. Since the cost of the generator was significantly greater than what was allowed by our budget we decided to look for DIY solution. After fully researching the idea we realized that the cost of a pedal generator

even if it were self made would still be around $230. It would have also added extra complexity to the implementation of our project. Since this option was not a priority for our project we decided to not incorporate it into our prototype but keep it as a possible addition for the next generations of Deep RGB system.

Our last option to supply power to the chess board was to use a 110V AC wall outlet. This option eliminated most of the problems and trouble from the other two. It did not require extra money from the budget to implement it and did not add extra weight to the Deep RGB system. Since Deep RGB would normally be used indoors it would be easy to connect it to the wall outlet in any room or in the case of outdoor usage near the house the user could use an extension cord.

After we completed the research on power supply options to the chess board our next step was to research different methods of power distribution to the different components in the Deep RGB chess board. For many years creation of power supplies had not undergone many changes. They still consisted of the same major units such as transformers, bridge diodes and voltage regulators. In spite of the apparent simplicity it was very important to create an efficient and reliable power supply and that all elements of the supplying system worked efficiently and for a long time without any hazards or failures.

First we needed to determine the minimum amount of voltage for each component in our system. To do so we created Table 22 were we illustrated minimum required voltage needed to supply each component.

Table 22 - Minimum voltage requirements from the power supply for each element

Element	Voltage DC level, V
Microcontroller	From 7 to 12
RGB led lights matrix	5
Hall effect sensors	8
Display	5
Led backlight for display	5
Stepper motor driver	From 8 to 35
Stepper motor	12
Electromagnet	12
Wi-Fi unit	3.3
SD card reader	3.3
Music unit	3.3

From Table 22 we concluded that we had at least 3 or 4 DC voltage levels. One way to deal with all of those different voltage levels was to buy an AC/DC power adapter so it could transform 110V AC power to 12V AC power then use rectifier to make 12V DC power. After that we could use different voltage regulators to switch to the different DC voltage levels. Another way was to buy a transformer which could transform 110V AC to

24 V AC so we can have higher voltage level for our stepper motors and after that we could convert to DC and lower the power level for the rest Deep RGB components.

There were many AC/DC power adapters available on the market at the time. After we had done our research on them we chose two different power adapters which suited our requirements better than other adapters. The first power adapter we chose was an external power supply from the web site hobbyking.com which could use 110V and 240V power input. Using of such adapter had certain advantages such as 110V/240V adapters had the ability to work in USA and in Europe. Another advantage was that those adapters had a low price due to their wide usage. The adapter delivered a constant DC 15V at 5A output, which would have been enough to supply our stepper motors, electromagnet and the rest of the electronics. Our team considered having an external power source an advantage for Deep RGB. This would have saved space inside the chess board and it was also safer for our project to keep the power adapter outside of the Deep RGB board in case any failures happened in the adapter it would not damage the board. This adapter also had a low price of $9.49 and it was CE approved which gave us an assurance of quality of this adapter.

An output of 15V from the adapter enabled us to use it directly to supply the stepper motor drivers. To supply the rest of electronics we needed to use voltage regulators to step-down to different voltage levels such as 12V, 5V and 3.3V.

The second power supply adapter we were considering was the Balancer Charger from the web site ebay.com. The price of this adapter with shipping was $8.87, cheaper than our first choice. It had slightly different characteristics than the other adapter as well. The adapter had 110V/240V power input and the output of the adapter was a constant 12V DC and 5A. From one point of view it was preferable for our project to make use of this power adapter since we had to use fewer amounts of voltage regulators. We could directly supply the stepper motor drivers, microcontroller and electromagnet. On the other hand, this adapter had a smaller voltage level and might have supplied less energy to the stepper motors. After discussion our team decided to use the adapter with 12V and 5A output. Due to the fact that we had 5A and if we needed to have a higher voltage than 12V to supply stepper motor driver we could use a step-up voltage regulator to increase the voltage level to the required amount. Also the adapter had a DC male output connector which was 5.5 by 2.5mm and could be easily plugged into 5.5mm female connector the was available to mount on the PCB board.

The input voltage from the wall outlet to our external power adapter would be 110V 60Hz which was converted to 12V DC at 5A. After we did some current limitation, we could supply it to the stepper motor driver, microcontroller and electromagnet. Next we decided that the initial output of 12V had to be dropped to 5V and 3.3V. One 5V DC voltage regulator needed to be used. The first 5V DC voltage regulator we were considering was the AP150650T5L-U from digikey.com. This regulator was manufactured by Diodes Inc. The input voltage for the regulator was from 4.5V to 22V,

which was suitable for us since we had a 12V input from the adapter. The output was fixed at 5V and 3A current and the operating temperature range was -20C~85C and the highest unit price was $2.98. Another possible regulator we were considering was LM2576T-5.0/NOPB from digikey.com and manufactured by National Semiconductor. The regulator had mostly the same characteristics as the first regulator, the only difference was it had an operating temperature of range -40 C to 125 C and unit price was $2.83. For our project usage of the voltage regulator from National Semiconductor was preferable because it had a higher operating temperature range and we had two stepper motor drivers which created a lot of heat.

When we were choosing 3.3 V voltage regulators, we used the same method as for 5V regulators. Our first choice was the LM2576T-3.3/NOPB from digikey.com and manufactured by National Semiconductors. It allowed for input voltage from 4V to 40V, which was suitable for us since we could apply 5V or 12V. The output voltage was fixed at 3.3V and 3A of current which could be lowered if necessary. The mounting type of this voltage regulator was through-hole and an additional heat sink could be added in case the voltage regulator exceeded its operating temperature range which was from -40 C to 125 C. Unit price for this regulator was $2.83.

Another regulator we were considering to use for the 3.3 V DC voltage regulator was the LM2576T-3.3GOS-ND, from digikey.com and manufactured by ON Semiconductor. This voltage regulator had very similar parameters as the 3.3 V DC regulator from National Semiconductor, but differed when it came to the allowable input voltage which ranged from 7V to 40V which meant we could only use it with a 12 V input. It also had a lower price $2.05 per unit. However, since we were considering using the 3.3 V DC voltage regulator only with 12V input we decided to go with this choice and save some money.

3.1.8 WIRELESS DATA TRANSMISSION SYSTEM

3.1.8.1 WI-FI

Wi-Fi indicated a device or product that was capable using radio waves in order to transmit and receive data via a computer network. Wi-Fi devices were certified by the non-profit Wi-Fi Alliance and were centered on the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standards set forth in 1985. A Wi-Fi network was similar to any conventional Ethernet network but categorized into a variety of subsections based on the signal's transfer rate and operating frequency. The 802.11 categories are as followed:

- 802.11a had an operating frequency of 5 GHz with an indoor range of approximately 100 feet (depending on antenna size). It was capable of a data transfer rate of 25 Mbps to 54 Mbps by using an orthogonal frequency division multiplexing method that encoded digital data on several carrier frequencies.

- 802.11b had an operating frequency of 2.4 GHz with an indoor range of approximately 100 feet (depending on antenna size). It was capable of a data transfer rate of 6.5 Mbps to 11 Mbps by using Complementary Code Keying (CCK) modulation method. This method allowed for a higher data transfer rate but lowered the overall range due to narrowband interference. This standard seemed to be the most affordable choice, but lacked some long range capabilities.
- 802.11g had an operating frequency of 2.4 GHz with an indoor range of approximately 100 feet (depending on antenna size). It was capable of a data transfer rate of 25 Mbps to 54 Mbps by using an orthogonal frequency division multiplexing method that originated in the 802.11a standard. Since this Standard used the same operating frequency as the 802.11b standard, most devices were capable of accepting both types of wirelss signals.
- 802.11n had an operating frequency of 2.4 GHz and 5 GHz with an indoor range of approximately 160 feet (depending on antenna size). It was capable of a data transfer rate of 54 Mbps to 600 Mbps by combining multiple antennas and using a Spatial Division Multiplexing (SDM) method that combined multiple signals allowing the device to reach high transfer rates.

These standards allowed us to pick from a wide array of Wi-Fi capable devices to implement in DeepRGB. These devices allowed our system to wirelessly connect to the chess algorithm server through a secure and strong connection. Wireless connections were usually secured by a security protocol to ensure that the data wasn't corrupted or stolen by hackers. There were various types of these security protocols and had evolved to provide a virtually undecipherable connection. The security protocols most commonly used in modern wireless routers were WEP, WPA and WPA2 and since most wireless routers had a long cycle lives, the wireless module in DeepRGB needed to be compatible with these types of protocols.

3.1.8.1.1 WIFLY GSX BREAKOUT BOARD

The WiFly GSX Breakout board as shown in Figure 13 offered a radio module that was a complete embedded Wireless Local Area Network (WLAN) device. This device only required 4 pins and could transmit serial data to and from the Arduino microcontroller using its UART interface. The two pins used in UART for serial transmission were the TX and RX pins and could be used to write data to the microcontroller's flash memory.

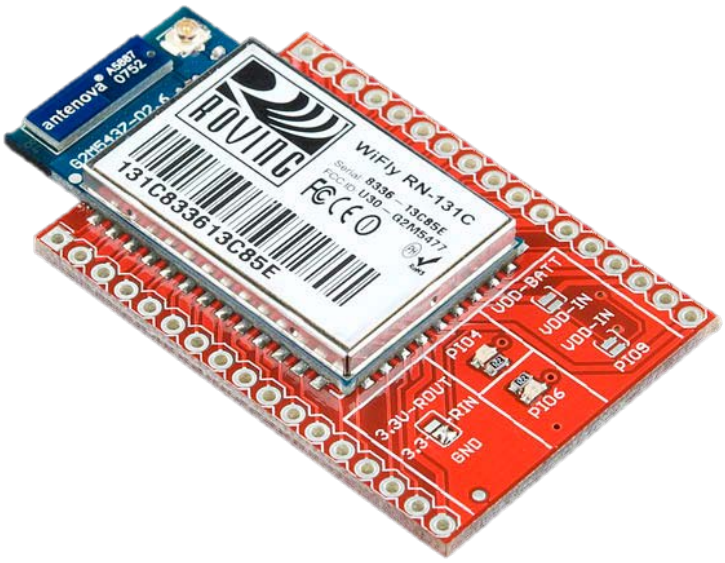


Figure 13 – Sparkfun's WiFly GSX development board reprinted with permission from Sparkfun.

It was capable of connecting to the 802.11b/g standard and accessing the most used types of security protocols. It utilized an embedded Wi-Fi module designed and built by Roving Networks. The Roving Networks RN-131G's features included:

- Qualified 2.4GHz IEEE 802.11b/g transceiver
- High throughput, 1Mbps sustained data rate with TCP/IP and WPA2
- Ultra-low power - 4uA sleep, 40mA Rx, 210mA Tx (max)
- Range: Up to 100m
- On board ceramic chip antenna and U.FL connector for external antenna
- 8 Mbit flash memory and 128 KB RAM
- UART hardware interface
- 10 general purpose digital I/O
- 8 analog sensor interfaces
- Real-time clock for wakeup and time stamping
- Accepts 3.3V regulated or 2-3V battery
- Supports Adhoc connections
- On board ECOS -OS, TCP/IP stacks
- Wi-Fi Alliance certified for WEP-128, WPA-PSK (TKIP), WPA2-PSK (AES)
- FCC / CE/ ICS certified and RoHS compliant.
- Industrial (RN-131G) and commercial
- (RN-131C) grade temperature options
- Price: $84.95

The WiFly GSX Breakout board allowed us to connect to our server from extreme distances. Unlike with the XBee about to be discussed, it didn't require the server to be in the vicinity of DeepRGB.

3.1.8.2 XBEE

Xbee embedded modules delivered wireless end-point connectivity to devices, such as microcontrollers. There were numerous different kinds of Xbee modules that could have provided the necessary connection for the microcontroller unit in DeepRGB, but most of them shared similar pin-outs thus making the modules interchangeable depending on our environment and needs.

All Xbee modules used the IEEE 802.15.4 standard networking protocol and provided both Point-to-Multipoint (P2MP) and Peer-to-Peer (P2P) networking solutions. The modules were designed for application requiring low latency, low power and reasonable transfer rates that utilized Zigbee centered protocol. Zigbee was an 802.11 alternative and followed the IEEE 802.15.4 standard that offered an affordable, power efficient, open wireless mesh networking environment as shown in Figure 14.

This standard was known to be exceptionally power efficient and made it desirable in most embedded systems. The mesh system ensured that data could be transferred to an out of range device by using one or multiple Zigbee radio devices in order to relay the information to the required destination. The data got routed by nodes that received and forwarded data from each other, thus ensuring that the data always found the quickest route to the destination. If a node became unavailable, the protocol used would seek out another route in order to maintain the connectivity between the two or multiple peers needed.

Most modules that incorporated the 802.15.4 standard dictated the use of Direct Sequence Spread Spectrum (DSSS), a modulation procedure where the carrier signal inhabited the full spectrum of the bandwidth. The data signal got multiplied by a noise signal with a much higher frequency than the original data signal. The entire signal then got multiplied once more before being deconstructed at the receiving end. This also provided the module with a built in resistance to jamming, ability to share a single channel with multiple users and fixed any timing issues that may have occurred during transmission

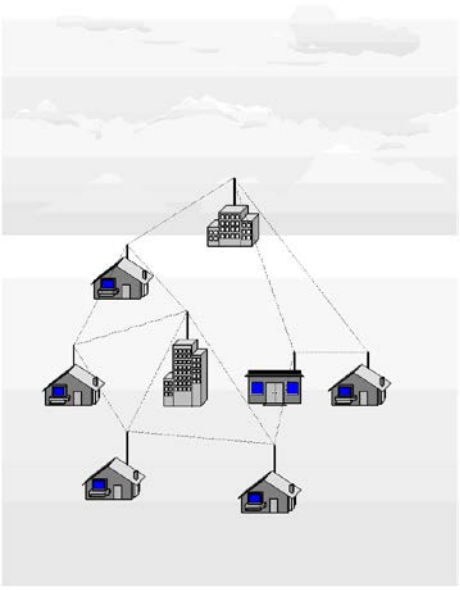


Figure 14 – A representation of an open wireless mesh network reprinted with permission from Wikipedia.

There were numerous Xbee modules categorized by their specifications. The following are just a few examples of the modules that were available on the market during this research:

3.1.8.2.1 XBEE 1MW CHIP ANTENNA - SERIES 1

The Xbee series 1 module was the simplest and easiest to work with devices for a P2P connection (but was also capable of P2MP). They had limited mesh networking capabilities but made up for that with ease of setup and flexibility.

The series 1 was capable of connecting to virtually any device with a serial port and had the following features:

3.3V @ 50mA

250kbps Max data rate

1mW output (+0dBm)

300ft (100m) range

Built-in antenna

Fully FCC certified

6 10-bit ADC input pins

8 digital IO pins

128-bit encryption

Local or over-air configuration

- UART hardware interface
- AT or API command set
- Price = $22.95

As seen in the summary above, the Xbee wasn't quite as fast as a Wi-Fi connection, but its power consumption was a fraction of the amount needed in a Wi-Fi module. Its range surpassed large scale Wi-Fi modules and had a small form factor which was desirable in most embedded systems.

3.1.8.3 BLUETOOTH

Bluetooth used low-power radio waves to transmit data over short distances on a frequency of 2.45 GHz. This frequency spectrum was also known as the Industrial, Scientific and Medical devices (ISM) band. In order to prevent interference with all the other devices within this band, Bluetooth devices tended to send out weak (approximately 1mW) signals. Bluetooth signals didn't require a clear line of sight in order to connect to a device; this caused it to be exceedingly popular in small embedded systems. It had the capability of connecting up to eight different devices all together without causing any interference with one another by a method called Frequency-Hopping Spread Spectrum (FHSS) that quickly swapped the carrier between seventy-nine different frequency channels using a classification recognized by both the receiver and transmitter. This technique also ensured that most Bluetooth devices would not interfere with one another due to the constant switching of carrier frequencies.

3.1.8.3.1 BLUESMIRF SILVER

The BlueSMiRF silver was meant to replace serial cables with a wireless solution in most embedded devices. It used an RN-42 Bluetooth module capable of a 60 foot (18) range from the module while providing low power and fast data transmissions. It was capable of using the Arduino's UART pins in order to access the microcontroller's flash memory. Further specifications included:

FCC Approved Class 2 Bluetooth Radio Modem

Extremely small radio - 0.15x0.6x1.9"

Very robust link both in integrity and transmission distance (18m)

Hardy frequency hopping scheme - operates in harsh RF environments like WiFi, 802.11g, and Zigbee

Encrypted connection

Frequency: 2.4~2.524 GHz

Operating Voltage: 3.3V-6V @ 45mA

Serial communications: 2400-115200bps

Operating Temperature: -40 ~ +70C

Built-in antenna

UART hardware interface

Price: $39.95

The BlueSMiRF was incredibly small compared to the other previously discussed wireless solutions, but it lacked the range needed to keep DeepRGB somewhat mobile. Nonetheless it needed to be considered against the two other options since it was very power efficient and had the capability of decreasing the overall size of the build.

3.1.8.4 WIRELESS MODULE SELECTION

With the numerous ways of wirelessly connecting the microcontroller unit to a server, Table 23 compares the specifications of the summaries discussed in the above sections.

Table 23 – Comparing the three wireless connectivity devices in order to make a final decision.

Device Name	WiFly GSX	XBee Series 1	BlueSMiRf Silver
Data transfer rate	1Mbps	0.24Mbps	0.1Mbps
Range	100m	100m	18m
Required input voltage	3.3V	3.3V	3.3V
Required input current	210mA	50mA	45mA
Microcontroller interface	UART	UART	UART
Price	$84.95	$22.95	$39.95

The BlueSMiRf Silver Bluetooth option could not be used for our project due to its limited range. Using Bluetooth would have required us to have a computer within 18m to provide the necessary chess piece movement updates and that would have ended up being a hassle. Both the XBee Series 1 and WiFly GSX module had the same wireless connectivity range, but the Wi-Fi option was preferable due to its higher data transfer rate and wouldn't have required a computer to be within 100m. Using multiple XBee Series 1 modules to create a mesh for longer range ended up being inefficient, so the choice of using the WiFly GSX as our development solution was made. The RN-131C

wireless module used in the WiFly GSX was to be incorporated into the PCB design when the development process was completed.

3.1.9 PRINTED CIRCUIT BOARD

In order to bring all of the modules together and ensure connectivity between them, a PCB could have been used and was the ideal choice, but had the potential of turning out to be quite expensive. There were many options that allowed us to create our main circuit board such as:

3.1.9.1 STRIPBOARD

A popular and cost effective method of connecting circuits was using a strip board. Strip board was widely used during prototyping and with good soldering techniques provided a professional looking circuit board with all of the components on one board. The name stripboard automatically explained the layout of the material; it was made up out of strips of copper pads that conduct electricity. Most of the devices needed for Deep RGB had through-hole counterparts, except the MCU. Since the MCU used 100 surface mounted pins, a strip board was nearly impossible to implement into our designs.

3.1.9.2 PERFBOARD

Perfboard was a board used to prototype circuit boards. It was very similar to strip board, except it was made up out of small squares of copper instead of strips. These copper pads could be located on both sides of the board in order to utilize both sides for prototyping. But perfboard suffered the same downfall as stripboard; surface mounted components could not be soldered onto their perfboard due to their small size.

3.1.9.3 PCB

Since the previous options had been ruled out due to their size and impracticality, a PCB seemed like the best choice for this project. In order to keep the size of the physical chess board small, a PCB was ideal since it allowed us to use as many surface mounted components as possible. Choosing the option of PCBs also allowed us to gain more knowledge into professional circuit board making. PCBs needed to be designed by using applications such as Eagle Cadsoft, ExpressPCB and DipTrace. Using these applications also allowed us to move around our modules and components in order to achieve the smallest circuit board possible. PCB manufacturers tended to set their prices on size, so having the smallest circuit board not only saved space, but also lowered our overall cost as well. Like perfboard, PCBs had the possibility to utilize both sides of the board. This could make the circuit board even smaller and give the project a more professional looking design.

3.1.10 SERVER SOFTWARE

The first part of designing the software side of the system was in fact choosing the hardware that would run the software. In that particular decision, there were only two options, running the chess engine on-board, using either the single main processor or adding another processor to handle only that, or offloading the chess engine and its related functions entirely to a separate server.

The benefits of running the chess engine on-board included having one singular piece of hardware that contained the entirety of the project, letting us focus on one physical thing and allowing tighter integration of the algorithm with the board itself. The downside of this was that any on-board processor was going to be severely underpowered compared to the highly powerful CPUs in modern PCs and servers, and chess engines are entirely processor limited operations which left any on-board solution very slow at computing moves relative to the speed capable otherwise.

Alternatively, offloading the computationally intense portions of the code to a separate server allowed us to compute deep searches of the chess problem space in a fraction of the time required by an on-board solution. The powerful, multi-cored CPUs in modern computers let the complex, threaded searches by chess engines operate efficiently in a very small amount of time. On the other hand, moving to an off-board solution required the addition of networking hardware to the otherwise self-contained unit, adding this new complexity of the board's main controller while it removed the hurdle of computing chess moves.

The decision was made in a group discussion early in the project. In order to add variety (and a certain "wow" factor) to our project, we would design the system to include connection to an off-board server for the purpose of handling game save states and computer-player chess moves. A Wi-Fi module would be added to the board to enable communication to and from a centrally located server via public or private Wi-Fi networks.

This decision also led to a bundle of new ideas for features since the server architecture allows for easy networking and centralization of information. As mentioned, the early idea of game saves was an easy target for redeployment to the server. In addition, the ease of networking brought up the possibility of multiplayer, which came to add another "wow" factor to the project. The possibility of tying into social networks like Facebook and Twitter was also discussed.

Once the decision was made, real research into the possibilities for a server-based architecture could be started. There were several questions to tackle, including what language to write the server software in, which protocols to use, and, of course, what chess engine to use.

3.1.10.1 SOFTWARE LANGUAGE

There were several options as far as programming language was concerned. Most current, high-level languages were capable of tackling the problem presented, but each would allow a different approach to the problem.

C#.NET - C# was the first choice of language research because of experience and familiarity with the language the group had. C# and Microsoft's .NET environment would have allowed for incredibly easy debugging through the use of their integrated Visual Studio.NET IDE. C# was an obvious research area as it was an immensely powerful language with libraries and extensions already available to handle many tasks that would be required of the software system. Primarily, C# was incredibly easy to get working with the command line chess engines researched. In addition, C# was a personal favorite language of the group's resident Software Engineer, Joe.

As a downside, however, C# was not known for its ability to interact well with non-Microsoft standards. C#'s interactions with MySQL, from Joe's experience, were clunky at best and using C# would have locked the group into using Active Server Pages (ASP) for delivery of web content. In addition, while Visual Studio was not the only IDE capable of compiling and running C# code, it was the most well-known and supported, potentially locking the group's programmer into a single IDE choice without as much flexibility as provided by other languages' IDEs.

PHP - PHP was looked at because of its nature as a web-focused language. Writing the server in primarily PHP would have allowed for a native web experience instead of the second-hand job that would have existed with C# and .NET. PHP was also very open to platforms such as MySQL that allowed the group to get production-level quality without paying for an enterprise license. The number of frameworks available for PHP also left the group with many options as far as design and implementation.

On the other hand, PHP was not as flexible on the command line as C# and was not quite as easy to interface with command line chess engines. As a web-based language, this was not surprising, but the benefits of utilizing the simple web-interfaces in exchange for a more difficult experience on the backend had to be weighed carefully.

Java - Java was usually an acceptable alternative for C# anywhere the latter would perform well, and this fact remained true for this project. All the benefits of C# (in this project's scope) would also apply to Java, but the since Java is slightly slower than C# on Windows machines, there was little reason to use it over C#.

Python - Python was another powerful language choice. It had several of the advantages of both PHP and C#.NET, but a severe downside that would likely prove untenable. Like PHP, Python was incredibly easy to implement on the web front and, like C#, was very powerful on the backend, easily tying into the command line input and output from chess engine programs. The downside was that Joe was not nearly familiar enough with

Python to write the server efficiently without devoting a large amount of time to learning the language first.

Perl - Perl was looked at as an afterthought. As a scripting language, Perl would have been easy to implement both the web front-end and chess engine back end. However, Perl was also limited by the platform that it would be performing on. To maximize the capabilities of the language, it would have to run on a Unix-based system, something that was not available for free to the group. Since every other programming language option had the benefit of working fine (or only, as the case with C#) on the Windows machines that the group had access to, Perl was dropped as a language choice early.

C++ - C++ was looked at as well, as it was an enterprise-level language that had robust and deep support spanning years. C++ was adept at port-based communications and would have actually allowed for direct communication with the few chess engines that provide non-command line interfaces, as many algorithms are themselves written in C++. C++ was considered a solid backup choice for each other language as, with work, C++ was almost limitless in its scope. While other language options provided benefits in certain areas, C++ had no downside that could not be corrected with time spent.

3.1.10.2 CLIENT-SERVER PROTOCOL

There were two primary methods investigated for the communications between the board and the server: direct port communication and web-server based communication.

Direct port communication had the benefit of being easy to work with on the low-level controller in the chess board. It would also have been relatively easy to implement in C++, C#, Perl, and Python. The downside was that, without significant infrastructure time investment, port-to-port communication was generally synchronous, leaving the server able to communicate with only one client at a time and requiring it to close and re-open the connection incredibly often as a result.

Web server based communication would have been easy to set up in almost any programming language that was explored, and that was why it was the primary focus of research for the project. Port-to-port communications were the basis of web-based communication, of course, but good open-source options exist for every platform considered that handle the problems that exist with port-to-port, including multi-threading and queue systems to handle several simultaneous asynchronous connections and always available information.

3.1.10.3 SERVER-ENGINE PROTOCOL

The research into which chess engine began with researching the two primary chess protocols, the Chess Engine Communication Protocol (CECP) and the Universal Chess Interface (UCI), along with the possibility of implementing a chess engine through

original code and therefore using a proprietary, original protocol. The idea of implementing an original chess engine was quickly discarded though as the depth of modern chess engines was found to be such that years could be spent on an algorithm, refining it's AI and search patterns, only to have minor improvements in overall play.

The CECP and UCI both utilized the concept of piping input and output from the engine to whichever device or program was using them. Interfacing with these protocols programmatically therefore required the ability to spawn a new process on the host machine and redirect it's standard input and output to the server process. As this involves direct access to the host machine's operating system, C#, C++, Python, and Perl excel at it while PHP suffers from its singular problem.

The primary difference between the two protocols was that the UCI offloads some duties from the chess engine itself and onto the program that was implementing the interface. As such, chess engines designed to function with the UCI had the opportunity to lack certain features that would be useful to have, such as support for opening books and endgame Table Bases.

3.1.10.4 CHESS ENGINES

There were countless chess engines available for use in DeepRGB, far more than could be researched fully in a reasonable amount of time. For this reason, only cursory research was done on only a handful of engines based on the overall skill level of the engines. This research is detailed in Table 24 - Chess Engines, shown below.

Table 24 - Chess Engines

Engine[1]	Maximum Elo Rating	Interface	Multiple Skill Levels?	Endgame Table Bases?	Opening Books?
Houdini v2.0c	3209	UCI	Yes	Yes	Yes
Houdini v1.5a	3203	UCI	No	Yes	No
Stockfish v2.2.2	3163	UCI	Yes	No	Yes
Rybka 4.1	3162	UCI	Yes	No	No
Critter v1.2	3159	UCI	No	No	Yes
Vitruvius v1.11c	3125	UCI	No	Yes	Yes
Komodo v3	3115	UCI	No	No	Yes

[1] All information was obtained by checking the engine's command line output. UCI interfaces are required to share their capabilities when started.

Deep Junior v13	3040	UCI	Yes	Yes	Yes
Spike v1.4	3033	Both	No	Yes	Yes

To meet requirements CHS04, CHS05, and CHS06, the system had to incorporate all three functionalities listed in Table 24, columns 4, 5, and 6. One engine alone was not required to meet all requirements, only the system as a whole, so it was possible to incorporate the use of multiple engines so as to cover all requirements.

4 DESIGN

4.1 HARDWARE DESIGN

4.1.1 MAIN CONTROL UNIT

The main control unit had to be able to handle all of the input and outputs needed to make DeepRGB functional. It was decided in section 3.2.1.3 that the ideal microcontroller choice would be an Atmel Corporation ATmega 2560. A brief explanation was made below of main components within the microcontroller unit and their respective functions.

4.1.1.1 VOLTAGE REGULATOR

The voltage supplied from the main power regulator was potentially not as stable as we wanted it to be. To ensure the microcontroller unit didn't get damaged in the case of an over-voltaging problem, a second voltage regulator was put in place. Figure 15 shows the circuit for the power supply of the microcontroller unit.

The component labeled IC1 was an adjustable output low dropout voltage regulator. This component ensured that the microcontroller unit only received the necessary 5 volts as stated in section 3.2.1.2.1.

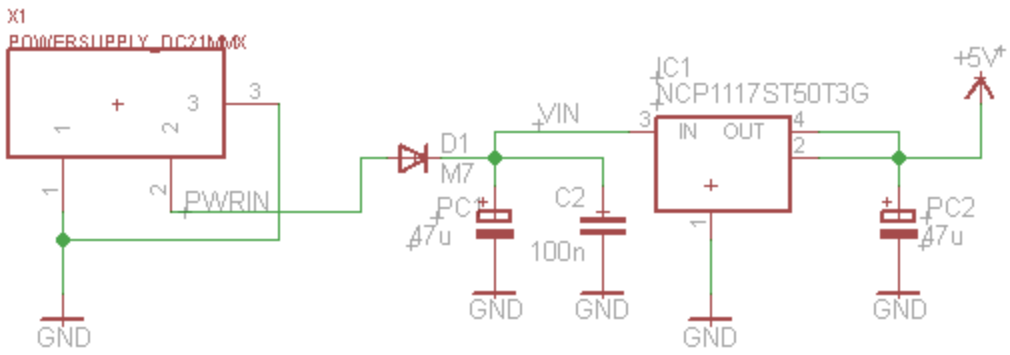


Figure 15 – The voltage regulator needed for the microcontroller reprinted with permission from Arduino.

4.1.1.2 THE ATMEL CORPORATION ATMEGA 2560

The Atmel Corporation ATmega 2560 was the component that received and transmitted data from the other devices nested within DeepRGB and processed this using the programming code in its built in flash memory.

Figure 16 shows the Atmel Corporation ATmega 2560 with each of its pins referenced. The most important pins used for this project were the SPI pins used for the serial bus. SPI was one of the fastest ways the Atmel Corporation ATmega 2560 could communicate with other devices.

The LCD display used four digital I/O pins to receive its data instead of using the serial bus. The wireless data transmission device needed to make use of the UART bus, this bus allowed the wireless device to receive data from the Atmel Corporation ATmega 2560 and transmit it to the server via a web interface. The movement motors needed eight digital pins and one PWM pin. These pins were arbitrarily chosen since they only sent out a high or low signal.

Table 25 below shows a summary of which pins will be assigned to transmit and receive data from every component within the subsystem.

Table 25 – Labeling the pins used for each module

DeepRGB Device	Control pins needed
Liquid Crystal display (LCD)	Register Select = PD7 (T0) Enable = PG0 (WR) Data = PG1 (RD), PC0 (A8), PC1 (A9), PC2 (A10)
Red, Green, Blue Light emitting diode matrix	Serial Clock = PB1 (SCK/PCINT1) Chip select 1 = PC3 (A11) MOSI = PB2 (MOSI/PCINT2)
Hall effect sensor matrix	MUX Select = PA0,PA1,PA2,PA3 Analog Read = PF0, PF1, PF2, PF3, PF4, PF5
Wireless data	RX = PE0 (RXD0/PCINT8)

transmission device	TX = PE1 (TXD0)
XY grid stepper motors	Step and Direction = 26, 28, 36, 38 Micro-step = 30, 32, 34, 40, 42, 44
Electromagnet*	Enabler = PA6 (AD6) PWM = PE3 (OC3A/AIN1)
Audio-Sound Module**	Serial Clock = PB1 (SCK/PCINT1) Chip select 3 = PA2 (AD2) MOSI = PB2 (MOSI/PCINT2) MISO = PB3 (MISO/PCINT3)

*In the final product the electromagnet was replaced with a permanent magnet on a solenoid

**In the final product the Audio-Sound Module was not used

These pins were connected via pathways on the final manufactured PCB. The stepper motor control on the other hand required some headers and long wires to be in place in order to ensure that its range of movement was not hindered in any way.

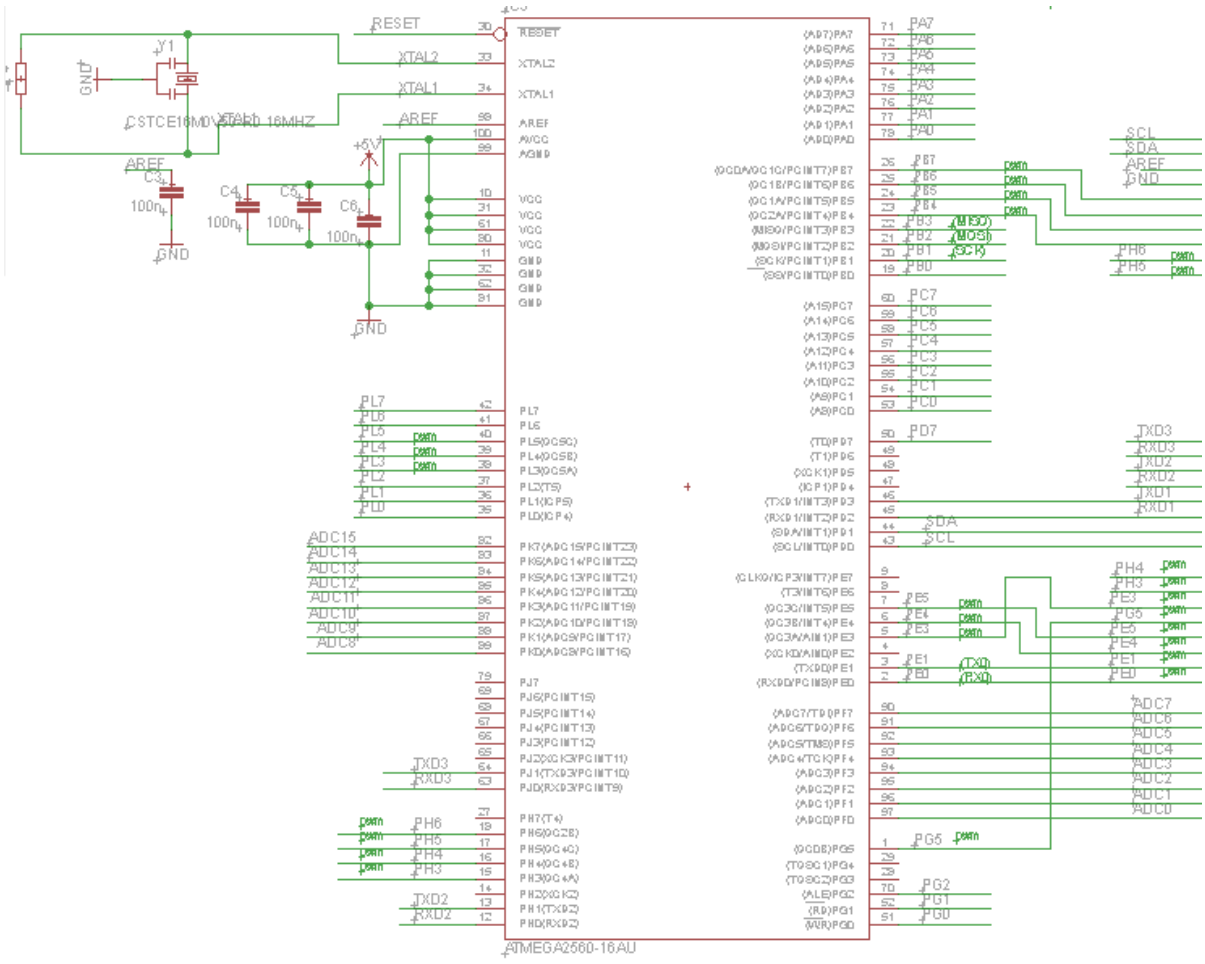


Figure 16 – The ATmega 2560 and its pin labels used for mapping reprinted with permission from Arduino.

4.1.1.3 USB TO SERIAL INTERFACE

The Atmel Corporation ATmega 2560 needed to be able to interface with a computer to receive its programming code. This device was able to connect to USB and transmit data to the flash memory of the microcontroller. The Atmel Corporation ATmega16U2-MU as shown in Figure 17 needed to have a DFU bootloader programmed onto it. The bootloader software was capable of ignoring malformed data and also resetting the Atmel Corporation ATmega 2560 if needed.

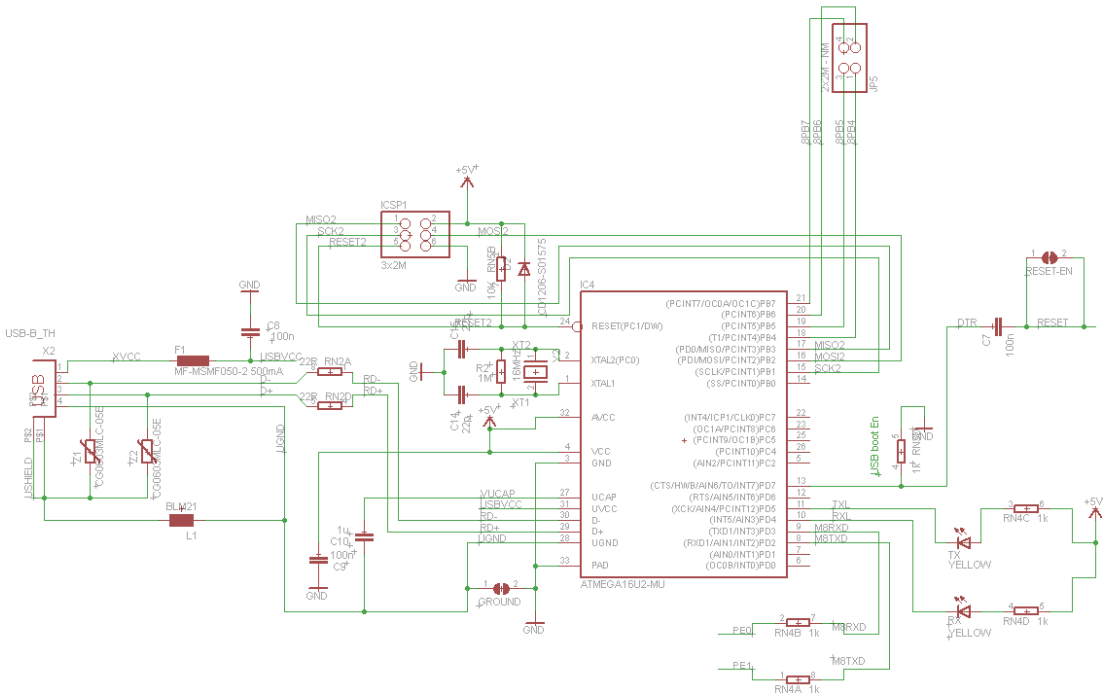


Figure 17 – The USB to serial interface used to upload the programming code reprinted with permission from Arduino.

4.1.2 MAGNETIC PIECE-MOVING SYSTEM

The Design of the magnetic piece-moving system was similar to the design of an XY positioned table. The system had the ability to move a magnetic device used to pull pieces to any required location under the chess board and it was able to be activated and deactivated by the MCU at any time.

To create this system a simple movement device was made using a rack and gear layout using drawer glides to guide the system in a similar fashion to the aluminum bars and

linear bearing setup shown in Figure 18. For the rack and gear system, we used gear tracks for stepper motors to grip and move along the path determined by the MCU. The Y-axis stepper motor was connected in the middle of a wooden bar that was guided on either end by drawer glides. Atop the wooden bar, which was parallel to the X-axis, another glide and gear track were mounted as well as the X-axis motor. In the original design an electromagnet was to be used to move the pieces. However this proved difficult since the electromagnets available were holding magnets and did not cast a magnetic field very far. To solve this problem we replaced the electromagnet with a permanent magnet mounted atop a solenoid. The solenoid was turned on to push the magnet up and hold it there while the piece is being moved then it is turned off to lower the magnet and allow it to pass under the board without affecting the pieces.

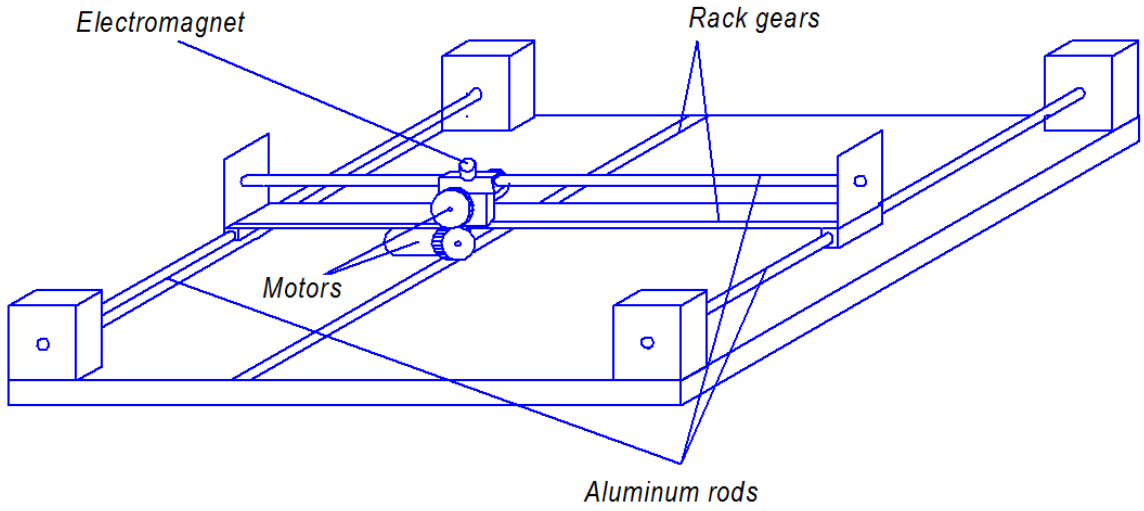


Figure 18 - Basic design of the moving-piece system

To control the solenoid we created a simple circuit where we had one switch which turned it on or off through a command sent by MCU. For the switch we used an NPN-transistor which opened if the pin out from MCU stayed at the high level and closed when the pin went low. We also needed to have a diode to protect the transistor from back voltage when we turned the solenoid off. Finally, we added a couple of capacitors to reduce signal noise from electromagnet in the solenoid. In Figure 19, we see the basic connection from the electromagnet to the MCU.

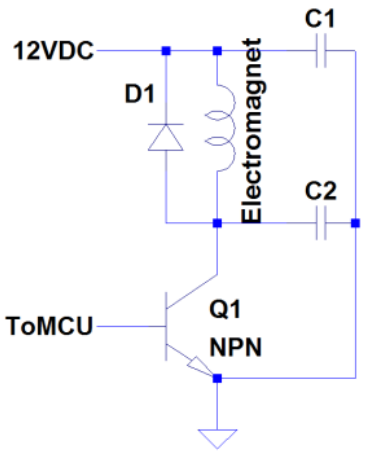


Figure 19 - Connection of the electromagnet to the MCU

The stepper motors we chose were the Mercury ROB-09238 Bipolar Stepper Motor; they had a light weight of 200g, but were still powerful motors with 31.9 oz/in torque. The motor also had the longest wires of the group, which allowed us to comfortably connect the motor to the motor driver without worrying about building a bridge. The stepper motor has a standard diameter shaft size of 5mm which allowed us to connect a cylindrically shaped gear. The cylindrically shaped gears and rack gears are sold at vexrobotics.com.

In order to be able to move pieces accurately we needed to be able to set the stepper motors to a certain position on start-up and after moving a piece. This reduced the error created by the slipping of the motors and allowed us to position pieces more accurately. To accomplish this two switches were attached to the movement system, one on the X-axis and another on the Y-axis. Upon start-up and after moving a piece the positioning system moves in the direction of the origin until both switches are triggered indicating that it was back at the start.

4.1.3 PIECE DETECTION SYSTEM

As discussed in the research section of this report Deep RGB employd a piece detection system powered by Hall Effect sensors. These sensors detected the magnetic fields generated by the magnets placed inside the pieces and send the information to the MCU which then used it to extrapolate the positions of the pieces. This feature allowed the device to reset the board after a game as well as re-center pieces moved by the user and store piece positions including the graveyard. The program which was described in the software section of this report stored the initial piece positions as well as updated the positions when the board moved a piece. The Hall Effect sensors were used to allow the board to determine if a piece had been picked up by sensing a drop in the magnetic field. The board then showed available moves of that piece using the LED subsystem.

Once the piece was moved the board determined where using the sensor grid and then cross reference that information with the position index to make sure it was a legal move as well as to determine if a piece was taken or not. If the move was invalid either by the player placing the piece on a line or in an invalid tile the board then moved the piece back to its original position before the move was made.

In the research section two arrangements of sensors were considered the triangle arrangement and the all corners arrangement. Both of these relied on ratiometric Hall Effect sensors, the difference was the positioning and therefore the potential distance from a magnetic field as well as the possibility of sensing more than one field at a time. Before either method could be applied there must be a decision on what sensor to use. It had already been stated that the sensors should be ratiometric but they must also be able to endure the magnetic forces created by the movement device and the pieces. In the triangular arrangement the sensors were approximately 1.27cm away from the center; the piece would never be farther from a sensor since the tiles were only 5cm squared. Therefore if the triangular arrangement was used the sensors should be able to at the very least detect a piece from 1.27cm away. However if the all corners method was used then the sensors would be 3.5cm away from the center. Once again this is the farthest a piece can be from a sensor so if the corners method was used the sensors needed to be able to detect a piece from that distance.

Another factor that needed to be considered while deciding on the appropriate sensor was the cost of the component. While the cost was not the most important factor it was certainly worth careful consideration, due to the sheer amount of components needed and the potential for cost to get high very quickly. It should be noted that when comparing the sensors below the prices were based on the special pricing for buying components in bulk. Deep RGB required at least 117 sensors, if the corners arrangement was used, and therefore the price shown was the lowest price for which 117 units may be purchased. This helped in the decision by showing the absolute worst case scenario price wise for each sensor being compared. The previously discussed requirements as well as some not mentioned are listed below and were used as the criteria for selecting the correct sensor.

- Cost: Less than $4.00 per unit
- Type: Linear Ratiometric
- Package: Through hole
- Sensitivity: Less than 2mV/G
- Operating range: 5000G preferable, above 3500G acceptable
- VCC(recommended): 5-6V
- VCC(max): Less than 10V
- I(max): Less than 30mA

Using these parameters to find the appropriate Hall Effect sensor was more difficult than apparent at first glance. The problem stemmed from the operating range which in turn relies on the sensitivity. Most Hall Effect sensors available were built for precision

therefore having a higher sensitivity allowing them to detect smaller fields. While many were programmable allowing users to set the sensitivity lower many still had a relatively high sensitivity i.e. 3mV/G and even more in some cases. This in turn limited the operating range, and if they were used in Deep RGB they would max out and possibly become damaged before the pieces were positioned. However there were some promising components available and four stood out for their potential to work in Deep RGB.

4.1.3.1 A1302UA

The first of the four candidates is the A1302 which was a member of the A1300 family produced by Allegro Microsystems Incorporated. This sensor was well known and used frequently by hobbyists for a wide variety of applications. The 1302 came in the same package as the 1301 which was available in both surface mount and through-hole packages. However it had less sensitivity making it the preferred sensor of the 1300 family for potential use in Deep RGB. The specifications for the A1302UA are listed below for ease of reference.

- Cost: $1.54 per unit
- Type: Linear Ratiometric
- Package: Through hole
- Sensitivity: 1.3mV/G(typical), 1.0mV/G(min)
- Operating range: 0-2307G(typical), 0-3000G(max)
- VCC(recommended):6V
- VCC(max): 8V
- I(max): 11mA

According to the specifications above the 1302 could have been used in Deep RGB, however this was not advisable. As the data shows the operating range only reached the requirements when at its absolute maximum under the recommended settings. This was with VCC set to 6V and the sensitivity considered at its minimum value. This however would not work since the 1302 was not programmable and therefore there is no way to ensure that the sensor would be at its minimum, it would be luck of the draw. The potential hazard of using the 1302 was the chance of a sensor turning on when a piece was either too close or even on top of it. This would max out the output of the sensor making it almost useless and potentially damaging the component.

4.1.3.2 MLX90215

The second sensor to be considered the MX90215 was a programmable Hall Effect sensor manufactured by Melexis Integrated Systems. The 90215 came in a through-hole package much like the other sensors except that it had a fourth pin used for programming the sensitivity and other parameters. Using this component would have

forced a change in the design however it was sturdier than other sensors and the ability to set the sensitivity would ease the construction process.

- Cost: $2.38 per unit
- Type: Linear Ratiometric
- Package: Through hole
- Sensitivity: 0.5mV/G (guaranteed minimum)
- Operating range: 0-4000G
- VCC(recommended):5V
- VCC(max): 5.5V
- I(max): 6.5mA

The 90215 was certainly within the parameters required to create the sensor net. It could handle the magnetic forces created by a piece placed directly on top of it. However the fact that there was an extra pin required for the component to function as well as the price barred the 90215 from being a number one choice. That said if the budget for the project were larger and time was taken to redesign the schematic and the program itself the 90215 would work well with Deep RGB.

4.1.3.3 A1384

The A1384 was a member of the 138X family of Hall Effect sensors and like the A1302UA manufactured by Allegro Microsystems Incorporated. The 1384 like the 90215 is a programmable sensor though unlike the 90215 it was programmed before reaching the consumer. This kept the package down to three pins rather than four but did not allow for the ability to program the sensor on the go. The 1384 was an economical substitute for a programmable sensor costing less than the 90215, though the reduction in cost did correlate to a relative drop in performance when compared to the 90215.

- Cost: $3.16 per unit
- Type: Linear Ratiometric
- Package: Through hole
- Sensitivity: 2.0mV/G (guaranteed minimum)
- Operating range: 0-1750
- VCC(recommended):5V
- VCC(max): 5.5V
- I(max): 8mA

After reviewing the specifications in detail it was obvious that the A1384 was not sufficient for use in Deep RGB. The operating range was too low due to the sensitivity being as high as it was and the maximum input voltage could not be set high enough to increase the operating range enough for use. This sensor like the A1302UA would simply reach its maximum output far too early.

4.1.3.4 A1360

The A1360 was part of the 136X family and was the least sensitive of them all, like the A1302UA and the A1384 the 1360 was manufactured by Allegro Microsystems Incorporated. The 1360 was a programmable Hall effect sensor like the 90215. It was available in a four pin through hole package and could be programmed to be both unipolar and bipolar. Unlike the 90215 the 1360 was far less expensive and posed an effective substitute for the 90215.

- Cost: $2.40 per unit
- Type: Linear Ratiometric
- Package: Through hole
- Sensitivity: 0.7mV/G (guaranteed minimum)
- Operating range: 0-3571G
- VCC(recommended):5V
- VCC(max): 8V
- I(max): 12mA

The A1360 did not perform as well as the 90215 in consideration to the project. However the 1360 was much less expensive costing nearly a dollar less per component. The 1360 like the 90215 could endure the magnetic field created by a piece sitting directly on top of it. Another feature like the 90215 was the package, in that the component had four pins. While this was still a problem it was one that had to be overcome for the sake of the project. Therefore the next best option was the A1360 from Allegro and was to be the Hall effect sensor used in Deep RGB.

Since the Hall effect sensor to be used came only in four pin packages the sensor grid design had to be altered somewhat. Upon further study of the datasheet for the 1360 it was apparent that changes to the design would not be that drastic. The fourth pin controlled the filter and allowed the user to set the bandwidth and eliminate noise. This was accomplished by using a capacitor between the fourth pin and the ground. Since the parts had not been purchased it was impossible to tell whether or not the filter needed to be set since it was set to 50kHz by default. Until experiments were done there was no way to tell if the noise levels would be too high and if so what capacitor was needed to fix the error.

The number of pins used to control the sensors was the same as the components will be programmed before being installed. This was accomplished by sending a pulse through the Vout pin which then programmed the component. The 1360 had two modes which allowed the user to test the features of the component. The first was called the Hold command which set two of the programmable parameters and held them until the VCC was reset. This allowed the user to evaluate how the two parameters affect each other and how they performed together. The other mode was called the try mode which allowed the user to set one parameter which was stored temporarily until the VCC was

reset. There were also two modes which were used to permanently program the part. One was called blow mode this mode was used to program each parameter by blowing internal fuses in the part, several parameters could be programmed sequentially by using blow mode more than once. Once the parameters were set the user used the lock mode which blew a device-level fuse and prevented the parameters from being changed any further.

In order to activate the Hall effect sensors several analog MUXs had to be used. In the corresponding research section of this report it was deduced that depending on the configuration used the sensor network would require 3 8-input MUXs and one 16-input MUX for the triangle configuration or 2 16-input MUXs for the corners arrangement. Additionally the shift registers necessary to run either setup were different; the triangle arrangement required a 16-bit shift register while the corners arrangement required an 8-bit shift register. This in conjunction with the fact that the sensor being used was able to manage the stresses involved with the arrangement the all corners method was applied to the board. This reduced the cost of components as well as simplified project construction considerably.

In response to the decision above the listed components from this point onward were specifically geared toward the corners method. This decision could only be made once the sensor was chosen. The triangle method could use the components that will be reviewed though the exact placement and number of parts needed varied a little.

The first part that needed to be chosen after the sensor was the analog MUX which was used to turn the sensors on and off. There were several types of analog multiplexers which included analog to digital as well as analog to analog. Analog to digital used an ADC to sample the analog signal selected and convert it to digital so that it was easily read by controllers. Analog to analog like its name sounds merely acted as a switch and allowed a signal to pass from the pin chosen to the output. Until now the MUX had always been assumed to be analog to analog this was true for the MCU portions of this report as well. This was because the signal that was to be read was not going to pass through the MUX, the signal that turns the sensors on will be what was multiplexed and therefore it must remain analog. This said analog to digital MUXs can work although a DAC would need to be used in series with the output so that the circuit could be connected.

The MUX had to be able to take in 16 inputs and output one signal, preferably an analog signal. The package needed to be through hole like the other components so as to fit on the board being used. Cost was not a big factor when deciding which MUX to use as there only needed to be two. These specifications as well as others are listed below for easy comparison to the components being considered.

- Cost: Less than $5.00 per unit
- Multiplexed inputs: 16
- Output(s): one analog

- Package: Through hole
- Voltage input(max):5.5V
- Iin(max): 12mA

Another consideration had to be made so that the input and output are as similar as possible though this may be best determined through experimentation

4.1.3.5 CD74HC4067

The CD74HC4067 was a 16 input single source analog to digital multiplexer manufactured by Texas Instruments. It came in both surface mount and through hole packages at a low cost. The 4067 offers high speed switching using silicon gate CMOS technology.

- Cost: Less than $0.95 per unit
- Multiplexed inputs: 16
- Output(s): one digital
- Package: Through hole
- Voltage input(max):6V
- Iin(max): N/A(output is digital)

The 4067 was an affordable 16-input analog multiplexer that could operate in the ranges needed by Deep RGB. However this particular component was an analog to digital MUX and therefore caused problems when trying to use it as a digital switch like it would be in the sensor grid. It was previously stated that it is possible to convert the digital signal back to analog which was true but unless a more preferable choice presented itself this would not be done.

4.1.3.6 ADG406

The ADG406 a member of the ADG4xx family of analog multiplexers manufactured by the Analog Devices company. The 406 was a 16:1 double source analog to analog multiplexer and was available in a 28 pin DIP package. The package was equipped with 4 extra pins that were not connected when used and were marked NC in the datasheet. This was because the ADG406 shared the same package as the ADG426 which contained on chip address and control latches to make interfacing with an MCU easier.

- Cost: $0.51 per unit(available for sample purchasable in tubes of 13 for $6.60)
- Multiplexed inputs: 16
- Output(s): one analog
- Package: 28 Pin DIP
- Voltage input(max):27V(cutoff at 25V)
- Iin(max): 20mA

After reviewing the specifications above it was apparent that the 406 was a potential match for Deep RGB and met all the requirements except cost. Fortunately there was a version called the ADG406BNZ available for sample. The BNZ was actually better than the regular 406 in one way and that was that it complied with the restrictions on hazardous wastes and contained a negligible amount of lead. This said the BNZ version of the 406 performed the same as the regular and worked just fine for Deep RGB's sensor grid.

4.1.3.7 ADG426

The ADG426 was another member of the ADG4xx family and came with a few features that the 406 did not have. While it was still a 16:1 double source analog to analog multiplexer the 426 came with onboard address and control latches making it easier to control with an MCU. Otherwise the 426 had the same exact specifications as the 406 the only difference was the cost.

- Cost: $0.53 per unit(purchasable in tubes of 13 for $6.91)
- Multiplexed inputs: 16
- Output(s): one analog
- Package: 28 Pin DIP
- Voltage input(max):27V(cutoff at 25V)
- Iin(max): 20mA

The ADG426 was a much better choice over the 406 when using an MCU due to its ability to easily interface with a microcontroller. This was done by using the four pins that went unused in the 406 to set the address and latch it so that it could not change until the WR signal was lowered again. Therefore the signal incoming from the MCU did not have to remain constant, and the microcontroller could move on to other tasks and save power. Since we were using shift registers to input the data into the multiplexer and therefore the latched address feature is unnecessary for this application.

4.1.3.8 ADG506A

The ADG506A was a 16:1 dual source analog to analog multiplexer in the ADG5xx family manufactured by Analog Devices. The 506 was a CMOS monolithic analog multiplexer in comparison to the 406 which used LC^2MOS technology. The main difference between the 506 and the 406 was the resistance created by the device, while on the 506 had roughly ten times the impedance as the 406. This amounted to less than three hundred ohms of impedance to the 406's thirty ohms.

- Cost: $0.51 per unit(purchasable in tubes of 13 for $6.91)
- Multiplexed inputs: 16
- Output(s): one analog
- Package: 28 Pin DIP
- Voltage input(max):27V(cutoff at 25V)

- Iin(max): 20mA

While the 506 shared the same specifications, at least those of concern, as the 406 it did create more impedance than the latter. Since the 406 and the 506 cost the same this put the 506 at a disadvantage and was thus a less sound component for use in this application.

After comparing the four multiplexers above it was quite clear which one should be used in Deep RGB. The analog MUX that was to be used was the ADG406BNZ which as stated before was the lead free version of the original 406. This component stood out over the others for its availability to be sampled from the manufacturer, potentially eliminating the cost of this component altogether. The 406 was a dual source component and the sensor net was to be run on positive DC voltage the negative source of the MUX needed to be grounded reducing the range of the output to 0V-25V.

Since the multiplexers being used required 4 digital inputs each to control them an 8-bit serial in/parallel out shift register was used to link them to the MCU. This not only reduced the total pins needed from the MCU to control the grid to 3 it also provided the added feature of an address latch. Once the register was loaded and the store command was enabled the address saved in the register and was constantly outputted until the register was changed. This freed up the MCU to perform other tasks without having to keep the output to the multiplexers at a steady level. Shift registers were used in the design of the LED subsystem and since there was a quantity of 8-bit shift registers purchased for that application the same component was used for this one as well.

When the project moved into the building phase it became evident that the system needed to change. The sensors were found to be inadequate and therefore replaced with A1325 from Allegro. Once the sensors were received it was decided to instead power them all at once and use 6 CD4067BE analog multiplexers from Texas Instruments to funnel the data into the microcontroller. The MUXs were then integrated into the PCB and the outputs of the sensors were wired from the board to them. The arrangement also changed for this system, instead of placing a sensor at each corner like planned a sensor was placed in the center of each tile. This was a decision that was made when we had still planned on using an electromagnet which we had planned on mounting a mobile sensing array to. When the electromagnet could no longer be used it was already too late to change the board and it was decided to forgo the extra accuracy in order to move on with the project.

4.1.4 AUDIO SYSTEM

As discussed in the corresponding research section of this project the chessboard was expected able to play alert sounds when certain events occurred during a game. In addition the board would play music based on the preferences set by the user which were loaded onto the board from the server when the user logged in. To do this it was

necessary to use an audio shield to decode the mp3 files stored in memory and output them as signals to the speakers making the appropriate sound. There were several of these peripherals available on the market several of which were very reputable and well known amongst hobbyists who used Arduinos.

A criterion needed to be established by which to select the proper shield; to begin with the audio shield had to be able to play mp3 files so that the sound quality was not too low like MIDI or WAV files. Secondly it had to output to either a speaker or a 3.5mm jack although both would be preferable. The ability to store data on the board itself either by removable drives or memory integrated into the board itself had to be available. These criterion and others are listed below to facilitate in the comparison of the different components.

- Cost: Less than $50.00
- Decoding capabilities: Mp3 or better
- Outputs: 3.5mm jack or speaker output
- Data Storage: Removable drives or onboard memory
- Input: SPI interface
- Voltage input: 6V or less
- Current input: 100mA or less while running
- Interrupt capable: Yes
- Arduino-019 Audio shield

The Arduino-019 was an audio shield based on the popular VS1053b mp3 decoder chip and specifically designed by Seed Studio to interface with Arduino microcontrollers. The 019 was a cost effective way to decode mp3 and other audio files while being driven by an Arduino. The 019 could output through a 3.5mm jack it was also capable of output to external devices through the I_2S serial connector and had an input 3.5mm jack for a microphone. Not only was it capable of decoding mp3s it was also capable of encoding Ogg Vorbis files from the microphone input. With this capability the 019 could take in a signal from an external device such as an mp3 player encode it as an Ogg file then replay it through the speakers.

- Cost: $27.50
- Decoding capabilities: Mp3, Ogg Vorbis
- Outputs: 3.5mm jack and line out
- Data Storage: micro SD card up to 2Gb
- Input: SPI interface
- Voltage input: 5V
- Current input: Unspecified
- Interrupt capable: Unspecified

The 019 boasted some impressive specifications although it had a few flaws the most concerning of which, was the lack of information provided by the datasheet available. Due to these missing details it was not possible to determine the current needed to

operate the board nor was it apparent whether or not the music could be interrupted at a moments notice although the likelihood that it could be was high. Therefore the 019 was only to be used as a last resort if a better candidate could not be found.

rMP3

The rMP3 was one of the newer Arduino peripherals created by Rogue Robotics. It was compatible with some of the more popular data structures such as FAT32 and capable of reading a micro SD card of up to 32Gb. It was also capable of supporting higher bit rate mp3 files than most audio shields since it could decode files with bit rates up to 320kbps. It also had the capability to read and write audio files from an external source to memory rather than just read and decode them from memory. However this did not come cheap and the rMP3 cost almost three times the average audio shield.

- Cost: $64.99
- Decoding capabilities: Mp3
- Outputs: 3.5mm jack
- Data Storage: micro SD up to 32 Gb
- Input: SPI interface
- Voltage input: 5V
- Current input: 60mA
- Interrupt capable: Yes

The rMP3 was an impressive piece of hardware although it did have its drawbacks, mainly the price was higher than the budget allowed. The unit did more than it needed to for its application and therefore the extra cost was unnecessary. If the project were ever to be turned into a consumer product the rMP3 could have added a few features to the board making it more desirable. However as the project did not require such a sophisticated piece of equipment it must be dismissed as a viable part.

DEV-10628

The DEV-10628 was an audio shield designed around the VS1053b mp3 decoder and was geared towards use with Arduino microcontrollers, capable of decoding mp3s as well as Ogg Vorbis and other popular audio formats. It was run using SPI between it and the MCU, the MCU loaded the buffer on the shield with the information that it needed to run for a small fraction of time. While the shield was performing the tasks on the buffer the MCU was free to perform other tasks. This buffer was relatively short so the MCU could wait until the end of the buffer to load an alert sound and most users would not notice.

- Cost: Less than $39.95
- Decoding capabilities: Mp3, Ogg Vorbis
- Outputs: 3.5mm jack and L, R, GBUF hardwire
- Data Storage: micro SD

- Input: SPI interface
- Voltage input: 5V
- Current input: N/A
- Interrupt capable: Yes

The 10628 was an inexpensive audio file decoder for use with an Arduino MCU, based on the most popular decoding chip used for those applications. Capable of decoding some of the most popular and best quality sound files used at the time the 10628 was also capable of outputting the sound to an amplifier used to drive large speakers. Thus the 10628 was the ideal audio module for use in this project for its simplicity and lack of frivolous features.

After the audio shield had been purchased it was connected to the MCU using 8-Pin headers to create a fast and reliable connection. When the player logged in the MCU received the data from the server and commanded the audio shield to play the music that the user preferred. Like the auxiliary input capability this feature might not have made it into the final prototype however it should have been included if possible. The MCU would also receive data from the server on the conditions of the match. If a notable event occurred then the MCU would receive an alert from the server where the chess algorithm was being run. The MCU then wrote the data onto the audio shield's buffer so that it stopped playing the music and sounded the appropriate alert which was stored on the micro SD card. For the purposes of this project a pair of powered computer speakers were to be used so that the signal did not have to go through an amplifier to be played.

Figure 20 is a flowchart that showed how the audio system would have generally worked, it must be noted that the external device and user blocks were there for the possibility that they would be included in the final prototype. They however were more likely to not be included and therefore were outlined and connected by dashed lines. It shows that the server communicated with the MCU this was through Wifi which was discussed elsewhere in the project documentation. The server sent the MCU user data as well as information such as alerts which the MCU sent to the audio shield in turn via headers where it was decoded and output through the 3.5mm jack into a switch and then into the speakers where it was converted into sound. The switch was controlled by the MCU which allowed the appropriate signal through based on the user input and current alerts sent to it by the server. As with the user and external device blocks the switch block is a dashed line because it may or may not have been included in the final product. This was also why there had not been any research into what switch to use for this application.

During early production the team did some viability testing which was mentioned later in the testing section of this report. One of the tests included using a development board to simulate the MCU controlling the different components. It became apparent then that there was a severe timing issue when trying to use the SPI bus to control both

the LED matrix and the Audio module. Since the Audio module was less vital to the overall project it was decided to not incorporate it into the final product.

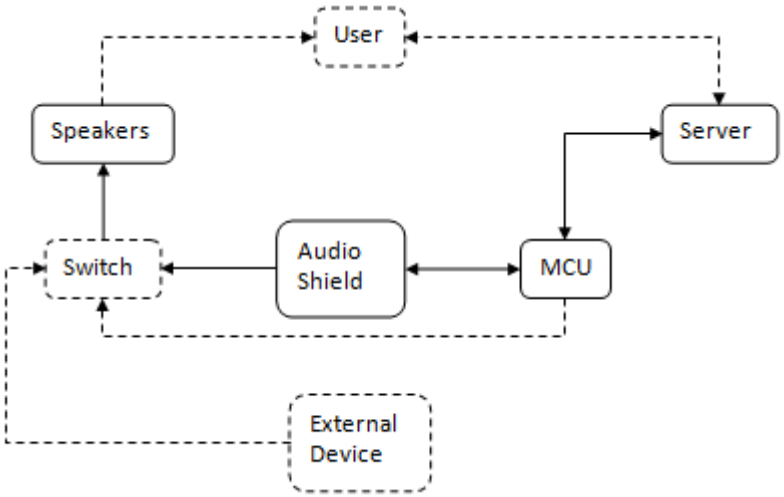


Figure 20 - A flow chart describing how the audio components will interact with each other

4.1.5 LED SYSTEM

To start off the LED system the first thing that had to be done was select the appropriate LED to use. It had already been decided that the LEDs used were to be a through-hole mount RGB LEDs but there were many different components that fit this description. To narrow it down further the LED selected had to fit the following criteria.

- Cost: $2.00< per unit
- Luminosity: between 500 and 5000 mcd (They can't be too bright or they will pollute other squares
- Pin Configuration: Common anode
- Height: Less than 45mm from tip to tip
- Diameter: 5mm
- Voltage(max): 5V or less
- Amperage(peak): 50mA or less
- Viewing angle: less than 80 degrees greater than 20 degrees

Once this criterion had been used it narrowed the search down to a more reasonable level. After looking through several sites from which to purchase the LEDs it came down to four possibilities. The four were listed below along with their relevant specifications.

4.1.5.1 RL5-RGB-C-2

The first of the four LEDs being considered was the RL5-RGB-C-2 (hereon referred to as the C-2). It was available from the company Super Bright LEDS. It was a cost effective RGB LED available in both common cathode and common anode. The rest of the specifications were as follows.

- Cost: $0.79 per unit
- Luminosity: 1200-2200 mcb
- pin configuration: common anode/cathode
- height: 35.1mm
- Diameter: 5mm
- Voltage(max): 2.4V,3.5V,3.5V
- Amperage(peak): 50mA
- Viewing angle: 60 degrees

The C-2 was within all of the parameters but the luminosity was lower than the other LEDs presented. On the other hand its viewing angle was larger than the other components presented. This could have helped make it more viable since the wider the viewing angle the more lumens or photons the component projects therefore making it able to light up a larger area. Another benefit of the C-2 was that it was shorter than most of the other LEDs being considered making it possible to create a skinnier board.

4.1.5.2 RL5-RGB-C

The RL5-RGB-C (hereon referred to as RGB-C) was in the same family as the C-2 LED. It was available from the same company Super Bright LEDs and had similar characteristics to the C-2. The full list of relevant specifications was given below.

- Cost: $1.59 per unit
- Luminosity: 1000-5000 mcb
- pin configuration: common anode/cathode
- height: 35.1mm
- Diameter: 5mm
- Voltage(max): 2.6V,4.0V,4.0V
- Amperage(peak): 50mA
- Viewing angle: 15 degrees

The RGB-C was relatively expensive to the other LEDs costing over twice as much as the C-2. However it did have a greater luminosity range than the C-2. This extra luminosity was a benefit but was severely limited by the viewing angle which was rather small. Thus the RGB-C was more suited for focused lighting applications such as flashlights and lamps rather than for illuminating a chessboard. If it were used the RGB-C had the potential to light up a small portion of the squares rather than the hole area.

4.1.5.3 YSL-R596CR3G4B5C-C10

This particular LED (referred to as the C10) was made by a Chinese company called China Young Sun LED Technology Corporation and was available on the Sparkfun website. It was available in packages of 100 which drove the cost per unit down. These were still more expensive to buy than the C-2 but since there were thirty-six extra it gave the group room for error. The specifications of the C10 were as follows.

- Cost: $0.60 per unit (when bought in packages of 100)
- Luminosity: 800-4000 mcb
- pin configuration: common anode/cathode
- height: 38.6mm
- Diameter: 5mm
- Voltage(max): 2.2V,3.4V,3.4V
- Amperage(peak): 30mA
- Viewing angle: 40 degrees

The C10 was an excellent choice to use in Deep RGB. It was inexpensive and came in packages large enough to account for any mistakes made while soldering the LEDs in place. It had the right pin configuration available and it was short enough for the purpose it was fulfilling. The light output while less than the RGB-C was still backed by the larger viewing angle. Another perk to the C10 was the lower max amperage which helped lessen the strain on the power supply.

4.1.5.4 YSL-R596CR3G4B5W-F12

This LED (referred to as the F-12) was manufactured by the same company that made the C10 and was very close to the C10 in its specifications. One of the differences was that the F12 was a diffused lens LED rather than a clear LED like the previous components. This helped the light scatter and become less focused than that of the clear LEDs which was ideal when trying to light up an area rather than a single point like on the chessboard. The specifications of the F-12 were as follows.

- Cost: $0.60 per unit (when bought in packages of 100)
- Luminosity: 1200-6500 mcb
- pin configuration: common anode
- height: 38.5mm
- Diameter: 5mm
- Voltage(max): 2.2V,3.4V,3.4V
- Amperage(peak): 30mA
- Viewing angle: 40 degrees

The F-12 while costing the same as the C10 had a brighter luminosity which combined with the same viewing angle made it easier to light an area. The diffused lens was

beneficial as well since the light would not be as focused and wouldn't appear as a single point in the middle of each square. These attributes made it nearly perfect for this application however before a final decision could be made one more test had to be conducted.

The parameter 'luminosity' was given in mcb or millicandelas; this was an adequate way of measuring the brightness of a source but did not account for what happened to the light after it left the source. The unit mcb was used to mainly describe LEDs that were being used in focused applications such as lighting and flashlights. These units could be converted into lumens which measured the light intensity over an area; it was proportional to both mcb and the viewing angle. The higher the lumen count the brighter the lighted area would be. Below is chart (Table 26) with the lumen count for each LED and was helpful in determining which of the LEDs was best suited for the task of lighting the board.

As the table shows the two LEDs that gave out the most light over the largest area were the C-2 and the F12. Out of these two the C-2 was less expensive if only sixty four were bought, that being said it was decided to go with the F12 due to the fact that it came with spares. The F12 might have put out a little less light per area but this was a small difference and in the end it did not matter. However when the time came to order the LEDs the F12 was no longer listed in packs of 100 on sparkfun.com. Since the cost of a single unit of the F12 was higher than that of the C-2 it was decided instead to purchase the C-2 in a common cathode arrangement.

Table 26 - A chart comparing the lumen values of the LEDs being considered for Deep RGB

LED	C-2	RGB-C	C10	F12
Red	1.18lm	0.07lm	0.31lm	1.07lm
Green	1.86lm	0.27lm	1.52lm	2.47lm
Blue	1.02lm	0.06lm	0.35lm	0.46lm
Total	4.06lm	0.40lm	2.18lm	4.00lm

After deciding which LEDs to use a decision had to be made as to what shift registers to use in the project. The purpose of the shift registers as stated before was to reduce the amount of outputs needed from the MCU. The LEDs were arranged in a grid and called by activating the anode and cathode of each specific LED color. To do this we linked each row with a common cathode which required 8 bits of data to control. Then each color anode in each column was connected by a common bar thus requiring another 8 bits of data for each color to control. In total there were 32 bits of data which required 4 8bit shift registers to regulate them.

The schematic in Figure 21 shows how each of the LEDs was wired so that only four 8 bit registers were needed to run the entire matrix. Now that the need for these components had been established more parameters needed to be defined so that a shift register could be chosen.

- Cost: $5.00 < per unit
- Size: less than 2 inches in length
- VCC: 5V or higher
- Iout: greater than 30mA
- Clock: greater than 16MHZ

The most commonly used family of shift registers for this application were the 595 series. The 595 was a serial in parallel out register which was perfect for the LED matrix because it reduced the data signal down to one bit. There were several of these 595 series registers and two of them came under consideration while trying to find the right parts for the project.

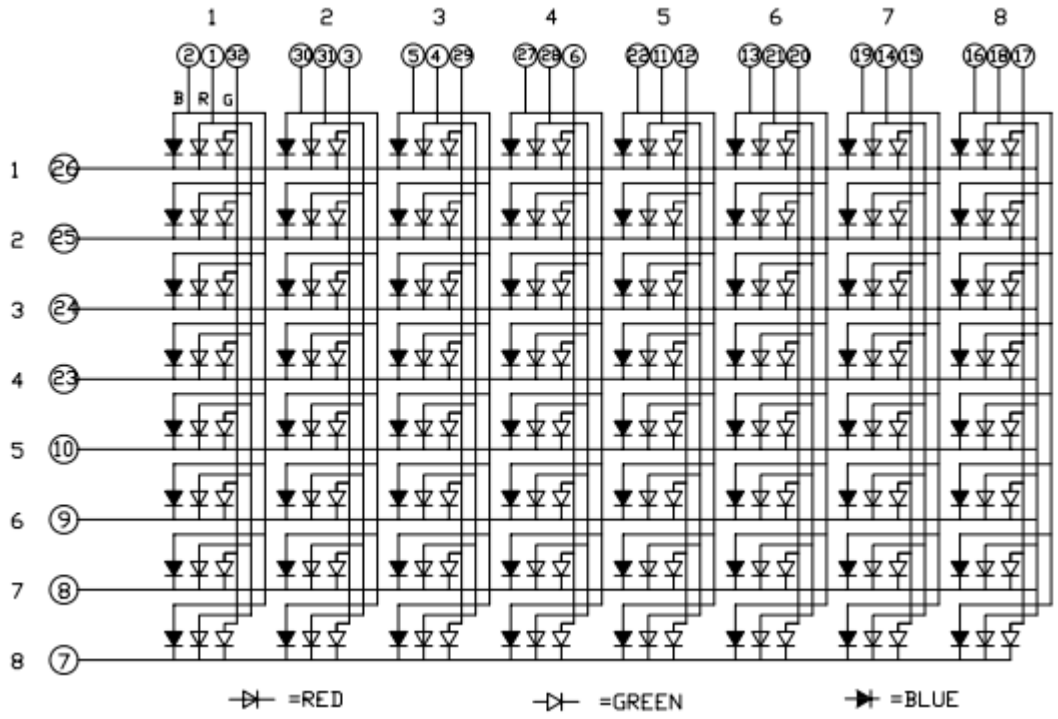


Figure 21 - An example schematic of an LED matrix awaiting reply to request for permission to reprint from the China Young Sun LED Technology CO. Picture taken from the YSM-2388CRGBC datasheet.

4.1.5.5 74HC595

The 74HC595 was a low current 8 bit shift register that was made by Texas Instruments, commonly used for latching data to run LED matrices and other large data applications to free pins on the microcontrollers being used. The specifications of the 74HC595 were as follows.

- Cost: $1.50
- Size: 2 cm in length
- VCC: Maximum of 6 volts
- Iout: 35mA
- Clock: 29MHz at VCC=6V

From the data it was easy to see why the 74HC595 was a commonly used component for LED matrices. It was relatively inexpensive, costing well under half the allowed price per unit. It was compact measuring under an inch (2.54 cm) in length. The register met the power needs of the LEDS being able to withstand up to 35mA of current passing through the component. Finally one of the most important features of this component was the clock cycle which was greater than that of the MCU being used so the registers would not slow the MCU down.

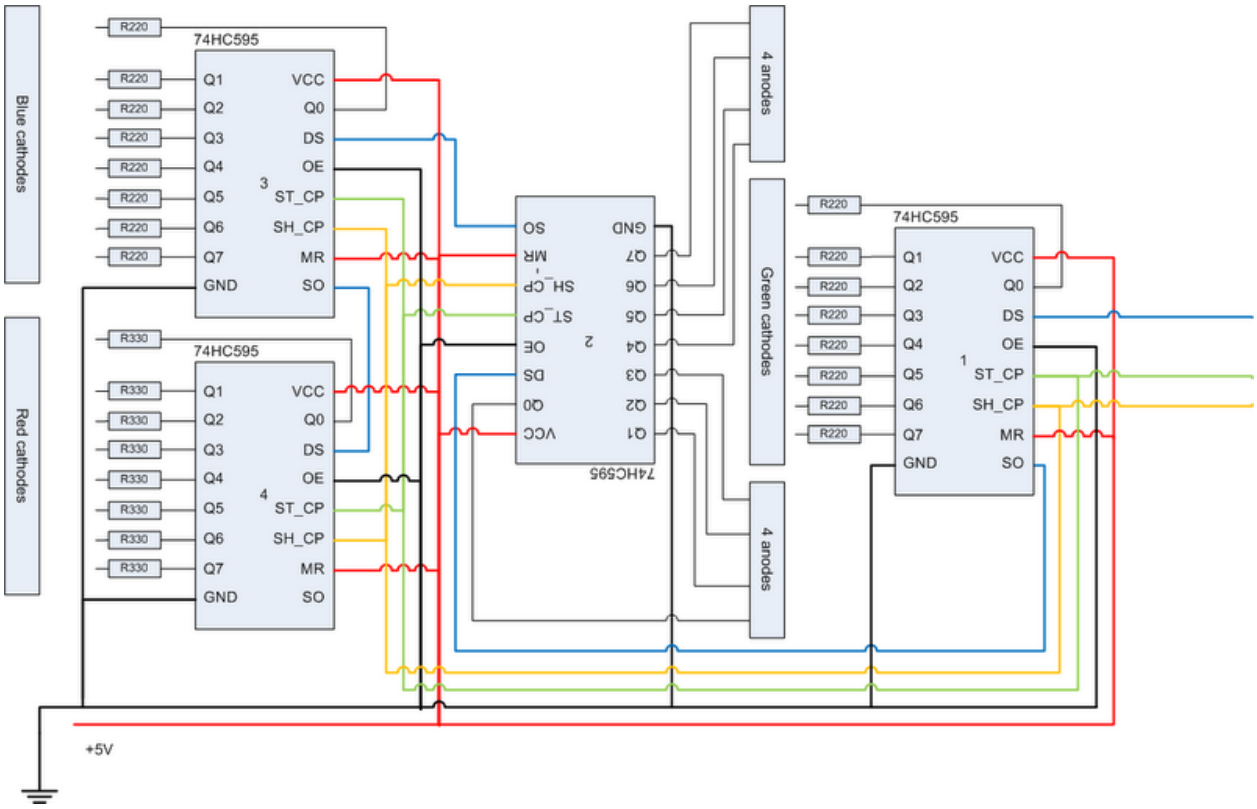


Figure 22 - A schematic of the four shift registers connected together awaiting response to request to reprint from Francis Shanahan

http://francisshanahan.com/index.php/2009/how-to-build-a-8x8x3-led-matrix-with-pwm-using-an-arduino/

The other candidate for the shift register was the TPIC6B595 which was a high power register. This register while fitting inside the parameters for the shift register, would simply have been overkill and so the 74HC595 shift register was used in the project. The four shift registers were connected together such that the data stream flowed through

one register and out to the input of the next register. This way there was a continuous stream of data and each register could be loaded from the same output pin. The other two inputs absolutely necessary for the matrix to function were the shift clock and the load clock. Each register output to one of the busses green, red, blue and anode. Once completed it resembled the schematic provided in Figure 22.

Once the matrix was completed and connected to the MCU via the shift registers the MCU simply loaded a bit onto the first register then when the shift clock went high it shifted over one and the MCU loaded the next bit. By this process the data was pipelined down the registers until all were loaded and then the load clock was activated which stored the data in the registers. This was a total of three outputs that the MCU must calculate and output, the details on how the MCU did this were discussed in the MCU software section of this report.

4.1.6 LCD DISPLAY SYSTEM

The display was not one of the defining features of Deep RGB although it was a necessary one. The main purpose of the display in Deep RGB was to provide extra information about the game such as the status of the game, as well as information required to log on to the system at the start. At the start of the game, the display showed this information and assisted the user in setting up an internet connection. It needed to be capable of displaying all this information without clipping.

The display needed to be able work inside the board, requiring that it operated correctly inside the temperature range from 5 C to 40 C. The cost of display was the lower than other elements of the board, and it needed to be able to complete its task while being as small as possible. From our display research, we found that the alphanumerical display model, BLUELCD16x2BL, met all our requirements. It had 2 lines with 16 symbols in each line, allowing us to display all necessary information. The resolution of each symbol was 5 by 8 pixels, which allowed us to use a wide selection of preinstalled symbols and even create some of our own if required. The full size of the module was 8 cm by 3.6cm while the actual display size was 6.5cm by 1.6cm.

The display BLUELCD16x2BL required a 5V DC supply voltage and supply current in the range of 2mA to 4mA. However, it had supply voltage range of 2.7V to 5.5V, which in our case could be used successfully with either the 3.3V supply line or the 5V supply line. Supply voltage ranged for the LCD from 3V to 13V, which allowed us to use this LCD with any supply line in our project. Operational temperature ranged from 0 C to 50 C which satisfied our temperature requirements.

Due to the fact that alphanumerical display BLUELCD16x2BL had a preinstalled display driver, the Hitachi HD 44780, our task of connecting the display to the MCU became far more simple. To connect the display, we needed only know the pin configuration of the driver. The BLUELCD16x2BL had 16 pins, with numbering going from left corner around

to the right. In the Figure 23, we illustrate the basic connection of the BLUELCD16x2BL display to MSU.

The V_o connection was used for contrast adjustment. R_s was used to register selection, and by connecting to ground, we could send it a low status. The R/W stands for read and write; we also connected this to ground. Display DB6 was used to enable or disable the signal. Pins DB7 through DB14, inclusive, were used to send data bits from the MCU to the display. For the regular text information we could get away with using only 4 data pins, from the 11 to 14 inclusive. To activate a backlight we used pins 15 and 16, where A stands for Anode (which needed to be connected to V_{cc}) and K stands for Cathode (which needed to be connected to the ground).

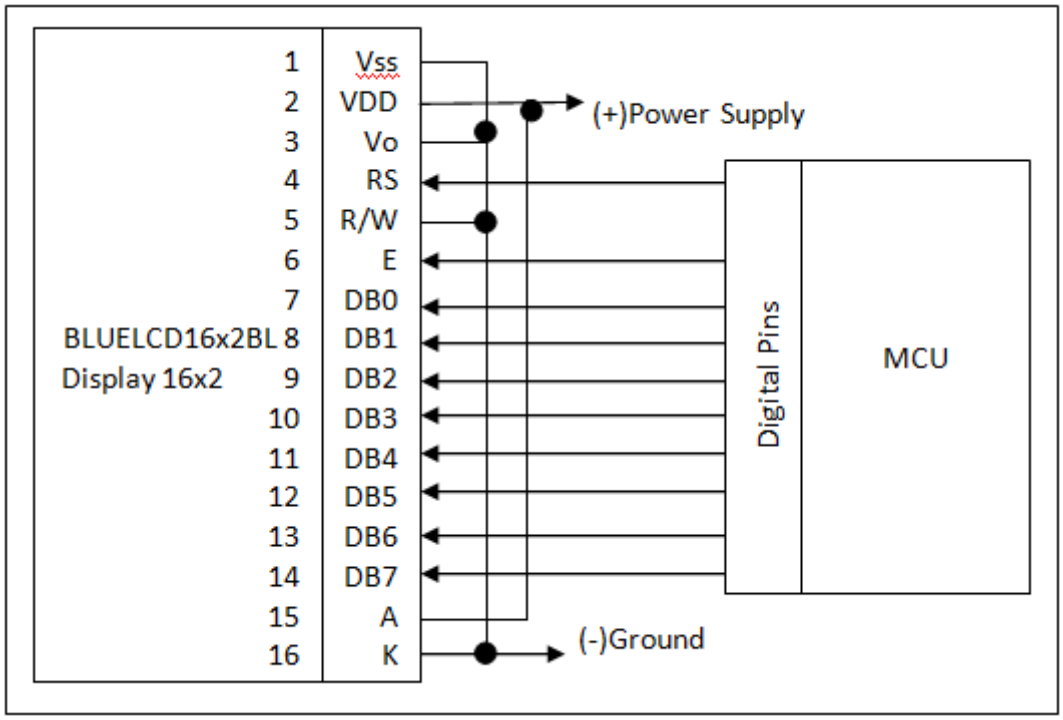


Figure 23 - Display connection diagram to the MCU

4.1.7 POWER SUPPLY

The implementation of power supply system in Deep RGB project was very important because the appropriate function of all components of the Deep RGB system relied squarely on the quality of the energy provided by the power supply. There were two important components of the power supply system: the source of the power supply, and the power distribution method within the system.

For the first component of the power supply system we included power outlet and AC/DC power adapter. The power outlet supplied power at 110 V and 60Hz to the adapter. The adapter lowered and rectified the input supplying 12 V DC voltage output to the Deep RGB system. Selection of the place where adapter was to be located whether inside of the board or outside was an important decision because the adapter had a transformer in it which got very hot. Since we needed to decrease the amount of heat inside the chess board, we decided on the external adapter placement. A laptop adapter ideally suited our requirements due to its external usage and high quality power output.

For the second component of the power supply system we included elements which needed to be implemented on the PCB board. Those elements were 5V DC and 3.3V DC voltage regulators. It also included some capacitors to reduce noise and some resistors to supply the appropriate current level to the components of Deep RGB.

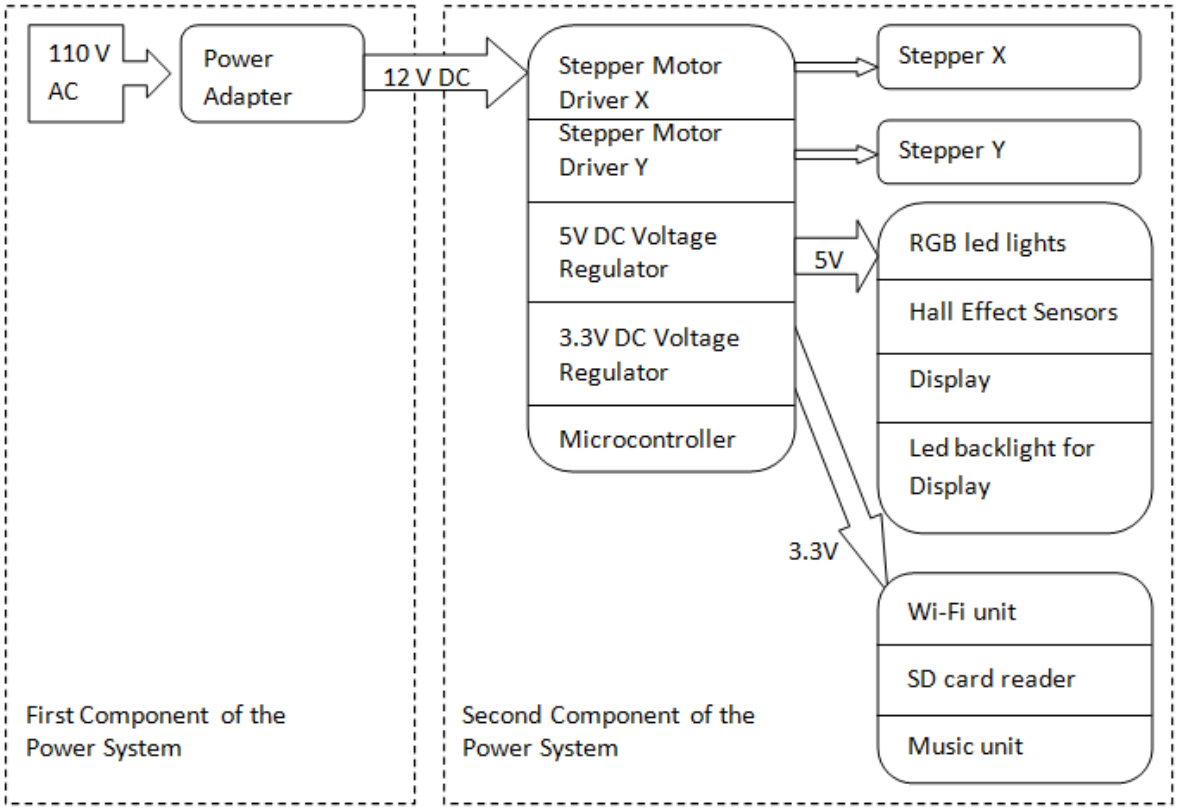


Figure 24 - Power Supply Block Diagram

Another approach to the design of the second component of the power supply was to have an individual voltage regulator for each component of the chess board. An advantage of this approach was that we can protect every other component in case a hazard of sorts arose in one single element. The disadvantage of such an approach was higher cost of implementing the power supply system. We would also have needed to

implement surge protection in the power supply itself. Incorporation of overheat protection was not very important in this case because all major elements had a built in overheat protection. In Figure 24, we demonstrate the implementation of the first approach to the whole power supply system.

During the PCB design and manufacturing process the team ran into a few roadblocks. After having the PCB created several weeks were spent trying to find a company to populate the board with the surface mounted parts. When no company was found the group decided to incorporate a third party power supply into the board. After some research we found the picoPSU-160-XT and used it to convert 120V wall power into the several different voltage levels we needed.

4.1.8 WIRELESS CONNECTIVITY SYSTEM

The wireless data transmission device was to be placed on a PCB alongside the other devices within the physical board. As mentioned in section 3.2.6.4, the Wi-Fi module chosen for use was a Roving Networks RN-131G as shown in Figure 25. Since it only required four wires to operate (PWR, GND, TX and RX), this module could be surface mounted onto the PCB and connected to the Atmel Corporation ATmega 2560 via pathways.

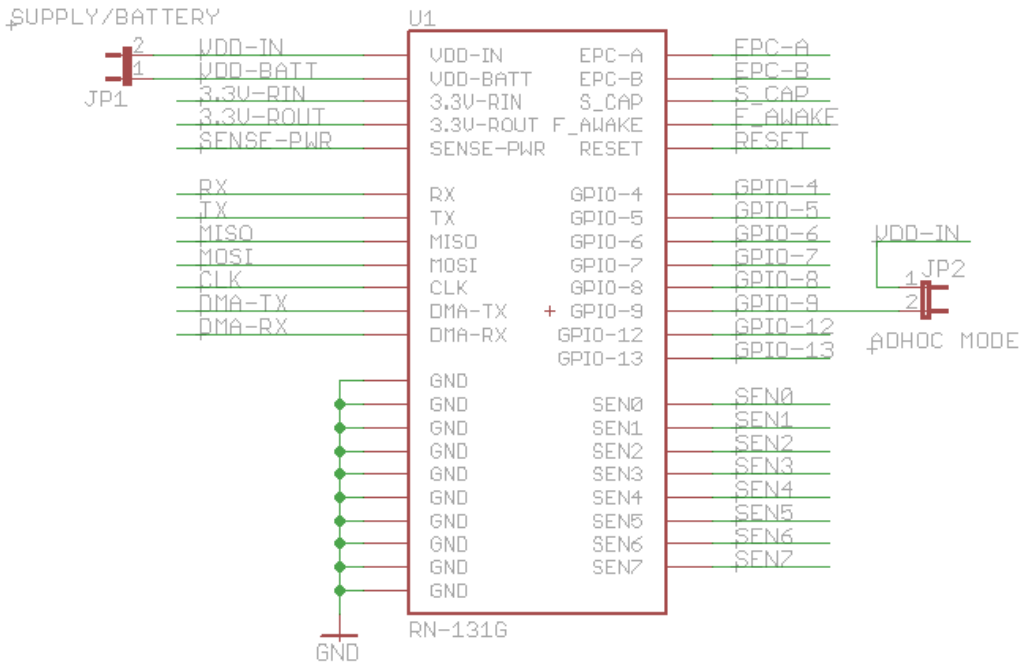


Figure 25 – The Roving Networks RN-131G and its pin labels for mapping reprinted with permission from Sparkfun.

The Roving Networks RN-131G was also equipped with ten general purpose digital I/O and eight analog sensor interfaces. Since the analog sensors were connected to the Atmel Corporation ATmega 2560 directly, these inputs and outputs remained unconnected. This module was capable of establishing a connection using UART or SPI, the UART option was chosen in order to free up the SPI bus during heavy data flows. This module also had the capability of having an antenna attached to allow for a longer connectivity range. After the PCB production ran into trouble the WI-FI module along with a few other components were instead mounted to prototyping board in order to make small breakouts.

4.1.9 PCB DESIGN

Since strip and perfboard could not be used for our project, the PCB layout for our circuit needed to be decided upon. Using Eagle Cadsoft, we designed our PCB and added the components onto the design to ensure that each device was obtaining a connection to their required pins. Using Eagle Cadsoft allowed us to move around modules before making a final decision. The placement of the modules was crucial, for example; the Wi-Fi module was best placed as close to the housing as possible in order to maintain high signal strength through the physical chess board casing. A prototype layout for our PCB can be seen in Figure 26

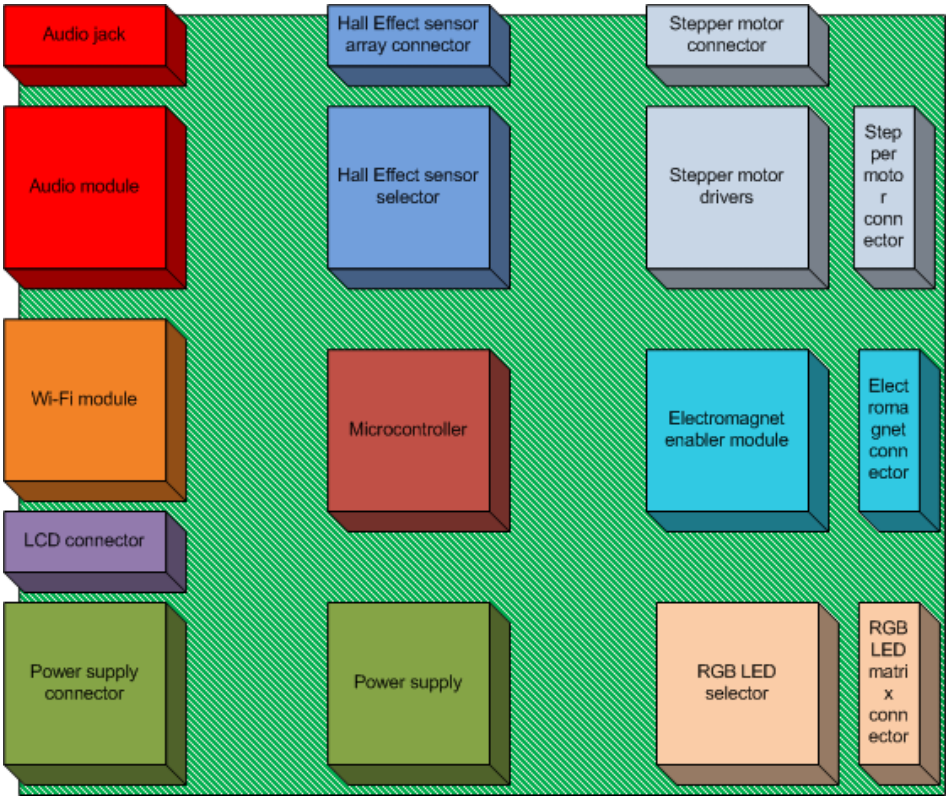


Figure 26 – The general layout of the PCB design.

The modules used for this project needed to be placed in a certain way that it maintained a small footprint and kept crosstalk to a minimum. An easy way to reduce crosstalk and ensure that pathways didn't cross over each other was to make a two layer PCB. Having multiple layers allowed us to shrink the total size of the PCB but increased the price per board, although the benefit of a multi-layered PCB greatly outweighed the increase in cost. Some of the PCB could not be completed due to the inability to find a way to mount some of the smaller more delicate parts. Therefore several components were not incorporated on the final PCB and were used by mounting them onto prototyping board. These components had more details placed within their own sections rather than listing them all here.

4.2 MAIN CONTROL UNIT SOFTWARE DESIGN

We used the Arduino IDE to code the Atmel Corporation ATmega 2560. Most of the devices already had open source libraries associated with them. This allowed us to have every device up and running as quickly as possible. The libraries needed to be installed in the same folder as the Arduino IDE in order to have them function as intended. These libraries were provided for public use and had a large community following that could provide feedback if the need for troubleshooting arose.

4.2.1 HALL EFFECT SENSOR ARRAY CODE

The Hall effect sensor array code needed to send data to the shift registers in order to scan specific squares through the SPI bus and the analog input assigned to the array. The array was made active when the slave select pin went low. There were for loops designed to quickly scan the values coming from the Hall Effect sensors. These values needed to be converted in a quantifiable number using the A/D converters on the microcontroller. The values in the array were then compared to see if any changes had occurred, if the piece wasn't centered on the square or if an illegal move was made, it needed to then correct the position of the piece moving to the center of the square and report back the new position via the Wi-Fi module.

4.2.2 RGB LED ARRAY CODE

Using the SFRGBLEDMatrix library, the RGB LED array was able to display several different colors through PWM. The array used the SPI bus to select specific squares that needed to be lit and wrote the color values to those squares. The array was made active when the slave select pin went low. When a piece was picked up by the user, the code needed to light up the blocks where a possible move could be made. Since chess has a strict set of rules on movement, this was easily implemented within the microcontroller's code. A player could also assign a color to their account and this color was displayed whenever the user is logged in and in a game.

4.2.3 AUDIO MODULE CODE

The audio module utilized the SPI bus in order to select and play an MP3 file located on a built in SD card. The SFEMP3Shield library was capable of selecting a song from the SD card and loading it onto the audio module's buffer. The buffer was capable of holding up to 100ms worth of audio data while the microcontroller performed other tasks. The code was also able to control the volume and save that value for each account. Unfortunately the audio module caused too many problems when it was implemented so it was not included in the final project.

4.2.4 MAGNETIC PIECE POSITIONING CODE

The code needed to control the 2 stepper motors used a very simple stepper library. The library known as AccelStepper only needed two inputs in order to move a stepper motor. The two variables were speed and number of rotational degrees. The chess board resembled a Cartesian coordinate system and the microcontroller had specific movements saved for each type of chess piece. If a piece was in the way, the code then needed to move the piece along the border of the tile and to its new location. Since an electromagnet was to be used, the code needed to enable the device when it moved a piece and disabled it when it was not in use in order to save power. Since the solenoid that replaced the electromagnet worked in a similar fashion this code didn't need to be changed as well.

4.2.5 WIRELESS DATA TRANSFER CODE

The Roving Networks RN-131G used the WiFly Serial open source library. In order for the library to work, some extra libraries needed to be installed in order to simplify the coding even further. Libraries such as Pstring (a lightweight class for printing to buffers), Streaming (a method to simplify print statements), DateTime (a library for keeping track of the current date and time in software) and NewSoftSerial (an improved version of the SoftwareSerial library) were also included in the same directory of libraries since the WiFly library needed access to them.

Only the Service Set IDentifier (SSID), encryption type (if any) and password (if any) were needed in order to secure a connection between the wireless network module and the network. Other options included port number, but that wouldn't be accessible to the end user. The port number will default to 80 since HTTP was TCP/IP on port number 80.

The values needed to connect to an access point were entered in by the user via a makeshift keypad designed using the squares of the chess board as a keyboard. This information could be saved to prevent the user needing to enter in their settings every time they want to connect to their network.

4.2.6 LCD CODE

The code for the 16x2 LCD used a library known as LiquidCrystal. This library greatly simplified the code needed to display text on the LCD. Its main function was lcd.print(), where one only need to write a string of text in between the parenthesis and it would display it on the display. This was used to prompt the user for account information. When not prompting the user, the LCD displayed useful information such as time and amount of moves placed so far.

4.3 SERVER DESIGN

After considering all possible software options presented in Section 3.2.7, it was decided that a web-based system would be constructed to handle the server-side processing of the chess engines and remote-play options. A web-based approach was chosen for many reasons including facilitation of alternatives, portability, and maintainability.

By avoiding direct port-to-port communication, it was immensely easier to create alternative interfaces for the system, which was a design intention from the beginning. While Deep RGB included only a website interface for remote play, it was intended to be able to accommodate the creation of iOS or Android applications or even native Windows, Mac, or Linux applications created by other users at other times in a graceful manner. The use of a web-interface combined with the portability of the Internet allowed for almost any programming language to interface with the server and thus provide access to the system.

Portability was important to the project because it facilitated the creation of alternative interfaces as explained above by being available to a wide range of other systems. As a web-based system, Deep RGB's server would be available to any machine that could log onto the Internet, which at the time included desktop and laptop computers, tablet devices, smart phones, even ordinary smart phones, and the list grew by the year. By interfacing via the Web, Deep RGB would be available to all of these platforms if a corresponding front-end system was available. For most systems, the included website interface was perfectly suited to on-the-go use of the system.

As for maintainability, a web-based system was the height of simplicity since all code for the system could be housed on a single server and the majority of it was available in non-obfuscated plain-text, not the unreadable byte code of Java or the machine code of pure compiled languages.

The Deep RGB server system was designed in C#.NET and ASP.NET using Microsoft Visual Studio 2010. As the multitude of in-progress games needed to be stored in a compact manner, a Structured Query Language (SQL) database was created using Microsoft SQL Server as it integrated with Visual Studio very easily and allowed use of the very powerful LINQ (Language Integrated Query) to SQL implementation of SQL

interaction. The chess website as well as the game progress update and request pages had to be implemented with ASP.NET frontends and C# backends. There was also a single management thread created in C# that ran continuously checking for games that needed to be played by a computer opponent and sent those games off to the chess engine to be processed. A visualization of the integrated server system is available in Figure 27 - Visualization of Chess Server System.

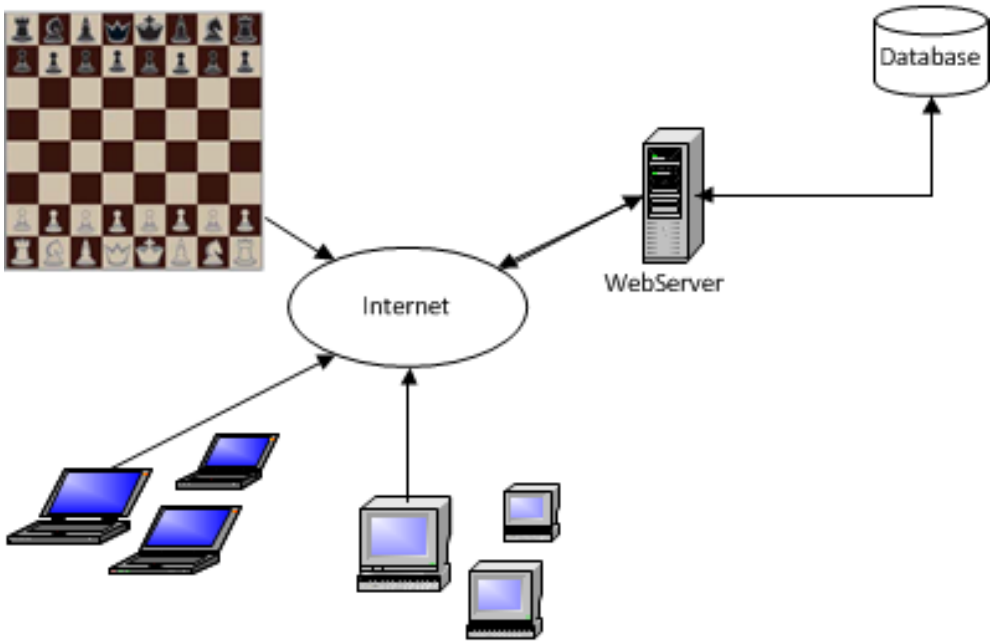


Figure 27 - Visualization of Chess Server System

4.3.1 DATABASE

The server system was driven by a Microsoft SQL Database. This database consisted of five tables, *users*, *games*, *moves*, *sessions*, and *reset*. The database kept information gathered via the various management pages described in Section 4.3.3 and stored it in an organized manner. The database was not made directly accessible from outside the server system. The only way to view information in the database or update the values that were stored there was through the management pages. Even though this was true, the database had to be encrypted in memory.

4.3.1.1 TABLES

The ***users*** table was created to store relevant information about each individual user in the system. It had columns for user id number, username, the user's real name, LED color, and audio theme, as well as a hashed password field and fields for the date and time the user's account was created along with when the user last made a move.

The ***games*** table stored the relevant state information about each of the games the system was tracking, in-progress or completed. It's list of columns included game id number, white player id number, black player id number, observer id number (if applicable), whether or not observers were allowed at all, the current state of the board, the turn number, which player's turn it was, the date and time when the game was created, and the date and time of the last move made in the game.

The ***moves*** table held a single record for every move taken in every game. Whenever a move was made via the update move management page, a record was inserted into this table, tracking the id number of the move, the id number of the game that was being played, the id number of the user that made the move, a bit representing which side made the move (black or white), what the turn number was when that move was made, and the time and date that the move was made.

The ***sessions*** table was a temporary record table designed to store the authentication keys of users who had logged in via the log in management page. The only records in this table were user ids and the hashed authentication key that went with them. This table was occasionally cleaned out if the user the authentication key was assigned to had not made a move in more than an hour.

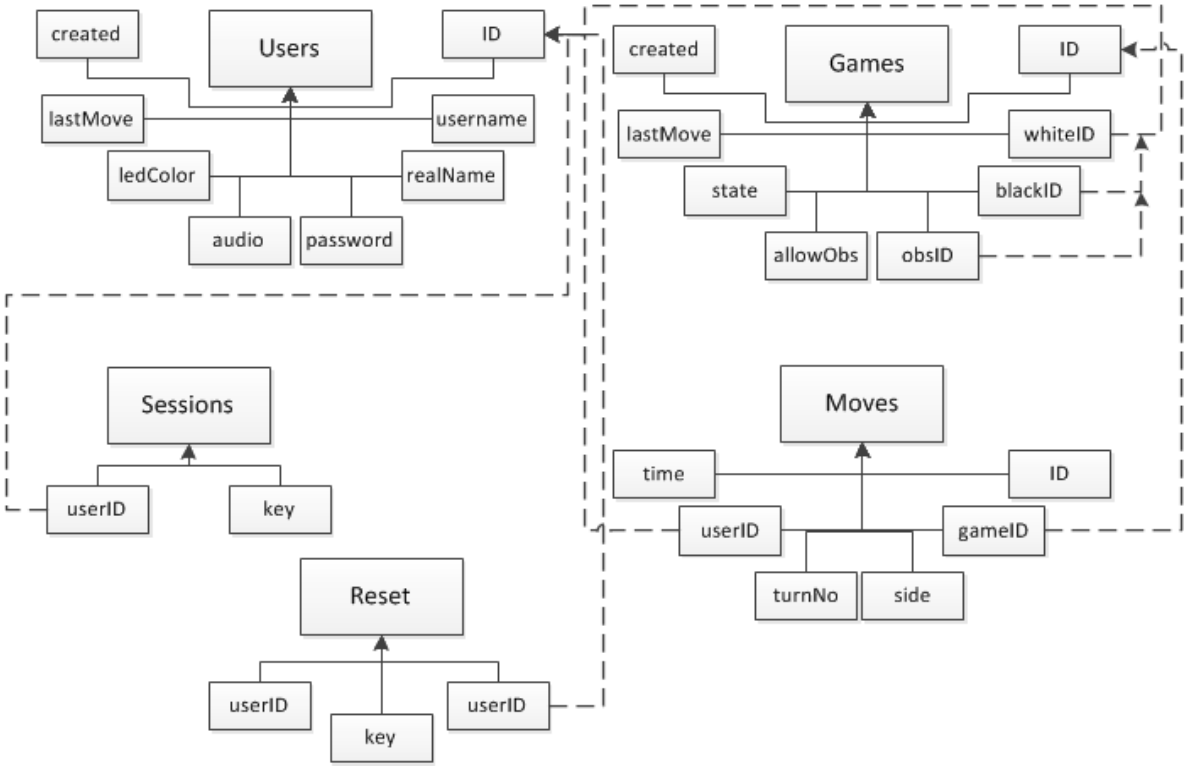


Figure 28 - Database Structure

The final table was the ***reset*** table, another temporary record table that held the password reset keys generated for the password reset system. The records in this table contained a user id, a password reset key, and the date and time that key was created. This table was cleared out on the same schedule as the *sessions* table, even though the password recovery system itself did not allow use of the keys if they were more than an hour old.

The overall structure of the database is seen in Figure 28 - Database Structure below. The four main tables were depicted with their corresponding columns. Important to note are the key mappings (depicted by dotted lines) in the diagram. The mappings from table to table allowed the system to connect data in one table (for example, a move record in the moves table) with related data in another table (such as the game the move was made in, stored in the games table).

4.3.1.2 SQL FUNCTION DESCRIPTIONS

The Structured Query Language (SQL) that Microsoft SQL Server implemented described functions that were to be used when constructing a database query. Table 27 below describes each function, along with its inputs and outputs.

Table 27 - SQL Function Descriptions

Function	Inputs	Outputs	Description
INSERT	Variable	None	Create a new record in a given table, with values as provided.
SELECT	Variable	Variable	Return zero or more rows from one or more tables in the database, optionally subject to filters
UPDATE	Variable	None	Change the contents of zero or more records in a given table to values as provided.
DELETE	Variable	None	Delete zero or more records from a given table, optionally subject to filters

4.3.2 CHESS ENGINE INTEGRATION

The chess engines described in Section 3.2.7.4 offered a variety of functions and skill levels, but since the features required by the server were set in stone there was only a single engine that would work. This engine was the Deep Junior engine, version 13. While Deep Junior was the only program that fulfilled requirements CHS04, CHS05, and CHS06, another engine fulfilled objective CHS05 and was also the most advanced (in

terms of Elo rating) engine available for free. Since Houdini v1.5 utilized endgame table bases to optimize the endgame, it was possible to utilize that engine for maximum computer opponent difficulty after the opening book period.

The chess engine was utilized by the C# management thread to handle processing all moves made by computer opponents. Since the UCI protocol did not require engines to implement opening books, we had to find an engine that supported that on its own in order to not have to implement them in the server software. Being as that chess strategy itself was not a primary goal of this project (and because we were not particularly skilled at it, and would therefore bring the overall skill level of the system down), we attempted to leave all algorithm programming to those who spent time optimizing it. Because of that, not implementing opening books or endgame table bases ourselves was important.

The chess engine, since it implemented the UCI interface, needed to respond to standard UCI commands and reply with a standard format. That allowed for the creation of a C# class that was interchangeable between the two engines, Deep Junior v13 and Houdini v1.5, that we were using for the computer brains. This class was called from a delegate method that was spun off into a new thread by the management program whenever the chess engine was needed.

The chess engines were delegated by a management process that ran whenever the server was active. This process was a very simple construction that managed scanning the database on a regular basis and spawning off a thread to deal with the chess engine when a game required a computer move to be calculated. In addition to checking for computer moves to be made, the management process also performed a simple aging on the *reset* and sessions tables in the database every hour to make sure the records in those tables were fresh. A simple flowchart for the process is available in Figure 29 - Management Process Flowchart and the UML diagram for the classes presented is shown in Figure 30 - Management Process UML Diagram.

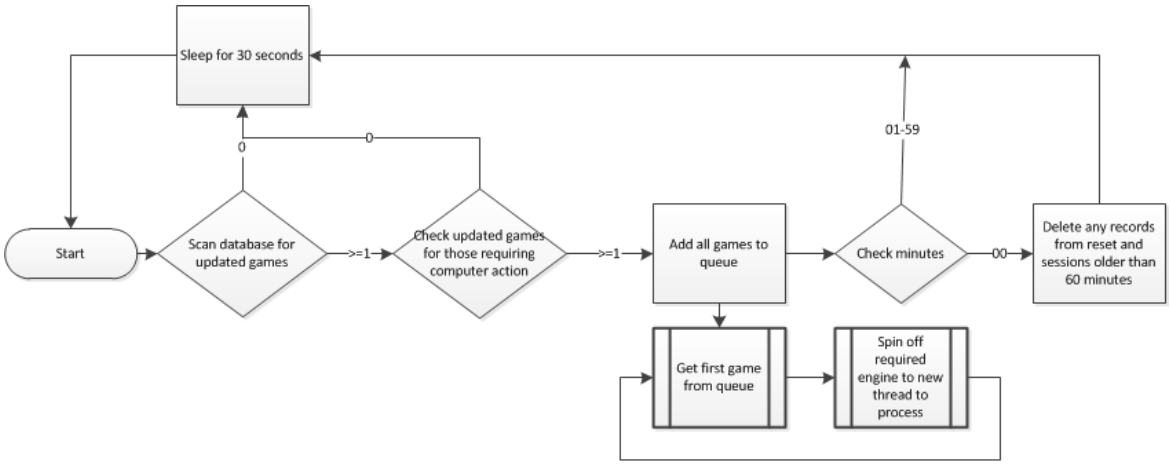


Figure 29 - Management Process Flowchart

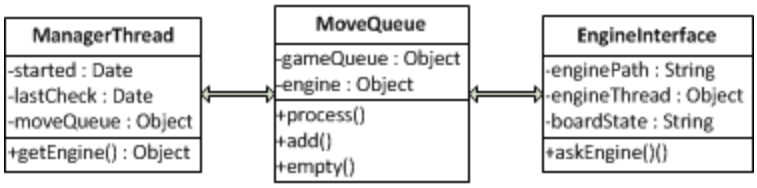


Figure 30 - Management Process UML Diagram

4.3.3 MANAGEMENT PAGES

Both the physical chessboard and the web interface described in Section 4.3.4 utilized the same management pages to control the progress of the game and update the system's database with the current state of the game after a move had been made. There was a separate page for each of logging a player in, getting the logged in user's profile information, requesting the list of games available, requesting the current state of a specific game, creating a new game and updating the database after a move had been made. In addition, there were four administration pages that handled creating new accounts, generating reset keys for users who had forgotten their passwords, resetting the passwords for those users, as well as adding a logged in user as an observer to an in-progress game.

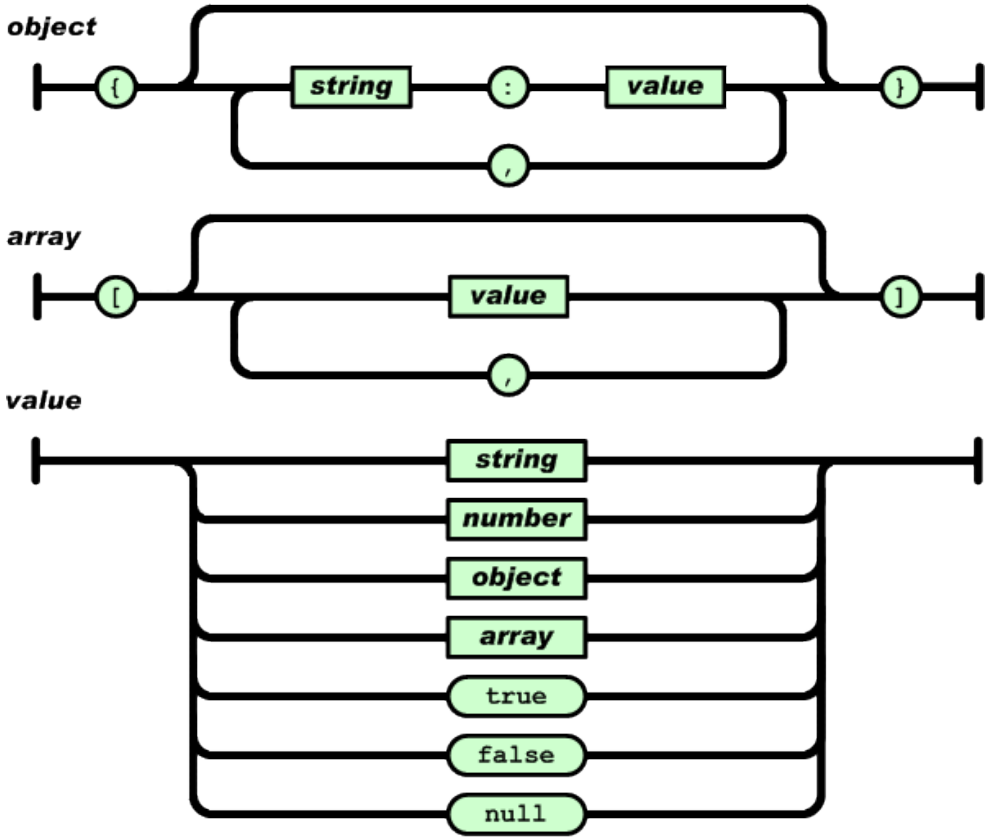


Figure 31 - JSON Object Definitions

Each page was kept separate for two reasons. Firstly, it kept the backend of each page constrained in what it needed to be dealing with, limiting the number of unintended side effects of given algorithms. This was primarily a design benefit, as it allowed the programmer to keep a more defined idea of the purpose of a single process when each process was divided into separate files. Secondly, separate pages for each function limited the amount of information any single page returned which made it easier on the physical board's RAM-limited microprocessor to process entire requests at once.

None of the management pages were made to be human-readable; each was designed instead for machine readers and, therefore, sent and received information in the JSON format. JSON, which stands for JavaScript Object Notation, was a lightweight, platform independent, text-based format. It was human-readable to an extent, which was a bonus for development, but it's format was very formulaic and therefore easy for programming languages to interpret. The format is described in Figure 31 - JSON Object Definitions and Figure 32 - JSON Type Definitions.

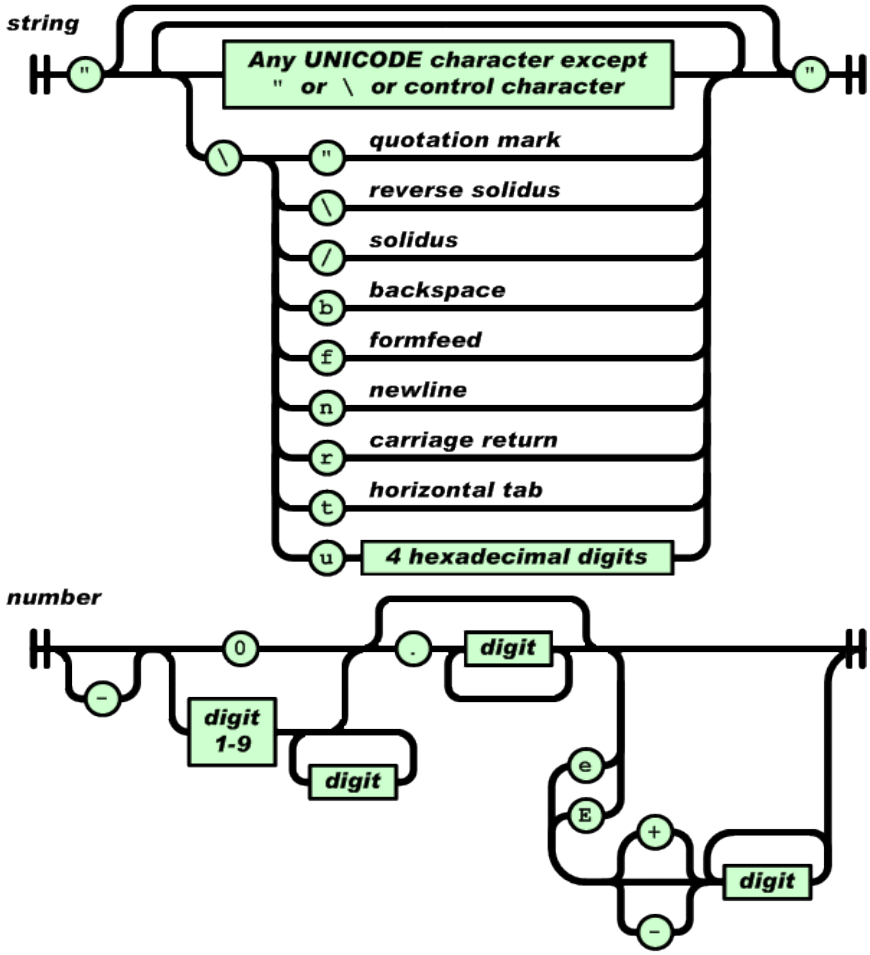


Figure 32 - JSON Type Definitions

A summary of the inputs and outputs of the management pages can be found in Table 28 - Management Page Descriptions below

Table 28 - Management Page Descriptions

Page Name	Required Input	Optional Input	Output	Description
Log In	• Username • Password (SHA-1 hashed)		• Authentication Key • User ID	Logs in a user and creates a session on the server
User Info	• Authentication Key • User ID		• Username • LED Color • Audio Theme	Get information about the currently logged in user.
Games List	• Authentication Key • User ID	• *showObserver* • *page_count* • *page_number*	• Games IDs • Player usernames	Get a list of all games the user can interact with.
Game Info	• Authentication Key • User ID • Game ID	• *moves* • *move_limit*	• Game ID • Players • Board State • Turn Number • Active Player • Turns (Optional)	Get the full state of a game, given its ID.

Create Game	• Authentication Key
	• User ID
	• Opponent Username
		• *allowObserver*	• (Success)
			  o Game ID
			  o Players
			  o Board State
			  o Turn Number
			  o Active Player
			• (Failure)
			  o Error Code
			  o Error Message
				Make a new game, given a logged in user and an opponent.

Page Name	Required Input	Optional Input	Output	Description
Update Move	• Authentication Key • User ID • Game ID • Board State • Move Made		• (Success) o Game ID o Players o Board State o Turn Number o Active Player • (Failure) o Error Code o Error Message	Submit a move in an ongoing game.
New Account	• Username • Password • Real Name • Email Address	• LED Color • Audio Theme	• (Success) o Success Code • (Failure) o Error Code o Error Message	Create a new account using the information provided.
Pass Lost	• Username		• Email containing reset key	Request a reset key for a user who has lost their password
Pass Reset	• UserID • Reset Key		• (Success) o Success Code • (Failure) o Error Code o Error Message	Choose a new password for the user
Add Observer	• UserID • Authentication Key • GameID		• (Success) o Success Code • (Failure) o Error Code o Error Message	Add a user to a game as an observer.

4.3.3.1 LOG IN PAGE

The first management page requested by both the physical chess board as well as the web interface was the log in page. The log in page took as an input a POST request from a web client with the parameters to include a plaintext username and a SHA-1 hash of

that user's password. It then took the plaintext username and applied a SHA-1 hash to it as well before concatenating the result with the password's hash. The salted password hash was then hashed once again with SHA-1. The resulting string was compared to the contents of the user's password as it was stored in the *users* table of the database. If it matched, a response was created containing the user's user id number and an authentication key that had to be resubmitted with each other request. This authentication key was concatenated with the user's remote IP address and then hashed and stored in the table. Further requests to management pages were required to submit the authentication key correctly and come from the same IP address, or else the management page being requested would return a status code 403 - Forbidden error. The system could not use the more standard method of storing authentication via client-side cookie relating to server-side sessions because the chess board was incapable of storing cookies.

4.3.3.2 USER INFORMATION PAGE

The second page that both clients requested was the user profile information page. This page took as input the authentication key provided by the log in page and the logged in user's user id and returned the user's profile information, including display name, LED color choice, and audio choice. The decision to store this information or request it again when it was needed again was left up to the client. The user profile information was of such small size that the bandwidth involved in serving it was miniscule.

Because the user information page needed to return the user's profile, it made a direct connection to the *users* table of the database. It required only read (*SELECT*) access to the table, and was therefore a relatively safe operation.

4.3.3.3 GAMES LIST PAGE

The first of the real workhorse pages was the games list page. It replied to a request containing an authentication key and a user id number with a list of games that user was involved in (observing or playing either black or white) and the players of those games. The page also supported a query string parameter, *showObserver*, to showed all games that were open to observers which would not be utilized by the physical chess board but by the web interface to allow for setting up observer status for a game currently in progress.

The games list page accomplished this task by connecting directly to the *games* table of the database and issuing a *SELECT* query with *WHERE* clauses searching for the user's user id in the *whiteID*, *blackID*, and *obsID* columns. Every game that was returned by that query was packaged into a JSON object and returned as the response to the initial request. Since this was potentially a very long response, another two query string parameters were available for use (primarily by the physical board), *page_size* and *page_number*, allowing for paged reading of the games list.

4.3.3.4 GAME STATE PAGE

The game state page required input of an authentication key and user id number from an authenticated user, as well as a game id number for a specific game. The page then queried the database's *games* table for information related to the given game. Finally, returning the game state, turn number and active player, as well as the display names for both players.

The page also offered other options accessible through query string parameters. With the *moves* parameter set to true, the page also queried the *moves* table and returned the series of moves that had occurred to date in the given game. With *move_limit*, the client could limit the number of turns returned if, for example, only the last five turns were desired.

4.3.3.5 NEW GAME PAGE

The new game page required an authentication key and the user id that corresponded to it, a bit representing the user's starting side, as well as another username. The second username is the name of the second player that was to take part in the game, who will then take the side of the match opposite the creating user. The *users* table was queried for the user id number of the user who's username was given.

If no user id number was found, then the user submitted a fictitious username, and a status code 200 response with a JSON object consisting of only an "error" object with a string value explaining the error in plaintext and an error code representing the lack of user. If, however, the user was found in the *users* table, then a new game was created by inserting a new row into the *games* table with the given users' id numbers. By default, the *allowObs* column was set to false, however a query string paramater, *allowObserver*, can be set to true to force the new game to allow observers. The new game page then requested the game state page for the newly created game and sent the information back as the response.

4.3.3.6 UPDATE MOVE PAGE

The final workhorse page was the update move page. This page took as input the authentication key and user id number of the logged in user, as well as a game id number, and finally the new state of the board and the move actually made. The page would first check to see if the move being submitted was a legal move according to the rules of chess, and then insert a new record into the *moves* table with the relevant info and then update the *games* table to advance the turn (if necessary), switch active player, and set the last played field to the current time. It then returned the response from the game state page for the updated game, similarly to the new game page.

If the move was illegal for any reason, an error code was generated along with a plaintext description of the error and this object is returned to the user.

4.3.3.7 NEW ACCOUNT PAGE

The new account page was the first of the administrative management pages. The input was all of the information required by the *users* table, namely username, real name, email address, password, and optionally LED color and audio theme. The password submitted to this page had to already be hashed using SHA-1 (as the web-based interface did) as a security precaution. The username, email address, LED color, and audio theme were all added in plaintext to the database, but the username was also hashed with SHA-1 as a salt, then the password hash and username hash were concatenated before being hashed a final time with SHA-1. This doubly hashed, salted string was then stored in the database. This was the only point in the server system that the hash in the user's password file was created, and along with the password reset page, was the only place it changed at all. The page responded with either a success message or error message in a JSON object.

4.3.3.8 PASSWORD LOST PAGE AND PASSWORD RESET PAGE

The password lost page was part of the backend to the web client's password recovery system, as described below in Section 4.3.4.2. The page's function was to create the password reset keys that were required by the password reset page and store them in the *reset* table along with the user id number of the user that requested the reset, and the time the request was made. The reset keys provided by the password lost page were only active for 30 minutes. The reset key was sent in an email to the email address stored in the database for the user. The email contained a link to the web-client's password reset page that was the front end for the password reset management page.

The password reset page was the management page that took the reset key provided by the password lost page and the user's new submitted password and determined if the they key was valid. If so, the new password was hashed as in the new account page and stored into the *users* database in place of the old password.

4.3.3.9 SET OBSERVER PAGE

The final administrative page was the set observer page. This page asked for an authentication key and user id number from the logged in user, as well as a game id number. The page then checked to see if the game in question was set to allow observers, and if so, added the user to the game as an observer. The page responded with either a success message or error message in a JSON object.

4.3.4 WEB INTERFACE

The only non-board interface provided for Deep RGB in this project was the web-based board. The web application allowed a user to connect via username and password and

then access the database to display all games the user was involved in. The games were listed in an easy to navigate manner allowing the user to select which game he would like to play. A table area was then created displaying, by default, the games the user was involved in that were currently in-progress. A small area to the side of the interface allowed the user to instead display all games that he had been involved with, in-progress or not, as well as sort the games listed by opponent, time the last move was made, time the game was started, or total number of moves. This area also linked to the profile editor page described below.

Once a game was chosen, the user was navigated to a page displaying the current state of that game on a chess board graphic. If it was the user's turn to make a move, a notification was made of this and the user's chess pieces were interactive. Selecting a piece would display that piece's available, legal moves by highlighting the squares to which it could be moved. The user then selected a square to make the given move, or re-selected the piece to de-select it and return to the default state of the board. Once a move had been chosen, the page submitted this action to the server by making a request to the same page the physical board used.

A second page available from the main log in page allowed the user to edit his or her profile. The profile stored the user's display name and real name, as well as the color the physical chess board used to identify that user and the audio the board played when that user was playing. The display name was a simple (less than 15 character) nick name that was displayed to the other player in any games the user was involved with, as well as observers to those games. The user's real name was requested so as to greet the user on log on to the website or physical board. The color used to identify the player was editable on the profile editor as well by selecting hexadecimal values for each of red, green, and blue to choose a specific color and hue. A color selector was available to assist the user in choosing a color. Finally, the profile editor page allowed for a dropdown choice of audio themes for use on the physical board.

4.3.4.1 LANDING PAGE

The landing page for the web application was the initial log in page. Log-ins were simple username and password combinations with both username and password chosen by the user. The elements of the landing page included text fields to allow the user to enter their username and password, a submit button to send the log in information to the server, along with links to create a new account and reset the password, if the user had forgotten theirs. Unless a user had logged in using this page, every other page in the system would redirect the user to this landing page to request that he or she logged in. A mock up of the page is found below, in Figure 33.

The landing page backend incorporated the secure sign-on algorithms to allow for a user to input his or her username and password only once per session and maintained a connection to the server for the remainder of his or her play time. When the page was submitted, the username and password were sent to the server via the log in

management page, described in Section 4.3.3.1. An authentication key was then created and stored in the user's session, allowing him access to the remainder of the web application.

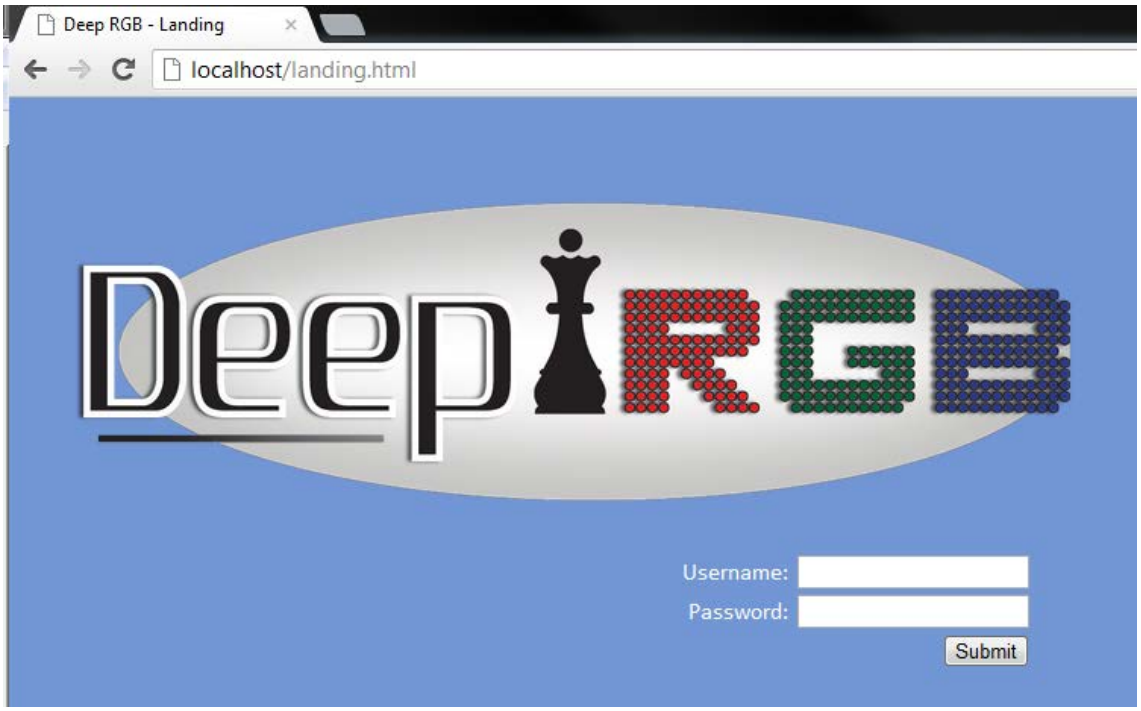


Figure 33 - Landing Page Mockup

4.3.4.2 PASSWORD RECOVERY AND RESET PAGE

The link from the landing page to recover the user's password took him or her to the password recovery page. This page requested the user's username and email address. If the data provided by the user matched a record in the users table of the database, an email was sent to the user's email address with a link to another page that contained an authentication key good for only one use and only 30 minutes. If the user did not use the link to navigate to the password reset page within the time limit, the authentication key no longer functioned and they would be unable to reset their password without requesting another password reset email. If the user did access the password recovery page within the given time limit, they were offered a single text field and allowed to submit a new password. That password was hashed and added to the user's record in the users table of the database and the user then redirected back to the landing page where they were able to log in with their username and the new password.

4.3.4.3 ACCOUNT REGISTRATION PAGE

A new account was only creatable from the web interface. A potential user of the physical chess board was required to access the web interface at least once to establish

a profile. The account registration page was accessible from the landing page and allowed an un-registered (or registered, if they desire a new username) user to create an empty user account for use on the system. All relevant user information (username, password, email address, LED color and audio theme) were available as fields to fill out, with only username, password, and email address being required. If submitted, the LED color and/or audio theme was then stored in the user's profile, otherwise a default color would be chosen for the LED System and no audio theme would be used for that user. A mockup of the account registration page is seen in Figure 34.

The account registration page utilized the LGPL-licensed JSColor library provided freely at http://jscolor.com/. This library allowed for the attachment of JavaScript color pickers onto text fields that allowed for the choice of a hexadecimal color value by visually inspecting the color spectrum available. The library was incorporated into the page through the use of a single JavaScript file (jscolor.js) that was installed into the website root along with a few helper files and used by appending a special class onto any <input> tags that require the color picker.

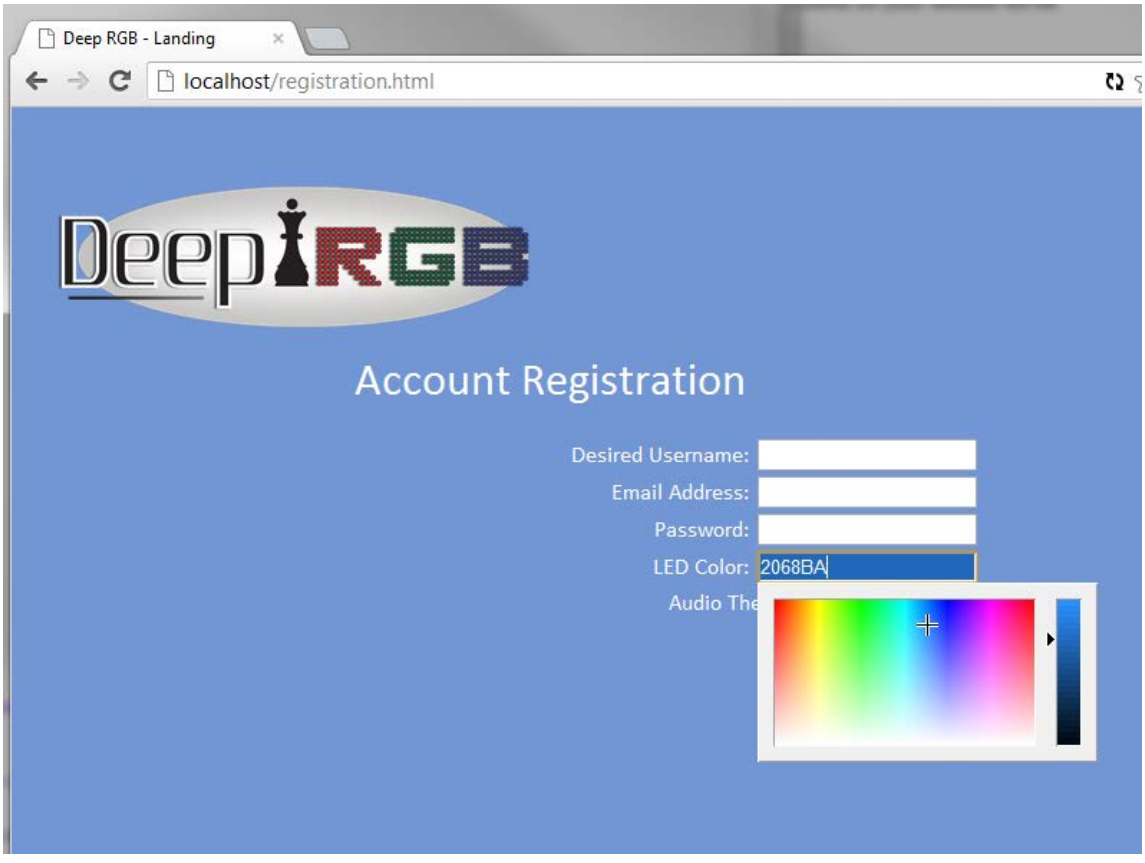


Figure 34 - Account Registration Mockup

The profile edit page was a derivative of the account registration page. It requested the same information (except username and password), and submitted that information to

the new account management page with the authentication token received from logging in. In this case, the new account management page would only update the corresponding row in the *users* table instead of creating a new one.

4.3.4.4 GAMES HOME PAGE

The games home page was the primary landing screen for a logged in user. The main part of the screen displayed a list of games that were active and included the logged in user, by default. A filter checkbox was available in the sidebar to also display all games the user was not involved with but were open to observers. The games home got this list from requesting the games list management page with appropriate query string. Each item in the list was linked to open the game that it referred to, as either a player or an observer, as was appropriate. If the Show Observable filter was on and the user clicked on a game they were not involved with currently, they automatically request to be added as an observer and were taken to the game page thereafter. A mockup of the games home page is available below as Figure 35 - Games Home Mockup.

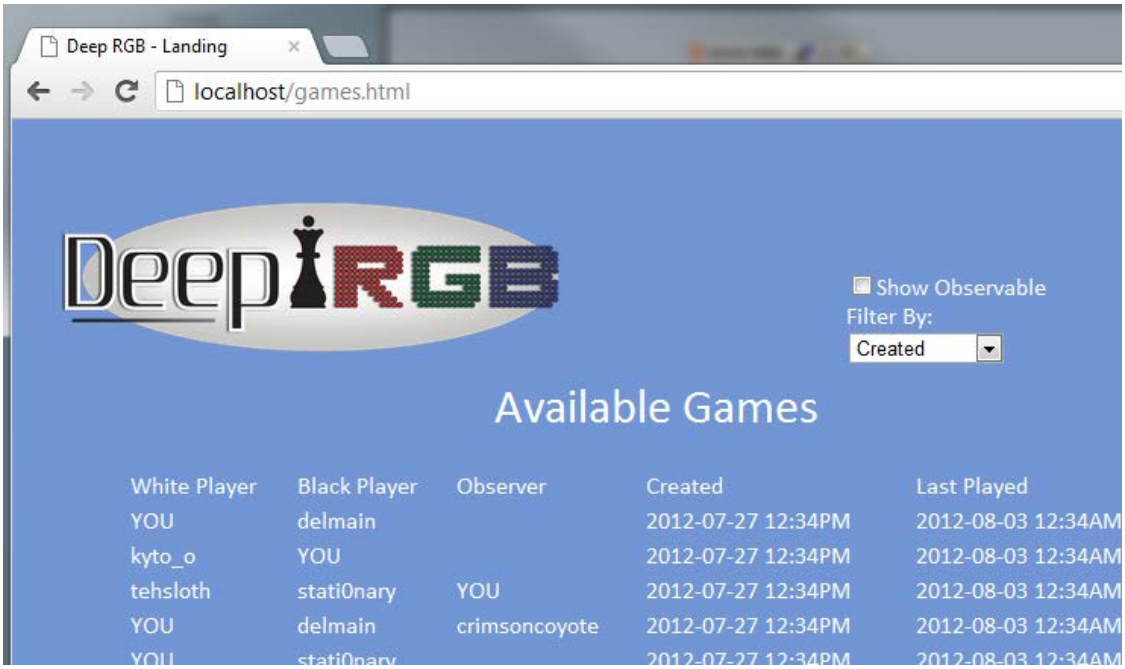


Figure 35 - Games Home Mockup

4.3.4.5 ACTIVE GAME PAGE

The active game page was the workhorse page of the web client, handling all the actions involved in playing a game of chess. This page used primarily AJAX (Asynchronous JavaScript and XML) requests to get updates from the server on the state of the game as well as send updates to the server. JavaScript functions were assigned dynamically to squares on the virtual chess board so as to have a fluid and well-mannered interface to

the chess game. The variables in use were defined in Table 29, and the functions used defined in Table 30 below.

Table 29 - Game Page Variables

Variable Name	Type	Description
currentState	JSON Object	Holds the current state of the board in the same format as is returned by the game state management page.
board	string	The board's current state, expressed in Forsyth-Edwards Notation (FEN).
activePlayer	bool	The player who's turn is active. True is white, false is black.
turnNum	int	The current chess turn number.
activeCellX	int	Coordinates of the currently active cell. Both set to -1 when no cell is being selected.
activeCellY	int	
activePiece	string	The piece located in the currently active cell.
refreshTimer	int	Number of seconds between calls to getState() when user is not active. Defaults to 30.

Table 30 - Game Page Functions

Function Name	Inputs	Returns	Effect
getState()			Requests a full state update from the game state management page. When the request is fulfilled, it sets the *currentState* variable, calls *unpackState*() to unpack the JSON into their respective local variables, and calls *displayState*() to refresh the view.
unpackState()	currentState	board activePlayer turnNum	Read the JSON object stored in *currentState* and extract the board state, active player, and turn number and store them in local variables for easier manipulation.
packState()	board activePlayer turnNum	currentState	Reverse of *unpackState*(). Take the local variables and pack them into a JSON object for return to server.
displayState()		View	Read the state saved in the *currentState* variable and sets the page up correctly to display that state at the beginning of the current player's turn

Function Name	Inputs	Returns	Effect
selectCell()	Coordinates	activePiece activeCellX/Y	The default action on every square containing the active player's chess pieces to start. Activating sets the active variables and disables *selectCell*() on all other cells. Enables *deselectCell*() on *activeCell*. Calls *showMoves*()
deselectCell()	Coordinates		Resets the active variables and places *selectCell*() back on all cells containing the active player's chess pieces.
showMoves()	activeCellX/Y activePiece		Highlights all cells the active piece can move to from the active cell. Does not display moves that are illegal (leaving yourself in check). Activates *makeMove*() on highlighted squares.
makeMove()	Coordinates	success (boolean)	Attempts to make the move selected by submitting a request to the update move management page. Will first call *packState*() to package the current state and chosen move into a JSON Object to be sent. Will receive the success or error code returned from the management page and handle properly.
refreshState()	refreshTimer		Active when user is not active player. Will call *getState*() every *refreshTimer* seconds.

A very rough mockup of the game page is available as Figure 36 below.

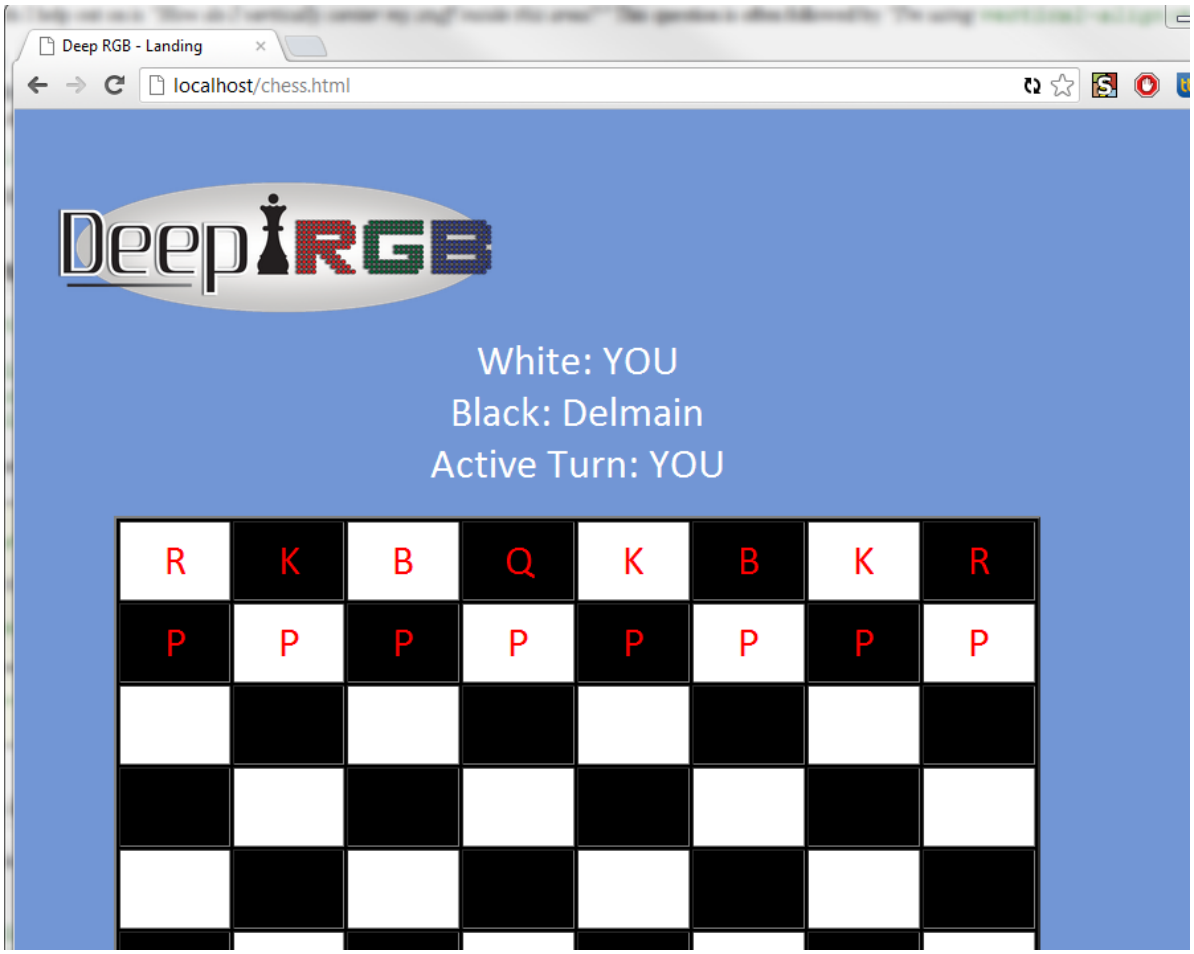


Figure 36 - Game Page Mockup

4.3.5 SERVER CONFIGURATION

The Deep RGB server was to be run on a temporarily donated PC that was owned by a group member. As such, it's hardware specifications and some software specifications were set. As mentioned previously, the platform was a Windows machine, running Windows 7 Service Pack 1, specifically. The software necessary to run the server is explained in Table 31 - Server Hardware and Software below, along with the hardware specification of the server.

Table 31 - Server Hardware and Software

Item	Manufacturer	Version/Value	Description
Server Operating System	Microsoft	b7601	Windows 7 with Service Pack 1
Http Host	Microsoft	7.0	Internet Information Services 7.0
.NET Interpreter	Microsoft	4..3039	.NET Framework 4

Item	Manufacturer	Version/Value	Description
Central Processing Unit	Intel	Q9400	Intel Core2-Quad. Four cores operating at 2.66GHz
Random Access Memory	OCZ	8GB	8GB OCZ Gold Series DDR2 RAM

4.3.6 CLIENT CONFIGURATION

As a web-based application, the client configuration required to interact with the Deep RGB web interface was very basic. Almost any modern device would have been able to connect to the web interface and play a game. The server for Deep RGB was hosted externally in Orlando, FL through a dynamic DNS based URL, allowing for connection through a simple URL (for example: http://deep-rgb-ucf.com/). Any user wishing to use the web interface to Deep RGB needed only to enter the URL into their web browser and they would be sent directly to the landing page, ready to log in.

The minimum specifications for access to the Deep RGB web interface were very lax, as it was designed to be compatible with the majority of users. Most users interested in the system were already running on a somewhat modern computer. Most modern desktop and laptop computers had all the software necessary to access DeepRGB without any further installation. The minimum support specifications are listed below.

Supported Operating Systems:

- Windows 2000/XP
- Windows Vista/7
- Mac OSX v10.5 "Leopard" or newer

Supported Desktop/Laptop Web Browsers

- Microsoft Internet Explorer 6 or later
- Mozilla Firefox 3.6 or later
- Opera Software Opera 7 or later
- Mac Safari 3 or later
- Google Chrome 3 or later

Support Mobile Browsers: (All mobile browsers had to be entirely up to date for phone compatibility issues)

- Google Android Browser
- Google Chrome Mobile
- Opera Software Opera Mobile

5 TESTING AND PROTOTYPING

5.1 HARDWARE TESTING

5.1.1 MICROCONTROLLER UNIT

To test the microcontroller and it's IDE we needed to write code for subroutines that tested every pin available. In order to perform these tests it was recommended that the microcontroller be connected to the PC. This was to allow any error codes to return to the computer and be displayed on the screen. To test the IDE, we entered some incorrect segments of code to ensure the debugger was working as intended. To test the PWM pins we connected them to several LEDs and wrote code in order change the colors on them. We continued to change the code around several times to ensure that the LEDs could both change color and dim which was a good indicator that the PWM pins were working correctly. Since PWM had 256 levels of sensitivity, the output for every level would increase by approximately 0.0195 volts and show 5 volts output for a level of 255, we tested this with a multimeter just to double check that they worked even after the first test. In order to test the digital I/O pins we used a digital multimeter to observe the change in the voltage while we programmed each pin on or off. We observed the multimeter to ensure that it measured 5V for digital HIGH and 0v for digital LOW, assuring that the digital pin-outs worked as intended. The UART pins werr tested in conjunction with the wireless connection device since they used the UART pins to communicate. We were fairly certain that the WI-FI module would work correctly as it was from a company that manufactured plenty of other reliable products. Therefore we connected the UART pins to the WI-FI module and tested them by running a test code through them to the module once we saw that the module functioned as expected we knew that the UART pins were working. As for the SPI bus, the audio module was used to test these pin-outs since it used the complete SPI bus in order to send and receive data from the device. However later integration testing with the Hall Effect grid proved that the SPI could not run both the grid and the audio at once since the audio needed to be maintained most of the time.

5.1.2 HALL EFFECT SENSOR TESTING

In order to test the Hall Effect sensors several were ordered from the internet and wired into a breadboard. Once connected a single sensor was connected to the microcontroller and powered up. Then we used a piece with a magnet inside to test the sensor by moving it around the sensor. We observed how the sensors reacted on the program and gained some insight into how well they worked. To get a better idea of

how the sensors would function in the actual board we placed four in a grid connecting them all to the MCU. Then we re-performed the tests to observe how magnets would affect multiple sensors near each other. After this test we realized that the original sensor were in fact not sensitive enough and we needed to acquire better sensors.

Once we had received the new sensors we once again performed the tests mentioned above. We used this data to determine how far away the new sensors could detect the magnets inside the pieces as well as several other magnets we had purchased for testing. Once we determined that the pieces could be detected in the center of a tile reliably we knew that we had the right sensor which was the A1325 from Allegro. When we began to build the board we decided to incorporate a mobile sensing unit by attaching four sensors in an array mounted to the movement system. Since the sensing unit was mobile we decided to reduce the number of sensors by placing one in the center of each tile instead of every corner. By the time we had switched to electromagnet out for a solenoid we had already created the sensor array and so we decided to use it as it was. After the sensor array was completed we had to recreate the tests with the entire unit connecting to the MCU through the MUXs. We determined that the board could still function with the new arrangement and therefore decided not to change it back to the original design.

5.1.3 AUDIO TESTING

The audio shield was originally used to test the SPI pins on the microcontroller we had already begun to test it. We used the MCU to send the audio module a few orders and observed that the module carried them out. Unfortunately when we performed some integration testing we ran into a problem, the SPI bus was incapable of controlling both the audio module and the LED matrix. It simply required too much of the MCU's time and therefore the controller could not control the matrix as well. We researched to possibility of fixing the problem with a second microcontroller to run the audio unit by itself unfortunately this would have added too much to the cost of the project and was therefore not included in the final project.

5.1.4 LED TESTING

In order to test the LED array and make sure that it worked correctly, sufficient knowledge about its components was necessary. To begin we tested the LEDs by placing one into a breadboard and connected potentiometers to each anode. Then we powered the LED through the potentiometers and varied the resistance so that the power supplied to the LED varied with it. We used the potentiometers to hone in on the resistance was needed to provide the correct current level to create a white colored light from the LED. We had to do this since the forward voltage of the color red was lower than the other two colors therefore supplying five volts to each anode would have made the red shine brighter than the others. Once we achieved a white colored light we

measured the resistance of the potentiometers at that time and verified it against our calculations. The resistor values we discovered were 150Ω for red and 90Ω for blue and green, this ensured that each color got the same current. After we discovered what resistor values we needed we purchased a small LED matrix from Ebay that had the same circuit design as the one we were using. With this matrix we created a circuit using the shift registers to connect the MCU to the LEDs then on the MCU we created a program that could turn on specific LEDs. Once we verified that we could do this we continued to develop the program until eventually we were able to turn on individual LEDs as well as change their color and brightness. This was all the testing performed before the board was created , afterwards we used the full board with the same code to verify that it was working correctly. At first we had some problems with entire columns turning on but realized that it was a timing error in the program and we were able to fix it. Another problem we ran into was an overall dimness of the LEDs, they still lit up the board but were not as bright as we would have liked. Unfortunately could not find a solution to this error and decided instead to focus on other more important functions of the project.

5.1.5 POWER SUPPLY TESTING

In order to test the power supply appropriately we had to wait until we had created the board to be truly sure that it would work effectively. Unfortunately the power supply was one of the components that was to be integrated onto the PCB and required surface mounted parts. After we could not find a company to populate the PCB for us we tried to solder the parts on the board ourselves which proved to be too difficult. Therefore we purchased the third party power supply mentioned in the corresponding design section of this document. Since we knew that the part was coming from a reliable company we were pretty sure that it would work as stated. To be sure we connected it to 120V AC power and then used a multimeter to test the power levels being supplied from the unit. The final test was to connect the power supply to the completed system. After it was connected we used it to power the board and continued to check each subsystem with a multimeter to ensure that they all received the correct voltage and amperages.

5.1.6 DISPLAY TESTING

The display we were using was not connected directly to the MCU because the display had a pre-installed display driver which took care of most of the operations required to display text on the LCD screen. All we needed to do to make display work was to connect display unit data inputs to the MCU and connect the remaining display pins to the power supply and to the ground, as directed by the display driver instruction set. Once the display had been connected we used the MCU to write strings to it. We started with a simple "hello world" and once that had been done we continued to write different strings to the display to test the limitations of what we could use it for. After

we had the display printing the messages that we wanted we used the MCU to test the backlight on the device. Once we saw the backlight and how well it worked we decided to hardwire the component so that it always stayed on, eliminating the need for a photo-resistor and/or additional programming on the MCU.

5.1.7 WIRELESS MODULE

The Wi-Fi module came with a testing procedure to ensure that a secure and strong connection was made to a server. After connecting the UART cables to their proper location, a test code was downloaded from the manufacturer's website. While connected to a computer, this test checked the firmware of the module to ensure that was updated to the most recent version. The test code then dumped the memory of the wireless module and entered a command mode. When the command mode was activated, we allowed it to scan for a network and try to connect to one. When the connection was made, the network's information was printed along with the module's IP address and MAC address. If the connection got timed out or fails for any reason, a corresponding error was printed that explained the reason for failure. When a connection was made, the module could be commanded to ping an IP address to ensure packets are being sent and received. After we had tested the original code we then used our own code to send packets between the MCU and the server through WI-FI. At first we ran into some problems with the communication due to the size of the buffer that was used. The problem was fixed by rewriting the code so that it used a larger buffer and could send larger commands at once, afterwards the wireless module worked fine.

5.1.8 MAGNETIC PIECE-MOVER TESTING

A properly functioning magnetic piece-moving system was an important part of Deep RGB. We needed to make sure that this system satisfied all requirements, and this required that we use a detailed test procedure wherein we checked all working elements of the magnetic piece-moving system. To begin we tested the stepper motors themselves using a DC power supply to ensure that they both spun and neither was defective. We then built the base of the project which included the rack gear two glides and the mounting arm for the motors that was described in the design section. After we soldered the motor connectors to the drivers and then connected the drivers to the MCU we had to test the system together. We used the MCU to send commands to the motors through the drivers and ensured that the system worked as a whole and that both motors moved correctly without making too much noise or vibrating too much. There was a problem with noise at first, however they were reduced by adding a few weights on the end of the X axis motor as well as lubricating the guides properly. Since the case of the board was made out of plywood there was no way to make it perfectly straight and the slight irregularities in the surface made the rack and gear system vibrate and make noise.

After the stepper motors had been tested we moved on to test the electromagnet however this resulted in the realization that the electromagnets available would not work. Hence we decided to swap the electromagnet with a solenoid that moved a piston with a permanent magnet affixed to it. After this decision was made and a solenoid had been purchased(from skycraft), the solenoid was affixed to the side of the X axis motor. Before the completed setup could be tested we had to finish building the case of the board including the playing surface. Afterwards we used the MCU to turn the solenoid on and off to ensure that it worked and then used it to grab a piece and move it across the board. After we verified that we could move a piece we began placing more pieces on the board and attempted moving pieces around each other. Here we ran into the problem of pieces attracting each other and pulling pieces that were supposed to stay stationary with them. This problem was solved by adding a ring of fishing weights to each piece to increase the weight and reduce the chances of the pieces pulling each other. After we were able to move a piece between two pieces in adjacent tiles we installed two switches on the movement system and used them to set the origin of the XY coordinate system. To test them we ran the X and Y motors in the direction of the switches until they were flipped at which point we turned the motors off and verified that the switches were giving off an active reading. The final test we performed on the movement system was running the motors through their complete range so that we knew how many motor steps made up the different axes. The X axis could be traversed in 989 steps while the Y axis was made up of 635 steps, we used these values to program the software side of the movement system.

We ran into a problem once the board had been completely assembled and that was how to keep the wires which controlled the motors from falling onto the racks and being torn apart by the gears. We planned to solve this problem by using two or three retractable key chains to create a varying length "net" behind the armature that would grow longer or shorter depending on where the arm was giving the wires something to lay on. This proved to be a bit overzealous of a solution as it created the potential for pulling the arm one way or another and interfering with the movement system. Therefore we simply mounted the wires securely onto the armature and routed them in such a way that it was unlikely that they would get in the way.

5.1.9 PCB

Since our PCB was designed using Eagle Cadsoft, were able to implement testing pads. Testing pads were small conductive circles that were used to ensure that a pathway was conductive and unobstructed. These conductive test pads as seen in Figure 37 allowed us to use a multimeter with a short circuit function in order to check if the circuit was being completed. This was a primitive way of testing a circuit, because a pathway could show up as a short but still warp a signal passing through it. PCB manufacturers tended to test for these faults by using two probes and sending a signal pulse through the pathway. If the same signal was received on the receiving probe, then the pathway was unobstructed. If the signal gets warped along the way then there could be some sort of

interference in the pathway. We also supplied the new MCU on the PCB with the same code that we had tested previous sections with and observed that they resulted in the same events. This told us that at least for the functions we had tested so far that this PCB would work and we believed that the PCB was therefore functioning properly.

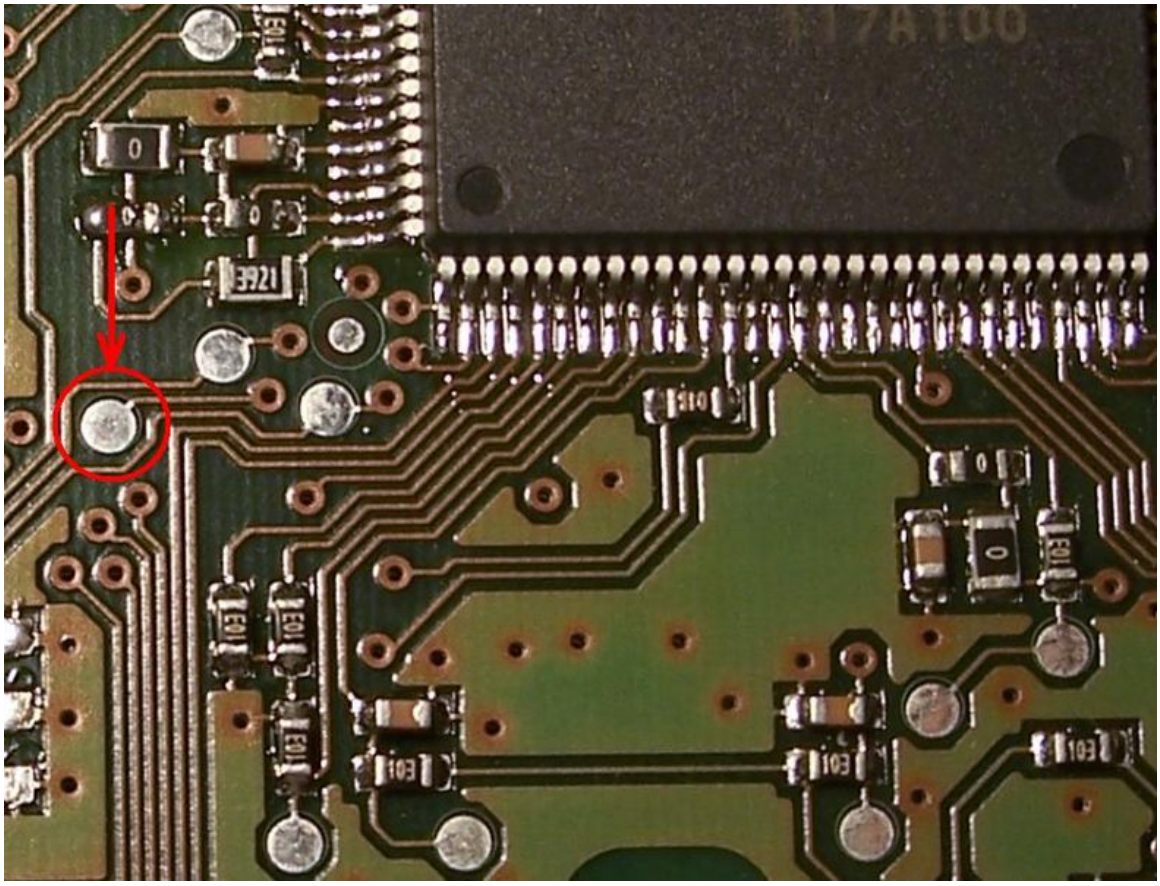


Figure 37 – A close-up on the testing pads used to test pathways on PCBs reprinted with permission from Wikipedia.

To further test the functionality of the PCB we had to wait until the board been completely built and all of the parts integrated. Once this happened we were able to test the PCB with the entire system. At first we had some issues with the timing which was evident already with the LED subsystem tests. This simply had to be overcome at the end of the programming process, once everything was programmed the timing between the shift registers and the MCU was tuned until the LED array functioned properly. Something we had not considered when making the PCB was how much space it would actually need this created a wiring problem although not a very serious one. We had a problem with the wires connecting the Hall Effect sensors to the MUXs pushing out too far and falling onto the glide of the movement system. To solve this

problem we placed an extra piece of plexiglass in between the PCB and the movement system which kept the wires bent up and away from the glide.

5.2 SERVER TESTING - 1 PAGE

Software testing of the server system was both incredibly important and incredibly easy because of the management page set-up utilized. Since each component was already sub-divided to a single task for a single file, there was no real risk of cross-contamination between processes since each occurs in its own space.

5.2.1 MANAGEMENT PAGES

Each management page was tested separately through the use of custom built HTML pages supplying the required info through an HTML form with the POST action making a request to the management page that was being tested. For every page that required an authentication key and user id number (all but Log In, New Account, Pass Lost, and Pass Reset), the first test to be performed was that of submitting to the page without a working key-id combo.

5.2.1.1 LOG IN PAGE

The log in page had a simple test procedure made up of three tests. The first test was submitting a correct username and password hash to verify that it returned code 200, ok response, as well as the correct user id and a functioning authentication key. The second test was to do the exact opposite and send an incorrect code and ensure that it responded with a 403 error which was the forbidden response error. The final test was to send a valid user id and a password from another user to make sure that a user could not sign in with the wrong password but one that was in the database, it should return the 403 error again. All of these tests proved to run fine the first time and the login page was functioning properly. An image of the login screen rejecting a false username and password was supplied in Figure 38.

5.2.1.2 USER INFO PAGE

The first test for the user info page was simply requesting the page and verifying that the JSON object returned contained the correct profile information for the logged in user, the test was a success. The second test was to request the page although without a valid current authentication key to ensure that the page could not be accessed without authorization. When the test was performed the server responded with a 403 error which was exactly the response we wanted and proved that the page was working correctly.

Figure 38 – An example of the response given by the server when an invalid username or password was entered on the login page.

5.2.1.3 GAMES LIST PAGE

There were two tests required for the games list page. The first was requesting the page while logged in, without passing the optional showObserver parameter, but with using the page_count and page_number parameters. The result received was a group of pages which contained the number of games that the user was already playing which was a success. The second test was with the showObserver parameter and a list of games that the user was not playing but could observe was returned showing that this part of the page worked. The final test was to try and gain access to the page with an invalid authentication key, when tried the page returned the 403 forbidden response error.

5.2.1.4 GAME INFO PAGE

The test for the game info page was also three-fold. The first test was to check for the info being returned correctly without either parameter turned on since the game state was received as expected without a moves array, the test was passed. The second part of the test was to send the same request but with the moves parameter set to true and the moves_limit parameter set to a value higher than the number of moves made; when the same game state was received along with a moves array containing each move, the test was a success. The authentication test was also successfully applied to this page verifying that the page could not be accessed without permission.

5.2.1.5 CREATE GAME PAGE

There were two tests for the create game page. The first was an expected failure by sending a non-existent username and resulted in a 403 error which meant the test was a success and the page behaved properly. The second request to the create game page was with a correct username the response was a game state JSON object containing a brand new game which meant that the test was passed. A screenshot of the create game page was supplied in Figure 39.

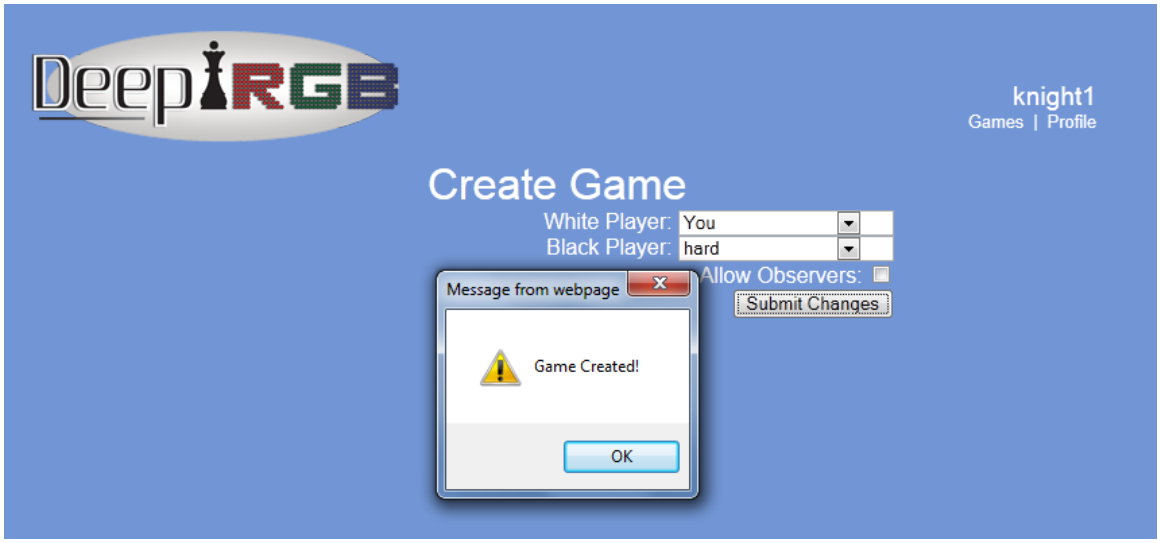


Figure 39 – An example of the create game page working successfully.

5.2.1.6 UPDATE MOVE PAGE

There were several tests required for the update move page. Each parameter in turn had to be set as a failure condition with the rest correct to ensure any failure point returned the correct error code and message. This was a longer process since it involved several different variables that could create a failure, for simplicity they were not listed here however the test was a success and the page responded appropriately. Finally a valid move was POSTed to the server and the update page responded correctly by updating the move database and saving the new state of the game. This page was required top, and passed, the authentication test.

5.2.1.7 NEW ACCOUNT PAGE

Only a single test was required for the new account page. All the required information was sent to the management page, the server responded by creating a new account in the database and a success code was returned as well as redirecting the user to the profile page. A screenshot of the account creation page after a new account was successfully created is supplied in Figure 40.

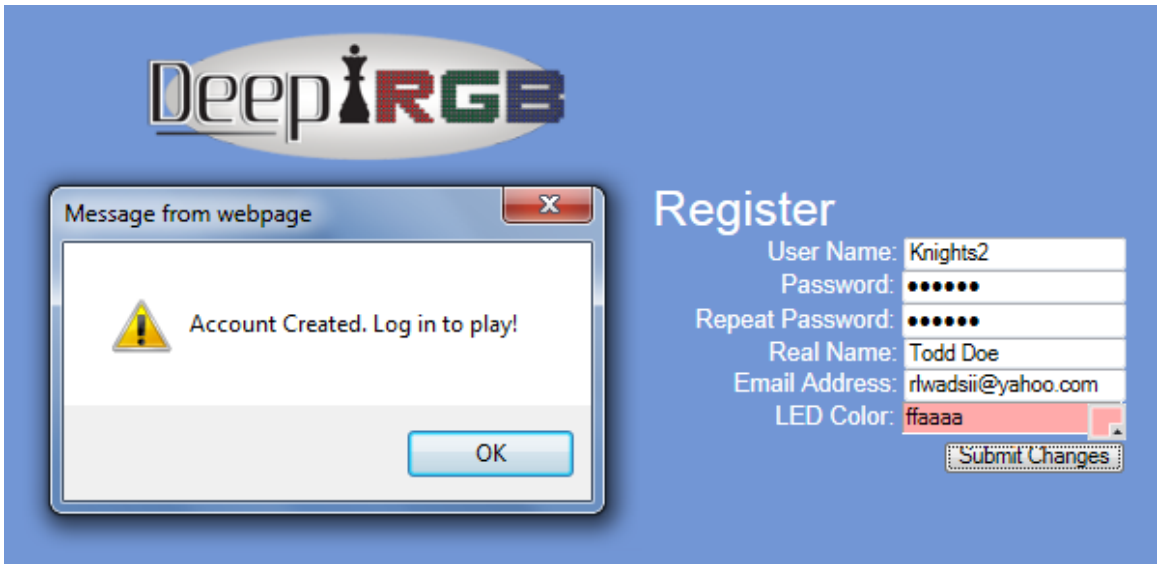


Figure 40 – The registration page working successfully and creating a new account.

5.2.1.8 PASSWORD LOST & PASSWORD RESET PAGES

A set of combined tests had to be performed for the password recovery system pages. The first test was to request a recovery key and use it to retrieve a lost password within thirty minutes. This test worked allowing for a new password to be selected and used to enter the account. Next as recovery key was requested and used after thirty minutes had passed. This resulted in a 403 error which meant the test was a success. Afterward two tests were given where a user tried to recover their password with a key requested by another user and to use a key that didn't exist. In both cases a 403 error was returned and sent the user back to the home page which indicated that the page functioned correctly.

As stated before in the design section of this paper the password recovery system consisted of several different sections. To begin the user would click the link on the home page to redirect to the lost password page where they would enter their user id, an example of the invalid id response was provided in Figure 41. After an id was entered correctly the server sent an email to the address associated with the id, a picture of this general email was provided in Figure 42. Finally to create a new password the user would click the link provided in the email and enter a new password on the password reset page. This link was only valid for 30 minutes before it went invalid and could only be used once a picture of the password reset page rejecting a authentication code that was no longer valid was provided in Figure 43.

5.2.1.9 ADD OBSERVER PAGE

The singular test for the add observer page was to attempt to add the user to an available game as the observer, then check the games list page for that user to ensure

that game appeared properly. This test was a success and the user was able to observe the game whilst the game was removed from the available games to observe list.

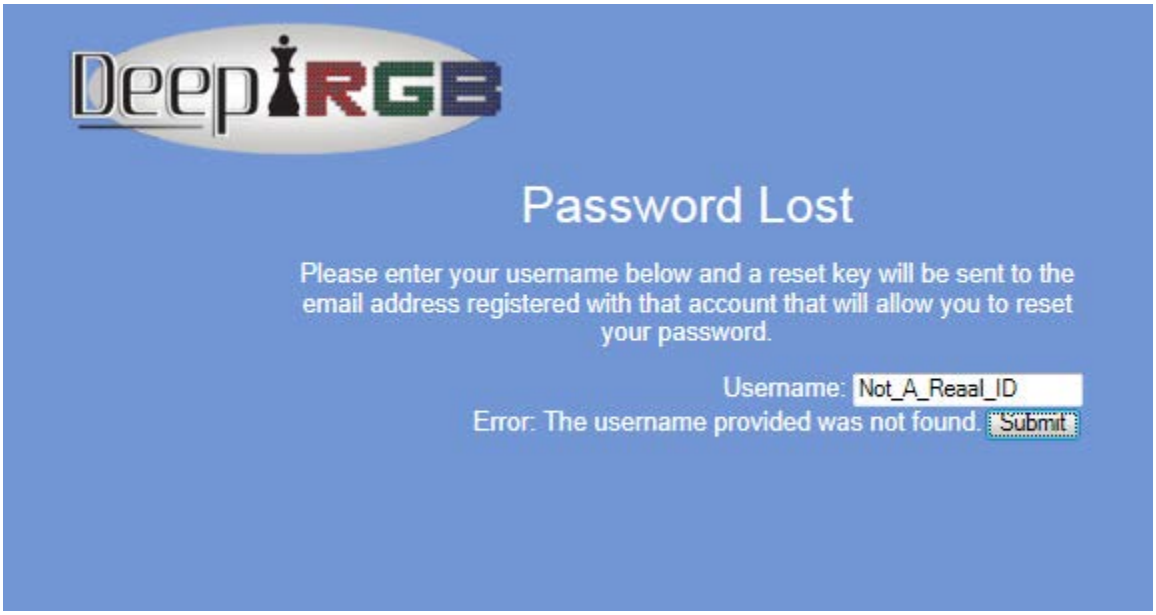


Figure 41 – The lost password page successfully rejecting an invalid username.

Deep RGB Password Reset Request
Deep RGB Server [deep.rgb.ucf@gmail.com]
Sent: Tuesday, December 11, 2012 12:41 PM
To: John Doe [rlwadsii@knights.ucf.edu]

John Doe,

You (or someone with your username) has requested a password reset from the Deep RGB System.

In order to reset your password, please click on the following link and you will be taken to a page that will allow you to reset your password. The following link is only good for twenty-four hours from the sending of this email. If resetting your password requires more time than this, please request another password reset key.

http://localhost:8080/web/resetpass.html?user=7&key=903C7D8FF6CCF3BC4357393E4DD68E06841EB595

If you have received this email without requesting a password reset, do nothing. In order to change your password, the above key is required.

Thank you for using Deep RGB,

-The Deep RGB Team

Figure 42 – The general email sent out to users when requesting a password recovery key including a link to the password reset page.

5.2.2 CHESS ENGINE PROCESS

The test for the chess engine process involved simply sending a board state to the chess engine thread and asking it for a response. After doing this a move was returned which

indicated that the chess engine was communicating with the thread correctly, to be sure the engine was working a second different state was uploaded which resulted in a different move being returned which indicated that the engine was functioning.

Figure 43 – The password reset page successfully rejecting an invalid password recovery key.

5.2.3 WEB CLIENT

Testing the web client was an integration test of the entire server system. Several games were played against the computer using the web interface and during these games illegal moves were made often. The server rejected the illegal moves but allowed the legal ones. Every possible combination of illegal moves for each specific piece was tried and the program wouldn't allow any proving that our engine and interface worked correctly.

The web client was updated from the state shown in the design section, by the final revision the interface had pieces and would in fact not allow a piece to be misplaced. Rather than allow a user to move a piece anywhere on the board when a piece was selected it flashed the tiles that piece could move to blue. A picture of the interface with the updated pieces whilst in the middle of a successful game instance was provided in Figure 44. If it was the opponent's chance to move the player could select the opposite color pieces, due to the way the interface was programmed it was easier to allow the user to select pieces from the side whose turn it currently was and simply respond with an error if it was not that players turn rather than not let the player select the opponent's pieces. This also served as an additional indicator as to whose turn it was in the game, an example of such an event and the response by the server was provided in Figure 45.

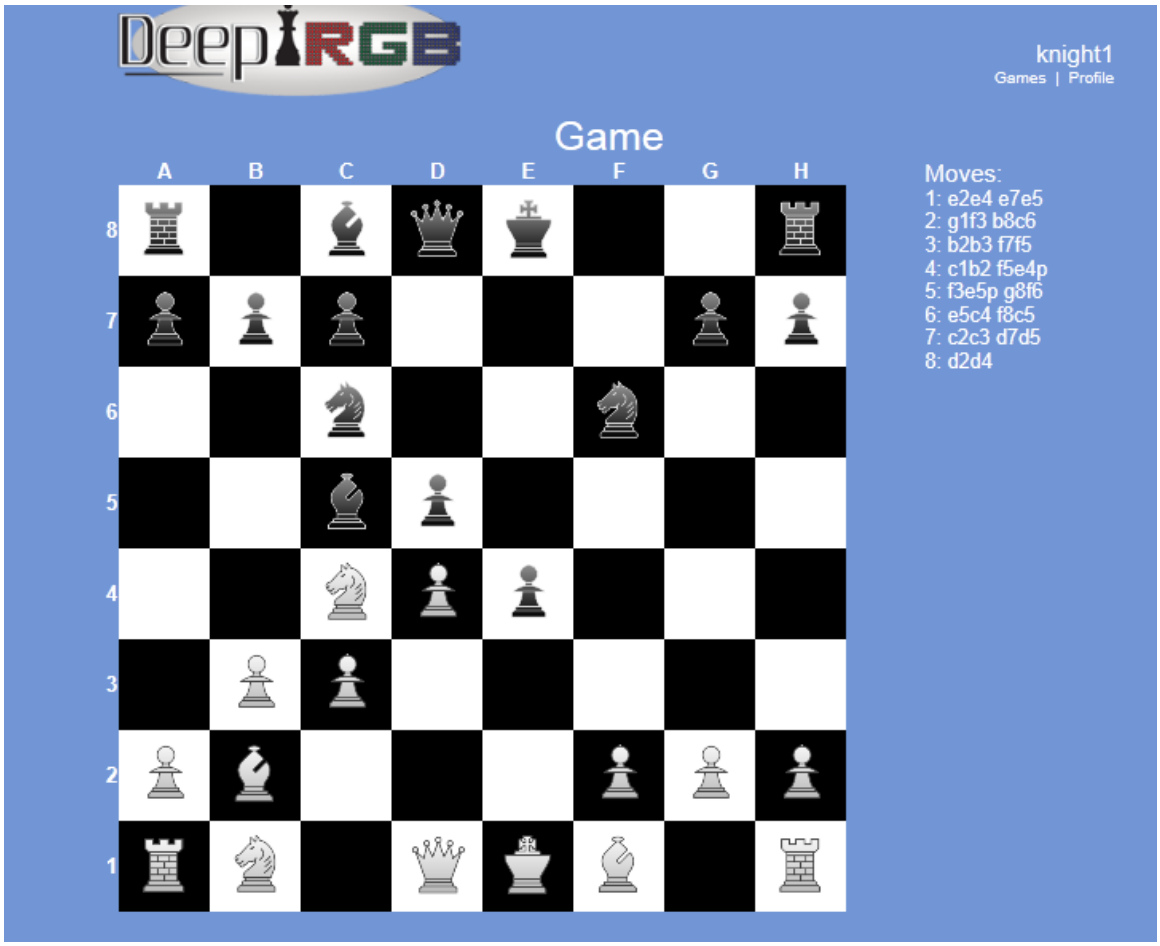
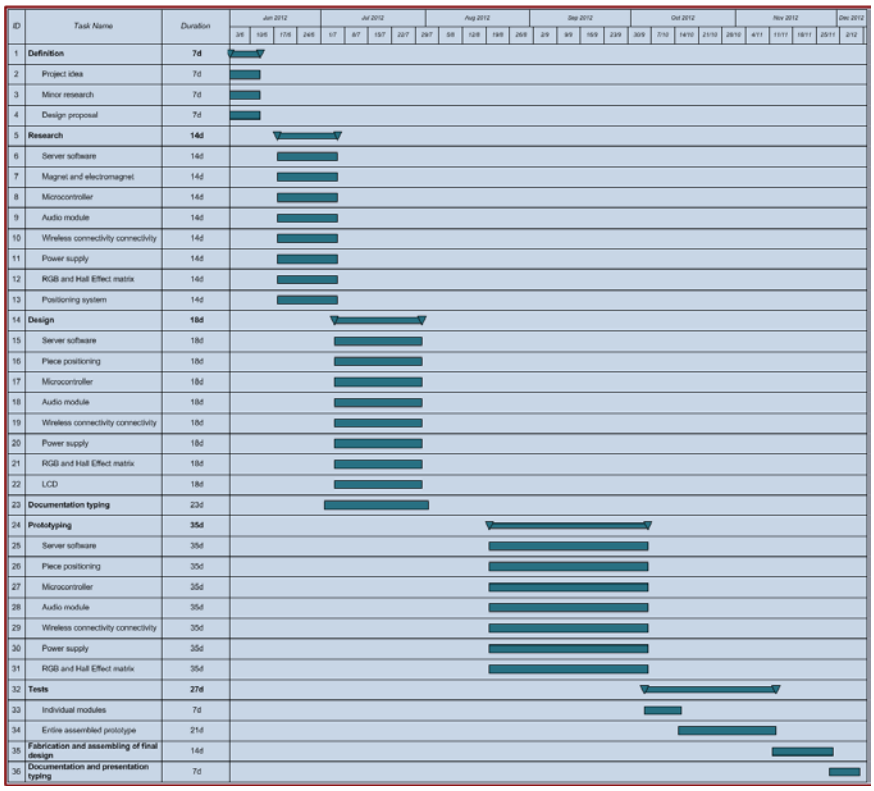


Figure 44 – The web interface in the midst of a game.

Figure 45 – The web interface successfully rejecting an invalid piece selection.

6 ADMINISTRATION

6.1 MILESTONES



ID	Task Name	Duration
1	**Definition**	**7d**
2	Project idea	7d
3	Minor research	7d
4	Design proposal	7d
5	**Research**	**14d**
6	Server software	14d
7	Magnet and electromagnet	14d
8	Microcontroller	14d
9	Audio module	14d
10	Wireless connectivity connectivity	14d
11	Power supply	14d
12	RGB and Hall Effect matrix	14d
13	Positioning system	14d
14	**Design**	**18d**
15	Server software	18d
16	Piece positioning	18d
17	Microcontroller	18d
18	Audio module	18d
19	Wireless connectivity connectivity	18d
20	Power supply	18d
21	RGB and Hall Effect matrix	18d
22	LCD	18d
23	**Documentation typing**	**23d**
24	**Prototyping**	**35d**
25	Server software	35d
26	Piece positioning	35d
27	Microcontroller	35d
28	Audio module	35d
29	Wireless connectivity connectivity	35d
30	Power supply	35d
31	RGB and Hall Effect matrix	35d
32	**Tests**	**27d**
33	Individual modules	7d
34	Entire assembled prototype	21d
35	**Fabrication and assembling of final design**	**14d**
36	**Documentation and presentation typing**	7d

Figure 46 - Gantt Chart

ID	Task Name	Start	Finish	Duration
1	**Definition**	**6/4/2012**	**6/12/2012**	**7d**
2	Project idea	6/4/2012	6/12/2012	7d
3	Minor research	6/4/2012	6/12/2012	7d
4	Design proposal	6/4/2012	6/12/2012	7d
5	**Research**	**6/18/2012**	**7/5/2012**	**14d**
6	Server software	6/18/2012	7/5/2012	14d
7	Magnet and electromagnet	6/18/2012	7/5/2012	14d
8	Microcontroller	6/18/2012	7/5/2012	14d
9	Audio module	6/18/2012	7/5/2012	14d
10	Wireless connectivity connectivity	6/18/2012	7/5/2012	14d
11	Power supply	6/18/2012	7/5/2012	14d
12	RGB and Hall Effect matrix	6/18/2012	7/5/2012	14d
13	Positioning system	6/18/2012	7/5/2012	14d
14	**Design**	**7/5/2012**	7/30/2012	**18d**
15	Server software	7/5/2012	7/30/2012	18d
16	Piece positioning	7/5/2012	7/30/2012	18d
17	Microcontroller	7/5/2012	7/30/2012	18d
18	Audio module	7/5/2012	7/30/2012	18d
19	Wireless connectivity connectivity	7/5/2012	7/30/2012	18d
20	Power supply	7/5/2012	7/30/2012	18d
21	RGB and Hall Effect matrix	7/5/2012	7/30/2012	18d
22	LCD	7/5/2012	7/30/2012	18d
23	**Documentation typing**	7/2/2012	8/1/2012	23d
24	**Prototyping**	8/20/2012	**10/5/2012**	**35d**
25	Server software	8/20/2012	10/5/2012	35d
26	Piece positioning	8/20/2012	10/5/2012	35d
27	Microcontroller	8/20/2012	10/5/2012	35d
28	Audio module	8/20/2012	10/5/2012	35d
29	Wireless connectivity connectivity	8/20/2012	10/5/2012	35d
30	Power supply	8/20/2012	10/5/2012	35d
31	RGB and Hall Effect matrix	8/20/2012	10/5/2012	35d
32	**Tests**	**10/5/2012**	**11/12/2012**	**27d**
33	Individual modules	10/5/2012	10/15/2012	7d
34	Entire assembled prototype	10/15/2012	11/12/2012	21d
35	**Fabrication and assembling of final design**	11/12/2012	11/29/2012	14d
36	**Documentation and presentation typing**	11/29/2012	12/7/2012	7d

Figure 47 - Gantt Details

6.2 BILL OF MATERIALS

Table 32 – Bill of Materials for development stage.

Item	Distributor	Part Num.	Quantity	Total Price
Arduino Atmega 2560*	Sparkfun	DEV-11061	1	$58.95
WiFly GSX	Sparkfun	WRL-10050	1	$84.95
Neodymium magnets	Amazon	B001ANVAHI	1 (pack of 100)	$4.59
16x2 LCD display	Donated by team member	LM1602C	1	$0.00
Stepper motors	Sparkfun	ROB-09238	2	$29.90
Rack Gearbox Bracket (2-pack)	Vexrobotics	275-1188	1	$9.99
Advanced Gear Kit	Vexrobotics	276-2184	1	$19.99
Amico Electromagnet	Amazon	B005F79TIW	1	$6.82
Sparkle power supply	Newegg	N82E16817103064	1	$37.99
MP3 Player shield	Sparkfun	DEV-10628	1	$39.95
Stepper motor driver	Pololu	1183	2	$39.90
Analog multiplexer	Analog	ADG406BNZ	4	$6.60
Allegro Hall Effect Sensor	Digikey	A1360LKTTN-T	117	$281.43
Shift registers	Digikey	296-14857-1-ND	10	$4.94
RGB LEDs	Sparkfun	YSL-R596CR3G4B5W-F12	1 (pack of 100)	$59.95
Custom PC functioning as server	Donated by team member	None	1	$0.00
Server OS	Donated by team member	None	1	$0.00
Visual Studio 10	Donated by team member	None	1	$0.00
SQL Server Browser	Donated by team member	None	1	$0.00
Chrome Developer Toold	Google Inc.	None	1	$0.00
Total price without tax or shipping				$692.36

Table 33 - Bill of materials for final design stage.

Item	Distributor	Part Num.	Quantity	Total Price
PCB	4PCB	None	1	$33.00
ATmega 2560*	Digikey	ATMEGA2560V-8AU	1	$19.97
ATmega16-U2*	Digikey	ATMEGA16U2-AU-ND	1	$3.71
Various main board components(USB socket, headers, etc)*	Digikey Sparkfun Ebay	None	1	$33.46
Rover RN131C	Digikey	WRL-10050	1	$84.95
Stepper motor driver IC	Digikey	620-1140-2-ND	2	$27.90
Motors	Sparkfun	ROB-09238	2	37.68
Audio Shield**	Sparkfun	DEV-10628	1	$19.95
Racks and Gears	SDP-SI	none	N/A	$47.77
Neodymium Magnets	ebay	None	N/A	$65.25
Solenoid	Skycraft	None	1	$5.00
Hall Effect Sensors	Newark	89T7955	100	$135.00
RGB LEDs	Superbright LEDs	RL5-RGB-C-2	70	$49.70
Construction Materials	None	None	None	$160.82
Custom PC functioning as server	Donated by team member	None	1	$0.00
Server OS	Donated by team member	None	1	$0.00
Visual Studio 10	Donated by team member	None	1	$0.00
SQL Server Browser	Donated by team member	None	1	$0.00
Chrome Developer Toold	Google Inc.	None	1	$0.00
Total price without tax or shipping				$745.37

7 APPENDICES

7.1 BIBLIOGRAPHY AND CITATIONS

7.1.1 WORKS CITED

Arduino. (n.d.). *Arduino Mega 2560*. Retrieved 7 15, 2012, from Arduino: http://arduino.cc/en/Main/ArduinoBoardMega2560

Arduino. (n.d.). *Arduino-mega2560_R3-schematic.* Retrieved 7 25, 2012, from Arduino: http://arduino.cc/en/uploads/Main/arduino-mega2560_R3-schematic.pdf

Kgrr. (n.d.). *Wikipedia.* Retrieved 7 2, 2012, from Wireless mesh network: http://en.wikipedia.org/wiki/Wireless_mesh_network

Microchip. (n.d.). *PIC18 Explorer Board*. Retrieved 7 21, 2012, from Microchip: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en535770

Microchip. (n.d.). *PIC18F46K80* . Retrieved 7 19, 2012, from Microchip: http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en550197

Networks, R. (n.d.). *WiFly GSX/EZX.* Retrieved 6 29, 2012, from Roving Networks: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/WiFi/WiFly-RN-UM.pdf

RSparkfun. (n.d.). *WiFly GSX Breakout Board*. Retrieved 8 3, 2012, from Sparkfun: https://www.sparkfun.com/products/10050

Sparkfun. (n.d.). *Arduino Mega 2560 R3*. Retrieved 7 21, 2012, from https://www.sparkfun.com/products/11061: https://www.sparkfun.com/products/11061

Sparkfun. (n.d.). *Bluetooth Modem - BlueSMiRF Silver.* Retrieved 7 6, 2012, from Sparkfun: https://www.sparkfun.com/products/10269?

Sparkfun. (n.d.). *RN-131G Breakout-v13.* Retrieved 7 5, 2012, from Sparkfun : http://www.sparkfun.com/datasheets/Wireless/WiFi/RN-131G%20Breakout-v13.pdf

Sparkfun. (n.d.). *XBee 1mW Chip Antenna - Series 1 (802.15.4).* Retrieved 7 3, 2012, from Sparkfun: https://www.sparkfun.com/products/8664?

TI. (n.d.). *MSP430FR5739*. Retrieved 7 16, 2012, from TI: http://www.ti.com/product/msp430fr5739#description

TI. (n.d.). *MSP-EXP430FR5739 Experimenter Board*. Retrieved 7 21, 2012, from TI: http://www.ti.com/tool/msp-exp430fr5739

Wikepedia. (n.d.). *Printed circuit board.* Retrieved 7 22, 2012, from Wikepedia: http://en.wikipedia.org/wiki/Printed_circuit_board

C-2. (n.d.). *RL5-RGB-C-2*. Retrieved 7 13, 2012, from Super Brigh LEDs: http://www.superbrightleds.com/moreinfo/component-leds/5mm-rgb-clear-tricolor-led-wide-angle/976/

RGB-C. (n.d.). *RL5-RGB-C*. Retrieved 7 13, 2012, from Super Bright LEDs: http://www.superbrightleds.com/moreinfo/component-leds/rl5-rgb-clear-tricolor-led/298/

C10. (n.d.). *YSL-R596CR3G4B5C-C10*. Retrieved 7 13, 2012, from Sparkfun: https://www.sparkfun.com/products/105

F-12. (n.d.). *YSL-R596CR3G4B5W-F12*. Retrieved 7 13, 2012, from Sparkfun: http://www.sparkfun.com/datasheets/Components/LED/YSL-R596CR4G3B5W-F12.pdf

74HC595. (n.d.). *74HC595*. Retrieved 7 13, 2012, from Sparkfun: https://www.sparkfun.com/products/733

1302. (n.d.). *Allegro A1302UA*. Retrieved 7 20, 2012, from Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?wt.z_cid=ref_hearst_0211_buynow&mpart=A1302KLHLT-T&v=620&cur=USD

90215. (n.d.). *Melexis MLX90215*. Retrieved 7 20, 2012, from Melexis: http://www.melexis.com/ProdMain.aspx?nID=12

1384. (n.d.). *Allegro A1384*. Retrieved 7 20, 2012, from Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?wt.z_cid=ref_hearst_0211_buynow&mpart=A1384LLHLT-T&v=620&cur=USD

1360. (n.d.). *Allegro A1360*. Retrieved 7 20, 2012, from Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?wt.z_cid=ref_hearst_0211_buynow&mpart=A1384LLHLT-T&v=620&cur=USD

4067. (n.d.). *Texas Instruments CD74HC4067*. Retrieved 7 21, 2012, from Texas Instruments: http://www.ti.com/lit/ds/symlink/cd74hc4067.pdf

406. (n.d.). *Analog Devices ADG406*. Retrieved 7 21, 2012, from Analog Devices: http://www.analog.com/en/switchesmultiplexers/multiplexers-muxes/adg406/products/product.html

426. (n.d.). *Analog Devices ADG426*. Retrieved 7 21, 2012, from Analog Devices: http://www.analog.com/en/switchesmultiplexers/multiplexers-muxes/adg406/products/product.html

506. (n.d.). *Analog Devices ADG506A*. Retrieved 7 21, 2012, from Analog Devices: http://www.analog.com/en/switchesmultiplexers/multiplexers-muxes/adg506a/products/product.html

019. (n.d.). *Seeed Studio Arduino-019 Audio shield* Retrieved 7 24, 2012, from Micro4you: http://www.micro4you.com/store/arduino/arduino-shield/arduino-mp3-shield.html

rMP3. (n.d.). *Rogue Robotics rMP3*. Retrieved 7 24, 2012, from Cutedigi: http://www.cutedigi.com/arduino-shields/mp3-shield-for-arduino.html

10628. (n.d.). *DEV-10628*. Retrieved 7 24, 2012, from Sparkfun: https://www.sparkfun.com/products/10628

7.1.2 REFERENCES

[1]Wabbot,"Stepper Motors (2)". [Online]. Available:

http://www.societyofrobots.com/member_tutorials/node/120

[2]"Stepper Motors". [Online]. Available:

http://www.tigoe.com/pcomp/code/circuits/motors/stepper-motors/

[3] Jones W. Douglas, "Control of Stepper Motors a Tutorial".[Online]. Available:

http://homepage.cs.uiowa.edu/~jones/step/index.html

[4]"Motor Control and Drive". [Online]. Available:http://www.microchip.com/pagehandler/en-us/technology/motorcontrol/motortypes/brushed.html

[5]"Stepper Motors". [Online]. Available: http://www.epanorama.net/links/motorcontrol.html#stepper

[6]"Electromagnet". [Online]. Available: http://www.geeplus.biz/FTPROOT/Electromagnets_technical_overview.pdf

[7]Stefan Holodnick, “Playing with my Arduino (Board)”. [Online]. Available: http://blog.dailyinvention.com/playing-with-my-arduino-board/

[8]”LCD Display”. [Online]. Available:

http://www.electronics-project-design.com/LCD_Display.html

[9]”Schematics –Robot Power Regulation”. [Online]. Available: http://www.societyofrobots.com/schematics_powerregulation.shtml

[10]”Pedal Powered Prime Mover – DIY Plans”. [Online]. Available:

http://www.los-gatos.ca.us/davidbu/pedgen/plans.html

7.1.3 LICENSES

Below are the licenses of the libraries and software used to program the code for Deep RGB.

7.1.3.1 ARDUINO IDE

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program

whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

> **a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
>
> **b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
>
> **c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the

terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> **b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> **c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place

counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to

the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE

PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7.1.3.2 JSON LIBRARY

Copyright (c) 2002 JSON.org

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

The Software shall be used for Good, not Evil.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.1.3.3 MTHREAD LIBRARY

GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates

the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified

version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the

copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

7.1.3.4 ACCELSTEPPER LIBRARY

The Accelstepper library uses version 3 of the GNU General Public License which is also used by the MThread library and was listed above, therefore it was not relisted.

7.1.3.5 C# WEBSERVER LIBRARY

Apache License 2.0 (Apache)

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and

2. You must cause any modified files to carry prominent notices stating that You changed the files; and

3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the

Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this

License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.