Chess Playing Robot

Sean Vibbert, Electrical Engineering

Project Advisor: Dr. Mohsen Lotfalian

April 26, 2019

Evansville, Indiana

Acknowledgments

I would like to thank Jeff Cron and Braden Elliott for their help with 3D printing and Ray Shelton for creating custom mechanical parts. I would also like to thank Dr. Mohsen Lotfalian and Dr. Anthony Richardson for advice on device implementation and Dr. Marc Mitchell for guidance on using microcontrollers to control motors.

Table of Contents

List of Figures

List of Tables

I. Introduction

In sports, training robots have given athletes an edge when preparing for competitions. If combined with problem solving algorithms, athletes can root out weaknesses in their technique very quickly. In chess, however, problem solving algorithms, known as chess engines, exist solely in software. For a chess player to train, he or she must play on a virtual board, unlike real competitions. To resolve this issue, the Chess Playing Robot creates an artificial opponent that can make decisions, moves pieces, and hit the clock independently. This robot could be invaluable for players who want to train in a realistic environment.

II. Objectives

The primary focus of the Chess Playing Robot was the ability to play chess independently. As such, the project goals include all the basic functions of chess: moving and taking pieces, hitting the clock, and making decisions. Also, a way to control the robot's various functions was required. These features also needed to be reliable. Movement needed to be consistent, firmly grasping the chess pieces and relocating them without knocking over other pieces. The same consistently is expected for hitting the clock. Regarding decision making, the program needed to select moves on its own, requiring knowledge of the game state after each human move. Not all necessary functions of the machine were explicitly stated, but basic features such as receiving power from a wall outlet are expected. Table 1 is a summary of both the stated and unstated expectations.

Table 1 – Project Requirements

Electromechanical Requirements	Logical/Algorithmic Requirements
■ The robot must be able to move pieces reliably, avoiding knocking over other pieces and repositioning within the desired squares ■ The robot must be able to hit the clock after it makes a move ■ The motors should remain in sync with each other, needing minimal tuning from the user ■ The entire project should be powered from a wall outlet. If a separate device is used for the chess engine, it can have an alternative power source	■ The robot must keep track of the board state, whether it be from a user entering individual moves or from visual recognition ■ The robot must be capable of choosing moves on its own, whether it be at random or by a chess engine ■ Pieces taken from the board must be placed in a "capture zone" next to the clock ■ The robot must be able to receive commands from the user, such as calibration of the claw's position

III. Project Design

To accomplish the goal of moving chess pieces, inspiration was taken from both claw machines and 3D printers alike. Dubbed the "X-Y-Z approach," each dimension over the board is controlled with its own motor. The original concept design, seen in Figure 1, made use of 6 beams and 3 platforms. Platforms which move across the Y-axis move a single center platform which moves across the X-axis. This center platform has a claw which can be extended on the Z-axis, giving full control over the entire chess board. Before investing in this design, a small-scale prototype was constructed which was successful at moving chess pieces.

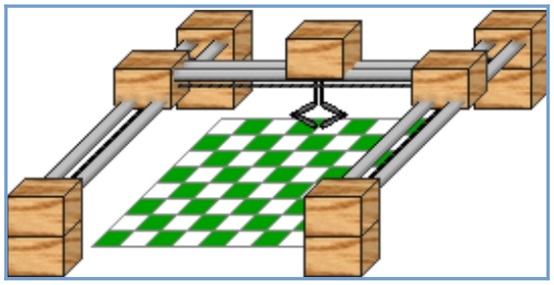


Figure 1 – Original Concept Art

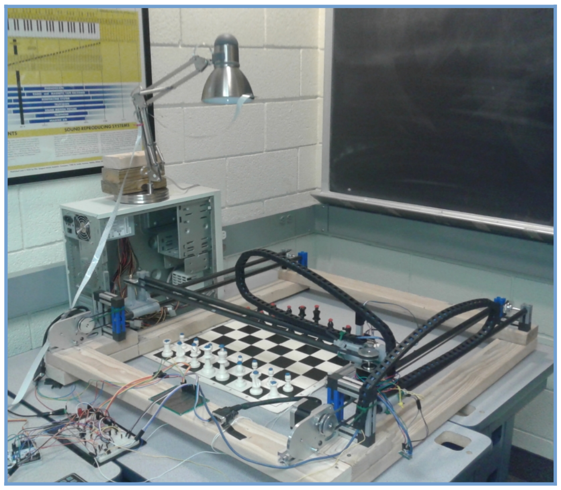


Figure 2 – Completed Project

After this test, full-scale implementation began. The physical result of this design can be seen in Figure 2. The design of the Chess Playing Robot is split into three sections: Mechanical Components, Electronics and Circuitry, and Logic and Algorithms.

A) Mechanical Components

The mechanical composition of the Chess Playing Robot is made from the following components: 8020 connectors, precision belts and pulleys, metal beams, custom 3D printed parts, and a wooden base stand. 8020 connectors are known for their ability to be completely disassembled or remain rigid, making them ideal for semi-permanent designing. In order to hold the metal beams in place, custom 8020 components were designed and 3D printed. Figure 3 details this custom design and Figure 4 shows them holding the metal beams in place. Also, the 8020 connectors are convenient for holding pulleys in place. A smooth metal pole is place into two pins to accomplish this, seen again in Figure 4.

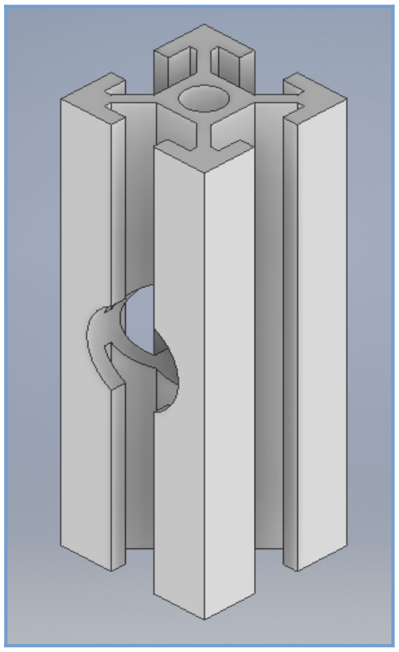


Figure 3 – Custom 8020 Connector (Single Pole)

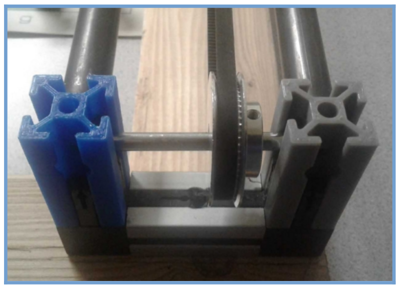


Figure 4 – 8020 Beam and Pulley System

For movement across these beams, custom 3D printed platforms were created. There are two Y-axis platforms. This platform, shown in Figure 5, is designed to carry a significant amount of weight. Each Y-axis platform has its own top piece, shown in Figure 6 and Figure 7. One end carries a stepper motor, and the other end holds a pulley. Also attached to each platform is another 3D printed 8020 connector, seen in Figure 8. These connectors hold two beams instead of one.

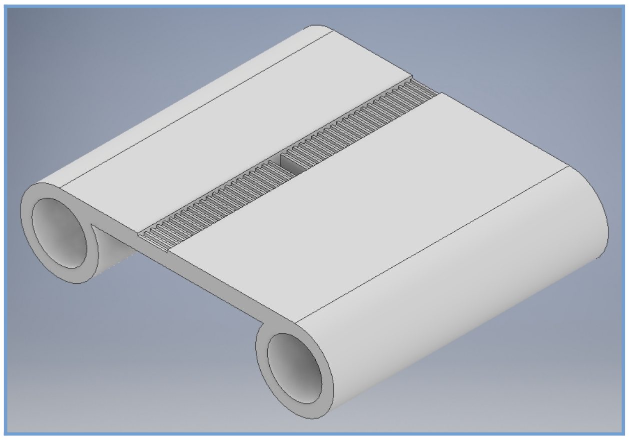


Figure 5 – X-axis Sliding Platform

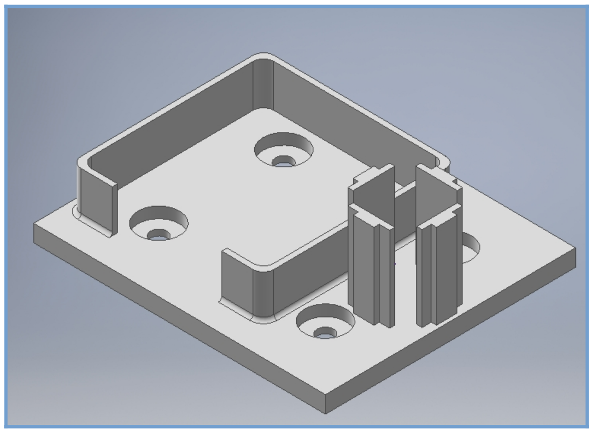


Figure 6 – Sliding Top, Motor

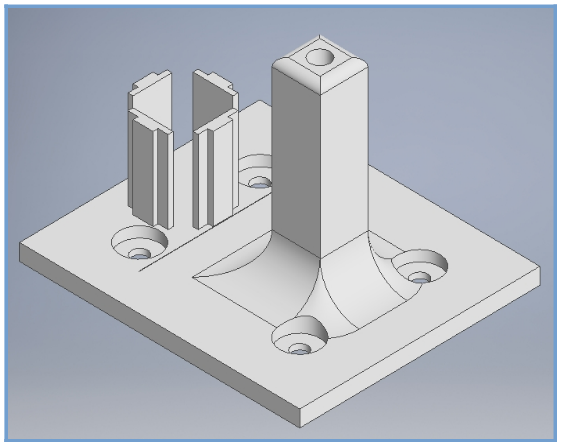


Figure 7 – Sliding Top, Pulley Tower

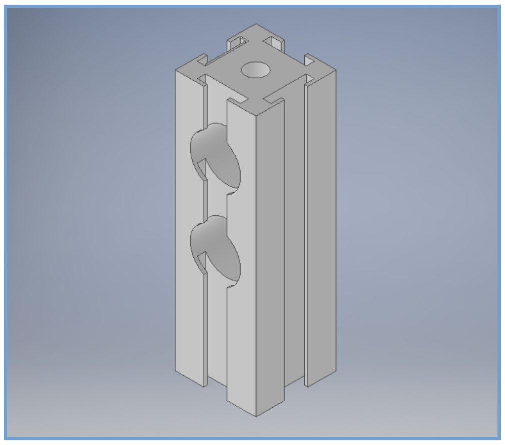


Figure 8 – Custom 8020 Connector (Double Pole)

The X-axis platform, which moves over the board, is also 3D printed and functions similarly to the Y-axis platforms. The square hole on the side holds the linear actuator, and the opening on the other side holds the belt in place. Figure 9 details this platform.

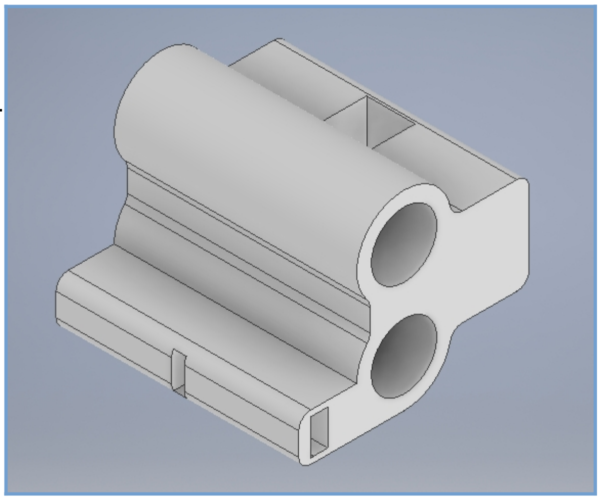


Figure 9 – Y-axis Sliding Platform

In order to pick up pieces, an electromagnet is used instead of a claw. Standard chess pieces are not magnetic, so ferrous metal nails and washers were drilled into the top of all 32 pieces. Figure 10 shows every custom piece, and Figure 11 shows the movement in action.

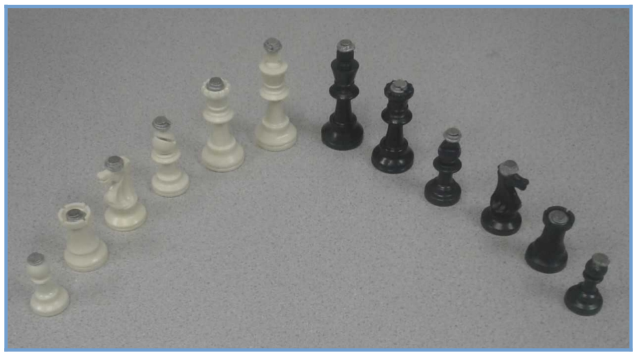


Figure 10 – Modified Chess Pieces

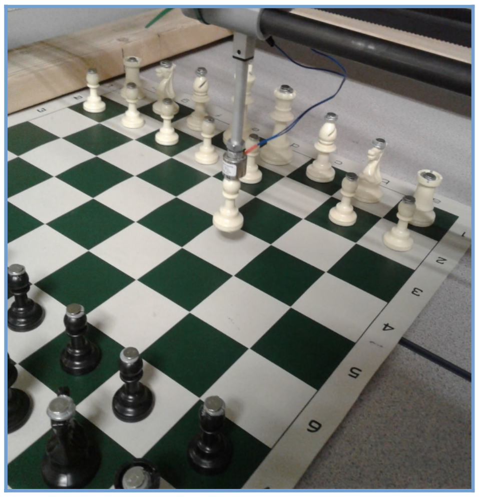


Figure 11 – Piece in Motion

B) Electronics and Circuitry

The chess playing robot uses three electrical components that move physical objects: stepper motors, a linear actuator, and an electromagnet *(See section 5"Items and Costs.")* All of these components operate from a 12 volt source. If all components are on simultaneously, the system draws about 2 amps, or 24 watts. Each stepper motor is controlled using a four channel stepper motor driver and are locked to half-stepping. If the stepper motors are full-stepped, they are not robust enough to move the platforms consistently. The linear actuator has an on-device controller and receives a 0-5 volt DC signal, where 0 volts is fully retracted and 5 volts is fully extended. Lastly, the electromagnet is controlled using an H-bridge, which operates as a digitally controlled on/off switch.

The ARM M4 Microcontroller, seen in Figure 12, is used to control all these aforementioned components. The GPIO pins have a logical high of 3.3 volts, which is also the logical high of the stepper motor drivers. Each pulse sent to drivers will cause a forward or backwards step, depending on the state of the direction pin. These steps are consistent, meaning that no feedback control system is required for precision. The linear actuator 0-5 volt input is controlled with an open drain PWM pin connected to 5 volts through a pull-up resistor. The H-bridge has a logical input level of 5 volts, so its enable bit is also pulled up to 5 volts. Input commands, such as coordinate for a chess move, calibration of the claw's starting position, and a start sequence for movement can be sent to the ARM board using a 16 button keypad. This keypad should only be used for debug purposes, and the clock button should be pressed instead to tell the robot it is time to make a move.

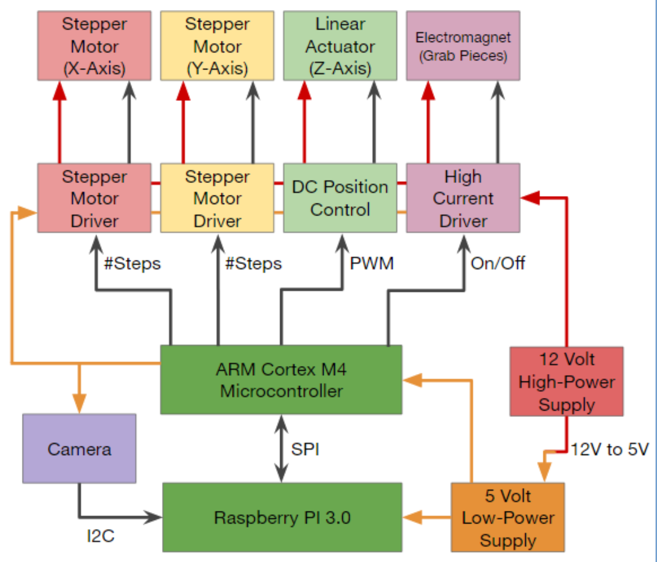


Figure 12 - High Level Circuit Design

The only other electronic device is the Raspberry Pi camera, which is controlled by the Raspberry PI 3.0. Communication is made from the Raspberry PI to the ARM board and vice versa via a custom communication protocol which is similar to SPI. The only main difference is that bits are clocked in on both rising and falling edges. Figure 13 is the complete circuit design.

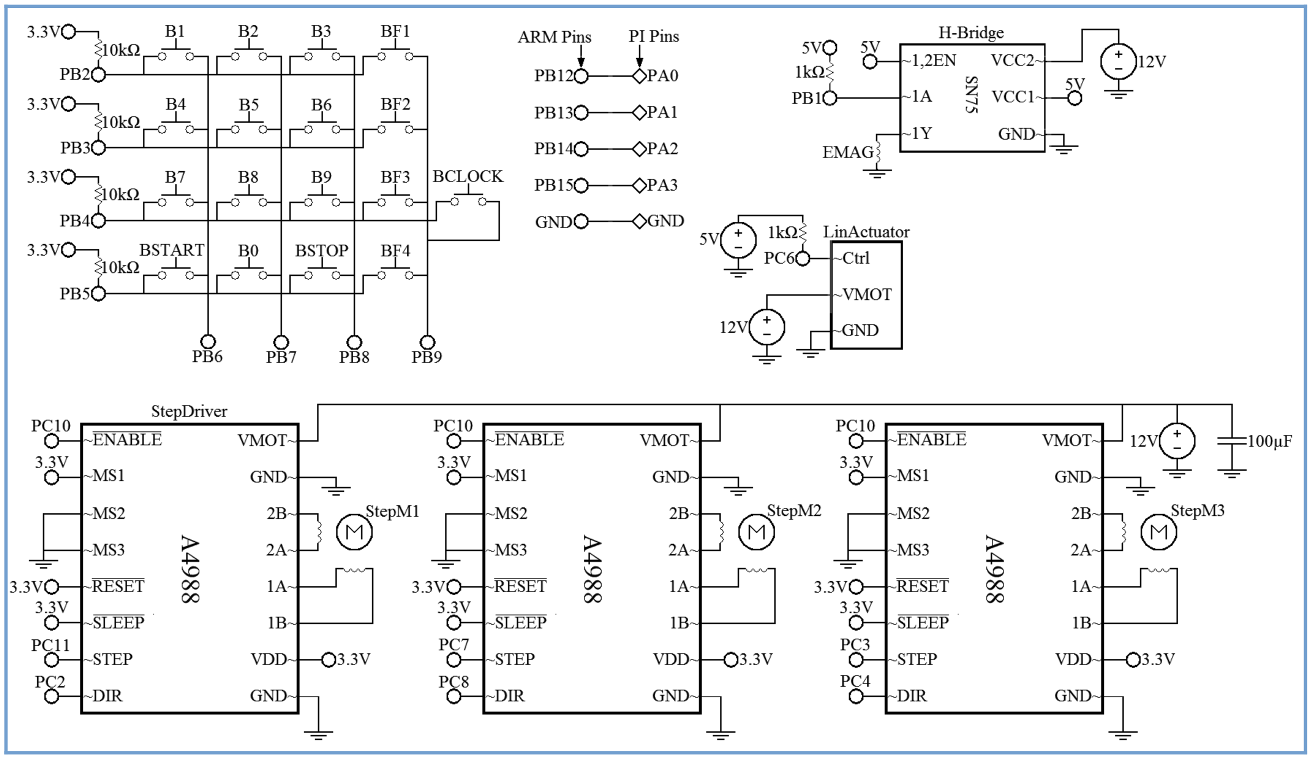
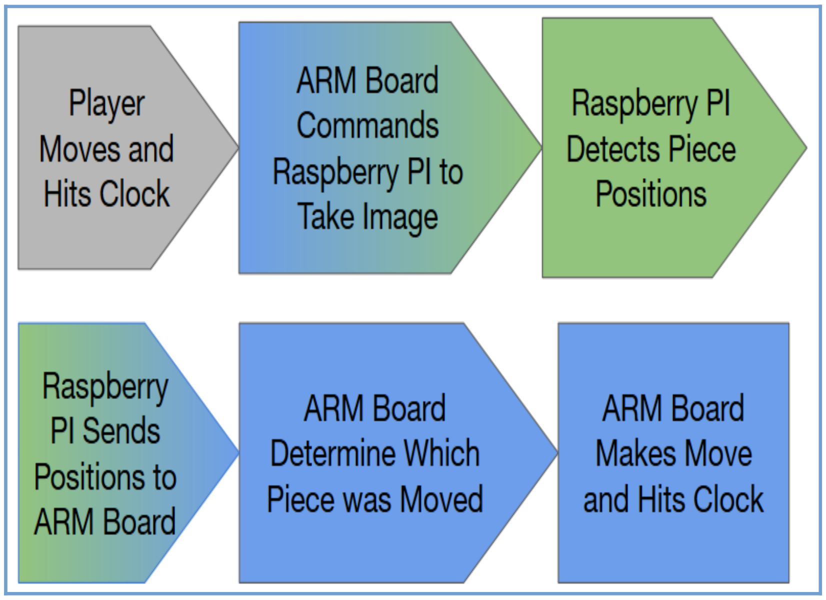


Figure 13 – Complete Circuit Diagram

C) Logic and Algorithms


Player Moves and Hits Clock
ARM Board Commands Raspberry PI to Take Image
Raspberry PI Detects Piece Positions
Raspberry PI Sends Positions to ARM Board
ARM Board Determine Which Piece was Moved
ARM Board Makes Move and Hits Clock
Figure 14 – Piece Movement Routine

Because the Chess Playing Robot has no significant feedback control, movement routines rely heavily on the consistency of algorithms. The stepper motors, in theory, will always rotate the same number of degrees, and the linear actuator will always extend to the same distance. Because of these two assumptions, the position of the electromagnet is predictable, and therefore the movement of pieces is predictable. Furthermore, the game of chess always starts in the same position. If the microcontroller knows each move that is made, it will know exactly where all pieces are located. The movement of a piece requires seven inputs: The first row and column, the second row and column, the piece type, whether or not a piece is being taken, and the type of the taken piece. Whenever a piece is taken, it moves it to a "capture zone," which is simply an area where taken pieces are stored. Figure 14 is the routine which is followed each move.

The ARM board runs a version of chess which detects every basic legal move for the side it is playing. Castling, pawn promotion, and en passant are not considered. Assuming legal moves are made, it will also track the position of moved pieces, which means an engine of any strength could be implemented. For the purpose of demonstration, a rudimentary chess engine is programmed on the ARM board which chooses a random legal move. It will also prioritize captures. The engine needs a position to evaluate in order to find a move, and there are two ways to capture the game state. This first option is by inputting all human moves via the keypad, though this option is recommended only for debugging. The second option is to use the Raspberry PI camera and visual recognition. After the human makes a move and hits the clock, the camera scans the board for a change. Atop each white piece is a blue dot and atop each black piece is a red dot. The change in these dots' locations is enough to determine a move. Figure 15 helps to visualize this concept.

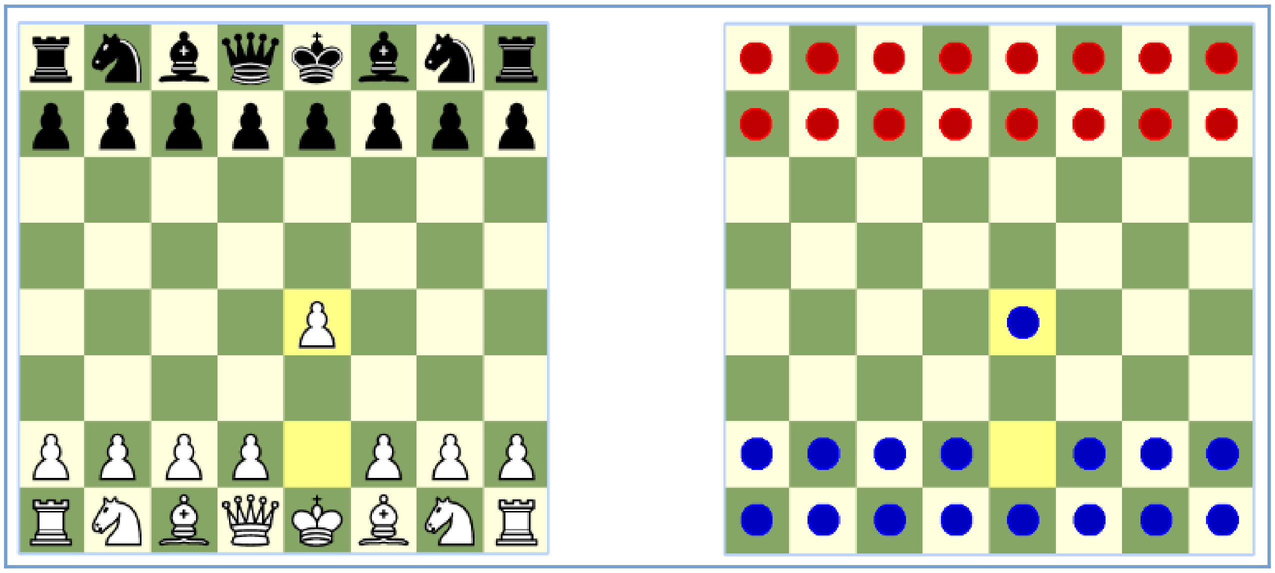


Figure 15 – Visual Recognition Illustration

For calibration and demonstration purposes, the Raspberry PI creates two ppm images which show blue and red content after filtering. The amount of blue and red in each grid determines whether or not a piece occupies a square. If the room is too dark, some areas will not filter properly. Figure 16 shows the original image alongside its filtered counterpart:

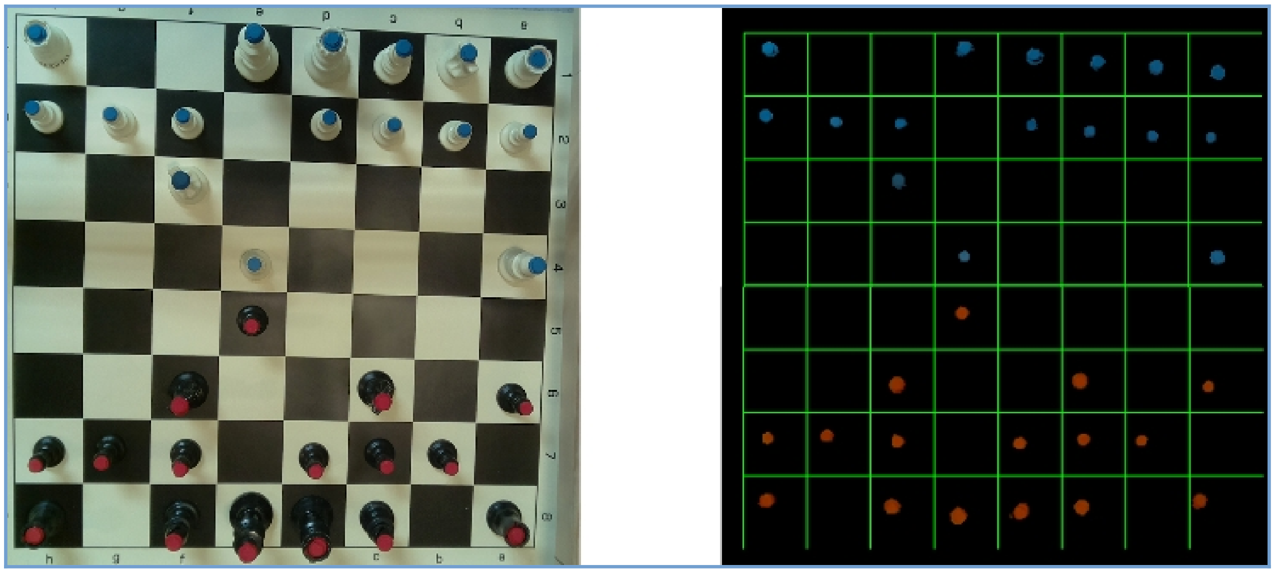


Figure 16 – Filtered Image

To take the image, the "raspicam" program is called and saves the overhead image as a JPEG. Then, "ImageMagick" is called to convert the JPEG to a magick file. Magick files are arrays of RGB values represented in hexadecimal. After this conversion, the magick file is placed into an array in C where filtering is applied manually.

IV. Performance and Results

The Chess Playing Robot worked exactly as intended. The stepper motors were extremely consistent, with no noticeable position error after hours of use. So long as the chess board was aligned properly, piece movement was also consistent. It takes the robot about a minute to make a non-capture move and about 90 seconds to make a capture move. The robot never knocks over other pieces in the movement process due transportation taking place in-between squares, and captured pieces were placed in the capture zone. After each move, the robot hits the clock. The built-in engine was able to make legal moves and the visual recognition consistently detected moves made by the human opponent. Video was recorded which shows the robot successfully completing a medium-length game of chess against a human.

V. Items and Costs

The costs for this project remained within the allotted budget of $498. Some of the money was spent on experimentation, while the rest goes into the cost of the final product. Table 2 and 3 detail each significant item used throughout work done on the Chess Playing Robot.

Table 2 – Final Project Cost

Item	Cost
Electromagnet, 50 Newton	$9.19
Raspberry PI Mini Camera	$10.57
Raspberry PI Camera Cable	$4.00
AC Power Adapter, 3 Amps	$10.59
Keypad Membrane, 16 Keys	$4.00
Timing Belt, 10 Meters	$9.99
2x Wire Carrier Drag Chain	$19.38
Nema Stepper Motor	$13.99
3x Stepper Motor Driver	$6.00
6x Timing Pulley (Large)	$21.60
AC Lamp + Stand	$18.18
Arm Board and Raspberry PI	$49.99
Chess Set	$15.00
6x Aluminum Rod	$12.00
2x Generic Stepper Motor	$30.00
8020 Aluminum Connectors	$5.00
3D Printed Material	$10.00
Actuonix Linear Actuator	$40.00
Total:	**$289.48**

Table 3 – Experimental Cost

Item	Cost
Electromagnet, 25 Newton	$7.99
Keypad Membrane, 16 Keys	$4.00
Jumper Wires (short)	$13.98
Jumper Wires (long)	$19.98
Raspberry PI Camera Cable	$4.00
2x Nema Stepper Motor	$27.98
2x Stepper Motor Driver	$4.00
8x Timing Pulley (small) + 5 Meter Timing Cable	$13.98
6x Timing Pulley (large)	$21.60
Timing Belt, 10 Meters	$9.99
Total:	**$127.50**

VI. Safety and Transportation

The chess playing robot uses both moving wires and moving mechanical parts. In some instances, unprotected wires can shock or even kill a user. To prevent such a disaster, the *IEEE Guide on Shielding Practice for Low Voltage Cables* was considered. All moving wires are protected by nested cable holders which prevent physical contact. Further, moving parts in the system such as the pulleys or platform could crush a user's hands. The project is designed to never allow such a scenario to occur. One end of the project is opened, and the other end is never reached due to the starting position of the moving platforms. The project is also designed to be portable via disassembly and reassembly. If desired, the metal beams and 8020 connectors can all be removed. It is highly recommended to loosen all belts before moving the project, as the pulleys are under tension and held in place due to gravity. The pulleys can become dangerous projectiles if held under tension at awkward angles.

VII. Conclusion

The Chess Playing Robot has successfully demonstrated that an artificial opponent can be created using a combination of moving parts and a chess engine. The project's electromechanical and algorithmic requirements were all met with the added features of visual recognition and convenient disassembly and reassembly. With the existence of an artificial opponent, the concept of automated chess can be expanded in the future into a more advanced training robot or even compatibility with a variety of different tabletop games.

VIII. Appendix A –Setup Instructions

To setup the robot after relocating it, the mechanical fixture must be level. It is also desirable for the table to be level. However, some error is acceptable because of the strength of the electromagnet. Small plastic stumps were placed under each corner of the robot during demonstration and will likely be necessary for different tables. If we define the "right" to be the side with the moving stepper motor and the "bottom" to be the side with the two stepper motors, the electromagnet claw should start in the bottom right. The chess set should be placed on the opposite side in the bottom left. To calibrate the chess set's location, simply move the set to where the electromagnet tries to grab a piece. The overhead camera should be aligned to the center of the board, and markers are placed in the corners to help with alignment. Figure 17 shows the position of the board as well as the markers.

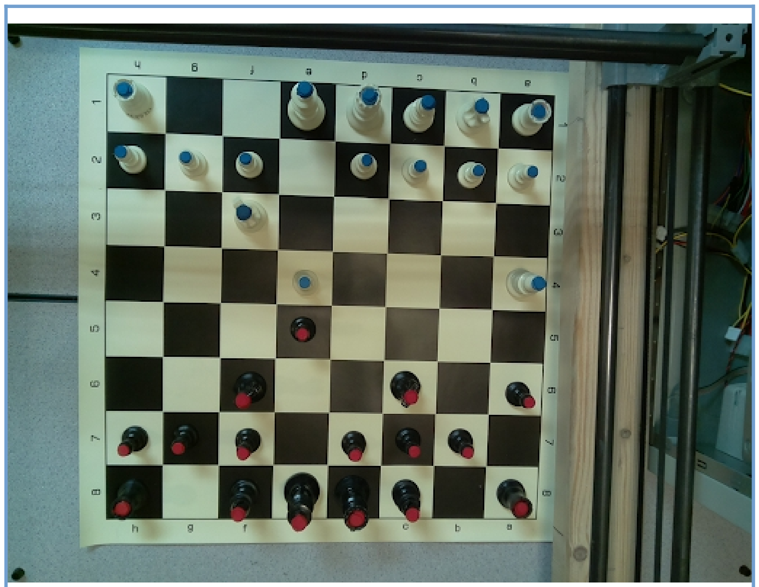


Figure 17 - Chess Set Alignment

The lightning in the room is important for visual recognition. A well lit room should suffice, but if the room is too dark, the filtering process will fail. A weak light bulb can be placed in the lamp to resolve this issue. To calibrate the camera, run the raspicam function via the command line. To start the program, boot up the Raspberry PI and run the .bin file in Desktop → Chessstuff → Camerafun.bin. The images a.ppm and b.ppm are also created in this folder, showing the output of the blue and red filters. Whenever the .bin file is running, it will wait for the ARM board to request an image. At this point, the robot is ready to start playing chess.

IX. Appendix B – Arm M4 Source Code

The ARM code is compiled in the Keil μVision IDE and uses header files to sort functions.

MAIN.c

```
//Sean Vibbert
//Chess Playing Robot
//MAIN.c
//Last Modified: 4/17/19
//This code runs the whole project and interfaces with all other header files. Some lines of code are irrelevant and are used for testing.
//In its current state, it operates the robot as normal.
#include "Movement.h"
#include "ChessLogic.h"
#include "PB10to15Communication.h"
#define ActuatorMax 10000
#define CAMERAPI 0xABABABAB
#define ENDPI 0xABCDABCD
unsigned int ServoLength = 50;//Global variable
unsigned int ActuatorLength = 38;
unsigned int GO = 0;
unsigned int GO2 = 0;
unsigned long CommandData = 0;
unsigned long Blue1 = 0;
unsigned long Red1 = 0;
unsigned long Blue2 = 0;
unsigned long Red2 = 0;
unsigned int randi1 = 100;
int r1;
int Test1 = 1;
int StepLeftIn = 0;
int StepRightIn = 0;
int StepClawIn = 0;
int StepBothIn = 0;
unsigned int ActuatorIn = 0;
int CurrentKey = 0;
int FunctionMode = 1;
int FunctionDone = 0;
int Func2Progress = 0;

int main()
{
PC6_PWM_INIT();//Initialize all headers
MagnetEnable();
StepperInit();
KeypadEnable();
CommunicateSetup();
SetupBoard();
AllOff();

//Main program loop
while(1)
{
          if (Test1 == 1)
          {
                    while(FunctionMode/4 != 3)
                    {
                    randi1++;
                    PC6_PWM_SETDUTY(ActuatorLength*100); //Constantly sets the position of the actuator. 0-5V PWM
                    //Do basic functions
                    if (FunctionMode/4 == 1 && FunctionDone == 0 && CurrentKey != 0)
                    {
                              AllOn();
                              LogicWait(1);
                              if (CurrentKey == 2)
                                        Ymove(0.1,0);
                              if (CurrentKey == 5)
                                        Xmove(0.1,0);
                              if (CurrentKey == 7)
                                        Xmove(-0.1,0);
                              if (CurrentKey == 10)
                                        Ymove(-0.1,0);
                              FunctionDone = 1;
                              AllOff();
                    }

                    if (FunctionMode/4 == 4)
                    {
                              FunctionMode = 1;
                              GO2 = 1;
                    }
                    //Getting Keypad always
                    if (CurrentKey == 0)
                    {
                              CurrentKey = GetKeypad();
                              if (CurrentKey == 4 || CurrentKey == 8 || CurrentKey == 12 || CurrentKey == 16)
                                        FunctionMode = CurrentKey;
```

```
			}
			if (CurrentKey != 0)
			{
				if (GetKeypad() == 0)
				{
				CurrentKey = 0;
				FunctionDone = 0;
				}
			}
			//End waiting for button
		}
		FunctionMode = 0;
		unsigned long DataTest = CAMERAPI;
		Send32Bits(DataTest);
		Blue1 = Get32Bits();//Get blue content
		Blue2 = Get32Bits();
		Red1 = Get32Bits();
		Red2 = Get32Bits();
		//Set into board
		for(int i=0;i<4;i++)
		{
			for(int j=7;j>=0;j-=1)
			{
				if ((Blue1 & 1) == 1)
					NewBoard[7-i][j] = 1;
				else if ((Red1 & 1) == 1)
					NewBoard[7-i][j] = 2;
				else
					NewBoard[7-i][j] = 0;
				Blue1 = Blue1 & ~(1);
				Red1 = Red1 & ~(1);
				Blue1  = Blue1 >> 1;
				Red1 = Red1 >> 1;
			}
		}
		for(int i=4;i<8;i++)
		{
			for(int j=7;j>=0;j-=1)
			{
				if ((Blue2 & 1) == 1)
					NewBoard[7-i][j] = 1;
				else if ((Red2 & 1) == 1)
					NewBoard[7-i][j] = 2;
				else
					NewBoard[7-i][j] = 0;
				Blue2 = Blue2 & ~(1);
				Red2 = Red2 & ~(1);
				Blue2 = Blue2 >> 1;
				Red2 = Red2 >> 1;
			}
		}
		CheckBoardChange();
		GetLegalMoves();
		int Capture = 0;
		TmpLegalNumber = 0;
		for(int i=0;i<LegalNumber;i++)
		{
			if(LegalTake[i][0] == 1)//Going to capture alwyas
			{
				Capture = 1;
				TmpLegalMoves[TmpLegalNumber][0] = LegalMoves[i][0];
				TmpLegalMoves[TmpLegalNumber][1] = LegalMoves[i][1];
				TmpLegalMoves[TmpLegalNumber][2] = LegalMoves[i][2];
				TmpLegalMoves[TmpLegalNumber][3] = LegalMoves[i][3];
				TmpLegalTake[TmpLegalNumber][0] = LegalTake[i][0];
				TmpLegalTake[TmpLegalNumber][1] = LegalTake[i][1];
				TmpLegalNumber++;
			}
		}
		if (Capture == 0)
		{
			r1 = randi1 % LegalNumber;
			int row1 = LegalMoves[r1][0];
			int row2 = LegalMoves[r1][1];
			int col1 = LegalMoves[r1][2];
			int col2 = LegalMoves[r1][3];
			MovePiece(col1,row1,ReturnPieceIdentity(row1,col1),col2,row2,LegalTake[r1][0],LegalTake[r1][1]);
			VirtualMovePiece(row1,col1,row2,col2);
		}
		else
		{
			r1 = randi1 % TmpLegalNumber;
			int row1 = TmpLegalMoves[r1][0];
			int row2 = TmpLegalMoves[r1][1];
			int col1 = TmpLegalMoves[r1][2];
			int col2 = TmpLegalMoves[r1][3];
			MovePiece(col1,row1,ReturnPieceIdentity(row1,col1),col2,row2,TmpLegalTake[r1][0],TmpLegalTake[r1][1]);
			VirtualMovePiece(row1,col1,row2,col2);
		}
		//End of demonstration program
		}
	}
}
```

ChessLogic.h

```
#include "stm32f446.h"
//Sean Vibbert
//Chess Playing Robot
//ChessLogic.h
//Last Modified: 4/17/19
//This header file contains all the logic for the game of chess
int WhitePawnArray[8][4];//Row, column, still exists, identity
int BlackPawnArray[8][4];
int WhiteKnightArray[2][4];
int BlackKnightArray[2][4];
int WhiteBishopArray[2][4];
int BlackBishopArray[2][4];
int WhiteRookArray[2][4];//Row, column, still exists
int BlackRookArray[2][4];
int WhiteQueenArray[1][4];
int BlackQueenArray[1][4];
int WhiteKingArray[1][4];//Row, column, still exists
int BlackKingArray[1][4];
int OldBoard[8][8];
int NewBoard[8][8];
int LegalMoves[800][4];//Row1, col1, row2, col2
int LegalTake[800][2];//is it a take? what piece take type is it
int LegalNumber = 0;
int TmpLegalMoves[100][4];
int TmpLegalTake[100][2];
int TmpLegalNumber = 0;
int firstrow = 4-1;
int firstcol = 4-1;
int secondrow = 5-1;
int secondcol = 5-1;
int piece = 0;
int take = 1;
int taketype = 0;
void SetupBoard(void);
void CheckBoardChange(void);
void GetLegalMoves(void);
int ReturnPieceIdentity(int row, int col);
void VirtualMovePiece(int oldrow, int oldcol, int newrow, int newcol);
void SetupBoard()
{
           for(int i=0;i<8;i++)
           {
                      OldBoard[0][i] = 1;//Initial setup of board
                      OldBoard[1][i] = 1;
                      OldBoard[6][i] = 2;
                      OldBoard[7][i] = 2;
                      NewBoard[0][i] = 1;//Initial setup of board
                      NewBoard[1][i] = 1;
                      NewBoard[6][i] = 2;
                      NewBoard[7][i] = 2;
           }

           for(int i=0;i<8;i++)//Pawns 1-8
           {
                      WhitePawnArray[i][0] = 1;//Rows
                      WhitePawnArray[i][1] = i;//Columns
                      WhitePawnArray[i][2] = 1;//Still Exists
                      WhitePawnArray[i][3] = 0;//Identity
                      BlackPawnArray[i][0] = 6;//Rows
                      BlackPawnArray[i][1] = i;//Columns
                      BlackPawnArray[i][2] = 1;//Still Exists
                      BlackPawnArray[i][3] = 0;//Identity
           }

           WhiteKnightArray[0][0] = 0;//Row //White Knight #1
           WhiteKnightArray[0][1] = 1;//Column
           WhiteKnightArray[0][2] = 1;//Still Exists
           WhiteKnightArray[0][3] = 1;//Identity
           WhiteKnightArray[1][0] = 0;//Row //White Knight #2
           WhiteKnightArray[1][1] = 6;//Column
           WhiteKnightArray[1][2] = 1;//Still Exists
           WhiteKnightArray[1][3] = 1;//Identity
           BlackKnightArray[0][0] = 7;//Row //Black Knight #1
           BlackKnightArray[0][1] = 1;//Column
           BlackKnightArray[0][2] = 1;//Still Exists
           BlackKnightArray[0][3] = 1;//Identity
           BlackKnightArray[1][0] = 7;//Row //Black Knight #2
           BlackKnightArray[1][1] = 6;//Column
           BlackKnightArray[1][2] = 1;//Still Exists
           BlackKnightArray[1][3] = 1;//Identity
           WhiteBishopArray[0][0] = 0;//Row //White Bishop #1
           WhiteBishopArray[0][1] = 2;//Column
           WhiteBishopArray[0][2] = 1;//Still Exists
           WhiteBishopArray[0][3] = 2;//Identity
           WhiteBishopArray[1][0] = 0;//Row //White Bishop #2
           WhiteBishopArray[1][1] = 5;//Column
           WhiteBishopArray[1][2] = 1;//Still Exists
           WhiteBishopArray[1][3] = 2;//Identity
           BlackBishopArray[0][0] = 7;//Row //Black Bishop #1
           BlackBishopArray[0][1] = 2;//Column
           BlackBishopArray[0][2] = 1;//Still Exists
           BlackBishopArray[0][3] = 2;//Identity
           BlackBishopArray[1][0] = 7;//Row //Black Bihsop #2
           BlackBishopArray[1][1] = 5;//Column
```

```
            BlackBishopArray[1][2] = 1;//Still Exists
            BlackBishopArray[1][3] = 2;//Identity
            WhiteRookArray[0][0] = 0;//Row //White Rook #1
            WhiteRookArray[0][1] = 0;//Column
            WhiteRookArray[0][2] = 1;//Still Exists
            WhiteRookArray[0][3] = 3;//Identity
            WhiteRookArray[1][0] = 0;//Row //White Rook #2
            WhiteRookArray[1][1] = 7;//Column
            WhiteRookArray[1][2] = 1;//Still Exists
            WhiteRookArray[1][3] = 3;//Identity
            BlackRookArray[0][0] = 7;//Row //Black Rook #1
            BlackRookArray[0][1] = 0;//Column
            BlackRookArray[0][2] = 1;//Still Exists
            BlackRookArray[0][3] = 3;//Identity
            BlackRookArray[1][0] = 7;//Row //Black Rook #2
            BlackRookArray[1][1] = 7;//Column
            BlackRookArray[1][2] = 1;//Still Exists
            BlackRookArray[1][3] = 3;//Identity
            WhiteQueenArray[0][0] = 0;//Row //White Queen
            WhiteQueenArray[0][1] = 4;//Column
            WhiteQueenArray[0][2] = 1;//Still Exists
            WhiteQueenArray[0][3] = 4;//Identity
            BlackQueenArray[0][0] = 7;//Row //Black Queen
            BlackQueenArray[0][1] = 4;//Column
            BlackQueenArray[0][2] = 1;//Still Exists
            BlackQueenArray[0][3] = 4;//Identity
            WhiteKingArray[0][0] = 0;//Row //White King
            WhiteKingArray[0][1] = 3;//Column
            WhiteKingArray[0][2] = 1;//Still Exists
            WhiteKingArray[0][3] = 5;//Identity
            BlackKingArray[0][0] = 7;//Row //Black king
            BlackKingArray[0][1] = 3;//Column
            BlackKingArray[0][2] = 1;//Still Exists
            BlackKingArray[0][3] = 5;//Identity
}

//int LegalMoves[500][4];//Row1, col1, row2, col2
//int LegalTake[500][2];//is it a take? what piece take type is it
//int LegalNumber = 0;

void GetLegalMoves()//It is playing white, so it moves first
{
            LegalNumber = 0;
            int row,col;
            int stop = 0;
            //Pawn
            for(int i=0;i<8;i++)
            {
                        row = WhitePawnArray[i][0];
                        col = WhitePawnArray[i][1];
            if (WhitePawnArray[i][2] == 1)//Must exist. Currently forcing move of knight
            {
                        if (row < 7)
                        {
                        if (NewBoard[row+1][col] == 0)//One space ahead is free
                                    {
                                    LegalMoves[LegalNumber][0] = row;//row1
                                    LegalMoves[LegalNumber][1] = row+1;//row1
                                    LegalMoves[LegalNumber][2] = col;//row1
                                    LegalMoves[LegalNumber][3] = col;//row1
                                    LegalTake[LegalNumber][0] = 0;
                                    LegalNumber++;
                                    }
                        }

                        if (row < 6)
                        {
                        if (NewBoard[row+1][col] == 0 && NewBoard[row+2][col] == 0 && row == 1)//Two space ahead is free
                                    {
                                    LegalMoves[LegalNumber][0] = row;//row1
                                    LegalMoves[LegalNumber][1] = row+2;//row1
                                    LegalMoves[LegalNumber][2] = col;//row1
                                    LegalMoves[LegalNumber][3] = col;//row1
                                    LegalTake[LegalNumber][0] = 0;
                                    LegalNumber++;
                                    }
                        }

                        if (row < 7 && col > 0)
                        {
                        if (NewBoard[row+1][col-1] == 2)//Taking right
                                    {
                                    LegalMoves[LegalNumber][0] = row;//row1
                                    LegalMoves[LegalNumber][1] = row+1;//row1
                                    LegalMoves[LegalNumber][2] = col;//row1
                                    LegalMoves[LegalNumber][3] = col-1;//row1
                                    LegalTake[LegalNumber][0] = 1;
                                    LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+1,col-1);
                                    LegalNumber++;
                                    }
                        }

                        if (row < 7 && col < 7)
                        {
                        if (NewBoard[row+1][col+1] == 2)//Taking right
                                    {
```

```
                                LegalMoves[LegalNumber][0] = row;//row1
                                LegalMoves[LegalNumber][1] = row+1;//row1
                                LegalMoves[LegalNumber][2] = col;//row1
                                LegalMoves[LegalNumber][3] = col+1;//row1
                                LegalTake[LegalNumber][0] = 1;
                                LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+1,col+1);
                                LegalNumber++;
                                }
                     }
           }
}

//Knight
for(int i=0;i<2;i++)
{
   row = WhiteKnightArray[i][0];
   col = WhiteKnightArray[i][1];
   //-60deg
            if (WhiteKnightArray[i][2] == 1)//Must exist
            {
   if(row > 1 && col > 0)//safety check
   {
      if(NewBoard[row-2][col-1] == 0)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-2;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-1;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
      }

      if(NewBoard[row-2][col-1] == 2)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-2;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-1;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-2,col-1);
         LegalNumber++;
      }
   }

   if(row > 0 && col > 1)//-30deg
   {
      if(NewBoard[row-1][col-2] == 0)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-2;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
      }

      if(NewBoard[row-1][col-2] == 2)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-2;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-1,col-2);
         LegalNumber++;
      }
   }

   if(row < 7 && col > 1)//+30deg
   {
      if(NewBoard[row+1][col-2] == 0)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-2;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
      }

      if(NewBoard[row+1][col-2] == 2)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-2;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+1,col-2);
         LegalNumber++;
      }
   }


   if(row < 6 && col > 0)//+60deg
   {
      if(NewBoard[row+2][col-1] == 0)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+2;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-1;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
      }

      if(NewBoard[row+2][col-1] == 2)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+2;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-1;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+2,col-1);
         LegalNumber++;
      }
   }

   if(row < 6 && col < 7)//+120deg
   {
      if(NewBoard[row+2][col+1] == 0)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+2;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+1;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
      }

      if(NewBoard[row+2][col+1] == 2)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+2;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+1;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+2,col+1);
         LegalNumber++;
      }
   }

   if(row < 7 && col < 6)//+150deg
   {
      if(NewBoard[row+1][col+2] == 0)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+2;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
      }

      if(NewBoard[row+1][col+2] == 2)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+2;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+1,col+2);
         LegalNumber++;
      }
   }

   if(row > 0 && col < 6)//+210deg
   {
      if(NewBoard[row-1][col+2] == 0)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+2;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
      }

      if(NewBoard[row-1][col+2] == 2)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+2;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-1,col+2);
         LegalNumber++;
      }
   }

   if(row > 1 && col < 7)//+240deg
   {
      if(NewBoard[row-2][col+1] == 0)
```

```
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-2;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+1;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
      }

      if(NewBoard[row-2][col+1] == 2)
      {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-2;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+1;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-2,col+1);
         LegalNumber++;
      }
   }
 }
}

//Bishop
for(int i=0;i<2;i++)
{
   row = WhiteBishopArray[i][0];
   col = WhiteBishopArray[i][1];
   stop = 0;
         if (WhiteBishopArray[i][2] == 1)//Must exist
         {
   for(int j=1;j<8;j++)
   {
      //135 deg diagonal
      if(row + j < 8 && col + j < 8 && stop == 0)
      {
         if(NewBoard[row+j][col+j] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+j;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
            else if(NewBoard[row+j][col+j] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+j;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+j,col+j);
         LegalNumber++;
         stop = 1;
         }
            else
            stop = 1;
      }
   }

      //45 deg diagonal
      stop = 0;
   for(int j=1;j<8;j++)
   {
      if(row + j < 8 && col - j >= 0 && stop == 0)
      {
         if(NewBoard[row+j][col-j] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-j;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row+j][col-j] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-j;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+j,col-j);
         LegalNumber++;
         stop = 1;
         }
         else
            stop = 1;
      }
   }

      //-45 deg diagonal
      stop = 0;
   for(int j=1;j<8;j++)
   {
      if(row - j >= 0 && col - j >= 0 && stop == 0)
      {
         if(NewBoard[row-j][col-j] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-j;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row-j][col-j] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-j;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-j,col-j);
         LegalNumber++;
         stop = 1;
         }
         else
            stop = 1;
      }
   }

      //215 deg diagonal
      stop = 0;
   for(int j=1;j<8;j++)
   {
      if(row - j >= 0 && col + j < 8 && stop == 0)
      {
         if(NewBoard[row-j][col+j] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+j;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row-j][col+j] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+j;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-j,col+j);
         LegalNumber++;
         stop = 1;
         }
         else
            stop = 1;
      }
   }
 }
}

//Rook
for(int i=0;i<2;i++)
{
   row = WhiteRookArray[i][0];
   col = WhiteRookArray[i][1];
   stop = 0;
         if (WhiteRookArray[i][2] == 1)//Must exist
         {
   for(int j=1;j<8;j++)
   {
      //180 deg left
      if(col + j < 8 && stop == 0)
      {
         if(NewBoard[row][col+j] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+j;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
            else if(NewBoard[row][col+j] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+j;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row,col+j);
         LegalNumber++;
         stop = 1;
         }
            else
            stop = 1;
      }
```

```
    }

    //0 deg right
    stop = 0;
    for(int j=1;j<8;j++)
    {
       if(col - j >= 0 && stop == 0)
       {
          if(NewBoard[row][col-j] == 0)
          {
          LegalMoves[LegalNumber][0] = row;
          LegalMoves[LegalNumber][1] = row;
          LegalMoves[LegalNumber][2] = col;
          LegalMoves[LegalNumber][3] = col-j;
          LegalTake[LegalNumber][0] = -1;
          LegalNumber++;
          }
          else if(NewBoard[row][col-j] == 2)
          {
          LegalMoves[LegalNumber][0] = row;
          LegalMoves[LegalNumber][1] = row;
          LegalMoves[LegalNumber][2] = col;
          LegalMoves[LegalNumber][3] = col-j;
          LegalTake[LegalNumber][0] = 1;
          LegalTake[LegalNumber][1] = ReturnPieceIdentity(row,col-j);
          LegalNumber++;
          stop = 1;
          }
          else
             stop = 1;
       }
    }

    //90 deg up
    stop = 0;
    for(int j=1;j<8;j++)
    {
       if(row + j < 8 && stop == 0)
       {
          if(NewBoard[row+j][col] == 0)
          {
          LegalMoves[LegalNumber][0] = row;
          LegalMoves[LegalNumber][1] = row+j;
          LegalMoves[LegalNumber][2] = col;
          LegalMoves[LegalNumber][3] = col;
          LegalTake[LegalNumber][0] = -1;
          LegalNumber++;
          }
          else if(NewBoard[row+j][col] == 2)
          {
          LegalMoves[LegalNumber][0] = row;
          LegalMoves[LegalNumber][1] = row+j;
          LegalMoves[LegalNumber][2] = col;
          LegalMoves[LegalNumber][3] = col;
          LegalTake[LegalNumber][0] = 1;
          LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+j,col);
          LegalNumber++;
          stop = 1;
          }
          else
             stop = 1;
       }
    }

    //-90 deg down
    stop = 0;
    for(int j=1;j<8;j++)
    {
       if(row - j >= 0 && stop == 0)
       {
          if(NewBoard[row-j][col] == 0)
          {
          LegalMoves[LegalNumber][0] = row;
          LegalMoves[LegalNumber][1] = row-j;
          LegalMoves[LegalNumber][2] = col;
          LegalMoves[LegalNumber][3] = col;
          LegalTake[LegalNumber][0] = -1;
          LegalNumber++;
          }
          else if(NewBoard[row-j][col] == 2)
          {
          LegalMoves[LegalNumber][0] = row;
          LegalMoves[LegalNumber][1] = row-j;
          LegalMoves[LegalNumber][2] = col;
          LegalMoves[LegalNumber][3] = col;
          LegalTake[LegalNumber][0] = 1;
          LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-j,col);
          LegalNumber++;
          stop = 1;
          }
          else
             stop = 1;
       }
    }
  }
}
```

```
//Queen
   row = WhiteQueenArray[0][0];
   col = WhiteQueenArray[0][1];
   stop = 0;
            if (WhiteQueenArray[0][2] == 1)//Must exist
            {
   for(int j=1;j<8;j++)
   {
      //135 deg diagonal
      if(row + j < 8 && col + j < 8 && stop == 0)
      {
         if(NewBoard[row+j][col+j] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+j;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
            else if(NewBoard[row+j][col+j] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+j;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+j,col+j);
         LegalNumber++;
         stop = 1;
         }
            else
            stop = 1;
      }
   }

   //45 deg diagonal
   stop = 0;
   for(int j=1;j<8;j++)
   {
      if(row + j < 8 && col - j >= 0 && stop == 0)
      {
         if(NewBoard[row+j][col-j] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-j;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row+j][col-j] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-j;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+j,col-j);
         LegalNumber++;
         stop = 1;
         }
         else
            stop = 1;
      }
   }

   //-45 deg diagonal
   stop = 0;
   for(int j=1;j<8;j++)
   {
      if(row - j >= 0 && col - j >= 0 && stop == 0)
      {
         if(NewBoard[row-j][col-j] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-j;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row-j][col-j] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-j;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-j,col-j);
         LegalNumber++;
         stop = 1;
         }
         else
            stop = 1;
   }
}

//215 deg diagonal
stop = 0;
for(int j=1;j<8;j++)
{
   if(row - j >= 0 && col + j < 8 && stop == 0)
   {
      if(NewBoard[row-j][col+j] == 0)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row-j;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col+j;
      LegalTake[LegalNumber][0] = -1;
      LegalNumber++;
      }
      else if(NewBoard[row-j][col+j] == 2)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row-j;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col+j;
      LegalTake[LegalNumber][0] = 1;
      LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-j,col+j);
      LegalNumber++;
      stop = 1;
      }
      else
         stop = 1;
   }
}

stop = 0;
for(int j=1;j<8;j++)
{
   //180 deg left
   if(col + j < 8 && stop == 0)
   {
      if(NewBoard[row][col+j] == 0)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col+j;
      LegalTake[LegalNumber][0] = -1;
      LegalNumber++;
      }
         else if(NewBoard[row][col+j] == 2)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col+j;
      LegalTake[LegalNumber][0] = 1;
      LegalTake[LegalNumber][1] = ReturnPieceIdentity(row,col+j);
      LegalNumber++;
      stop = 1;
      }
         else
         stop = 1;
   }
}

//0 deg right
stop = 0;
for(int j=1;j<8;j++)
{
   if(col - j >= 0 && stop == 0)
   {
      if(NewBoard[row][col-j] == 0)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col-j;
      LegalTake[LegalNumber][0] = -1;
      LegalNumber++;
      }
      else if(NewBoard[row][col-j] == 2)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col-j;
      LegalTake[LegalNumber][0] = 1;
      LegalTake[LegalNumber][1] = ReturnPieceIdentity(row,col-j);
      LegalNumber++;
      stop = 1;
      }
      else
         stop = 1;
   }
}
```

```
      //90 deg up
      stop = 0;
   for(int j=1;j<8;j++)
   {
      if(row + j < 8 && stop == 0)
      {
         if(NewBoard[row+j][col] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row+j][col] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row+j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+j,col);
         LegalNumber++;
         stop = 1;
         }
         else
            stop = 1;
      }
   }

   //-90 deg down
   stop = 0;
   for(int j=1;j<8;j++)
   {
      if(row - j >= 0 && stop == 0)
      {
         if(NewBoard[row-j][col] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row-j][col] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-j;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-j,col);
         LegalNumber++;
         stop = 1;
         }
         else
            stop = 1;
      }
                  }
   }

//King //Currently no castling or check implementation

   row = WhiteKingArray[0][0];
   col = WhiteKingArray[0][1];
         if (WhiteKingArray[0][2] == 1)//Must exist
         {
   //0 deg
   if (col > 0)
   {
         if(NewBoard[row][col-1] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-1;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row][col-1] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-1;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row,col-1);
         LegalNumber++;
         }
   }

   //45 deg
   if (row < 7 && col > 0)
   {
      if(NewBoard[row+1][col-1] == 0)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row+1;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col-1;
      LegalTake[LegalNumber][0] = -1;
      LegalNumber++;
      }
      else if(NewBoard[row+1][col-1] == 2)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row+1;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col-1;
      LegalTake[LegalNumber][0] = 1;
      LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+1,col-1);
      LegalNumber++;
      }
}

//90 deg
if (row < 7)
{
      if(NewBoard[row+1][col] == 0)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row+1;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col;
      LegalTake[LegalNumber][0] = -1;
      LegalNumber++;
      }
      else if(NewBoard[row+1][col] == 2)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row+1;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col;
      LegalTake[LegalNumber][0] = 1;
      LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+1,col);
      LegalNumber++;
      }
}

//135 deg
if (row < 7 && col < 7)
{
      if(NewBoard[row+1][col+1] == 0)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row+1;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col+1;
      LegalTake[LegalNumber][0] = -1;
      LegalNumber++;
      }
      else if(NewBoard[row+1][col+1] == 2)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row+1;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col+1;
      LegalTake[LegalNumber][0] = 1;
      LegalTake[LegalNumber][1] = ReturnPieceIdentity(row+1,col+1);
      LegalNumber++;
      }
}

//180 deg
if (col < 7)
{
      if(NewBoard[row][col+1] == 0)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col+1;
      LegalTake[LegalNumber][0] = -1;
      LegalNumber++;
      }
      else if(NewBoard[row][col+1] == 2)
      {
      LegalMoves[LegalNumber][0] = row;
      LegalMoves[LegalNumber][1] = row;
      LegalMoves[LegalNumber][2] = col;
      LegalMoves[LegalNumber][3] = col+1;
      LegalTake[LegalNumber][0] = 1;
      LegalTake[LegalNumber][1] = ReturnPieceIdentity(row,col+1);
      LegalNumber++;
      }
}

//225 deg
if (row > 0 && col < 7)
{
```

```
         if(NewBoard[row-1][col+1] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+1;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row-1][col+1] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col+1;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-1,col+1);
         LegalNumber++;
         }
   }

   //-90 deg
   if (row > 0)
   {
         if(NewBoard[row-1][col] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row-1][col] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-1,col);
         LegalNumber++;
         }
   }

   //-45 deg
   if (row > 0 && col > 0)
   {
         if(NewBoard[row-1][col-1] == 0)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-1;
         LegalTake[LegalNumber][0] = -1;
         LegalNumber++;
         }
         else if(NewBoard[row-1][col-1] == 2)
         {
         LegalMoves[LegalNumber][0] = row;
         LegalMoves[LegalNumber][1] = row-1;
         LegalMoves[LegalNumber][2] = col;
         LegalMoves[LegalNumber][3] = col-1;
         LegalTake[LegalNumber][0] = 1;
         LegalTake[LegalNumber][1] = ReturnPieceIdentity(row-1,col-1);
         LegalNumber++;
         }
   }
 }
 }

int ReturnPieceIdentity(int row, int col)//Returning piece type at index
{
   for(int i=0;i<8;i++)
   {
      if (WhitePawnArray[i][0] == row && WhitePawnArray[i][1] == col)
      {
      return WhitePawnArray[i][3];
      }
   }

            for(int i=0;i<8;i++)
   {
      if (BlackPawnArray[i][0] == row && BlackPawnArray[i][1] == col)
      {
      return BlackPawnArray[i][3];
      }
   }

   for(int i=0;i<2;i++)
   {
      if (WhiteKnightArray[i][0] == row && WhiteKnightArray[i][1] == col)
      {
      return WhiteKnightArray[i][3];
      }
   }
```

```
            for(int i=0;i<2;i++)
   {
      if (BlackKnightArray[i][0] == row && BlackKnightArray[i][1] == col)
      {
      return BlackKnightArray[i][3];
      }
   }

   for(int i=0;i<2;i++)
   {
      if (WhiteBishopArray[i][0] == row && WhiteBishopArray[i][1] == col)
      {
      return WhiteBishopArray[i][3];
      }
   }

            for(int i=0;i<2;i++)
   {
      if (BlackBishopArray[i][0] == row && BlackBishopArray[i][1] == col)
      {
      return BlackBishopArray[i][3];
      }
   }


   for(int i=0;i<2;i++)
   {
      if (WhiteRookArray[i][0] == row && WhiteRookArray[i][1] == col)
      {
      return WhiteRookArray[i][3];
      }
   }

            for(int i=0;i<2;i++)
   {
      if (BlackRookArray[i][0] == row && BlackRookArray[i][1] == col)
      {
      return BlackRookArray[i][3];
      }
   }

      if (WhiteQueenArray[0][0] == row && WhiteQueenArray[0][1] == col)
      {
      return WhiteQueenArray[0][3];
      }

                                 if (BlackQueenArray[0][0] == row && BlackQueenArray[0][1] == col)
      {
      return BlackQueenArray[0][3];
      }

      if (WhiteKingArray[0][0] == row && WhiteKingArray[0][1] == col)
      {
      return WhiteKingArray[0][3];
      }

                                 if (BlackKingArray[0][0] == row && BlackKingArray[0][1] == col)
      {
      return BlackKingArray[0][3];
      }

return -1;
}

void VirtualMovePiece(int oldrow, int oldcol, int newrow, int newcol)//Moving board virtually then forcing a recheck
{
           //Only manually moves white
           NewBoard[oldrow][oldcol] = 0;
           NewBoard[newrow][newcol] = 1;

           CheckBoardChange();
}


void CheckBoardChange()
{
           int MissingRow = -2;
           int MissingCol = -2;
           int NewRow = -2;
           int NewCol = -2;

           int ColorChangeRow = -2;
           int ColorChangeCol = -2;

           int take = 0;

           for(int i=0;i<8;i++)//Check for move made by rows and columns
           {
                      for(int j=0;j<8;j++)
                      {
                                 if (OldBoard[i][j] != NewBoard[i][j])
                                 {
                                            if (OldBoard[i][j] == 1 && NewBoard[i][j] == 0)//Piece missing (white)
                                            {
                                                       MissingRow = i;
```

```
                        MissingCol = j;
                }
                else if (OldBoard[i][j] == 2 && NewBoard[i][j] == 0)//Piece missing (black)
                {
                        MissingRow = i;
                        MissingCol = j;
                }
                else if (OldBoard[i][j] == 0 && NewBoard[i][j] == 1)//Piece new (white)
                {
                        NewRow = i;
                        NewCol = j;
                }
                else if (OldBoard[i][j] == 0 && NewBoard[i][j] == 2)//Piece new (black)
                {
                        NewRow = i;
                        NewCol = j;
                }
                else if (OldBoard[i][j] == 1 && NewBoard[i][j] == 2)//Piece change (white to black)
                {
                        ColorChangeRow = i;
                        ColorChangeCol = j;
                        take = 1;
                }
                else if (OldBoard[i][j] == 2 && NewBoard[i][j] == 1)//Piece change (black to white)
                {
                        ColorChangeRow = i;
                        ColorChangeCol = j;
                        take = 1;
                }
            }
        }
    }

                //General form for change in piece location-----------------------------PAWN
                for(int i=0;i<8;i++)
                {
                        if (WhitePawnArray[i][0] == ColorChangeRow && WhitePawnArray[i][1] == ColorChangeCol)
//Piece removed from game
                        {
                            WhitePawnArray[i][0] = -1;//Outside board
                            WhitePawnArray[i][1] = -1;//Outside board
                            WhitePawnArray[i][2] = 0;//Does not exist

                        }
                        else if (BlackPawnArray[i][0] == ColorChangeRow && BlackPawnArray[i][1] == ColorChangeCol)
                        {
                            BlackPawnArray[i][0] = -1;//Outside board
                            BlackPawnArray[i][1] = -1;//Outside board
                            BlackPawnArray[i][2] = 0;//Does not exist

                        }

                        if (take == 0)
                        {
                            if (WhitePawnArray[i][0] == MissingRow && WhitePawnArray[i][1] == MissingCol)
//Piece moving to a nontaken square
                            {
                                WhitePawnArray[i][0] = NewRow;//New square
                                WhitePawnArray[i][1] = NewCol;//
                            }
                            else if (BlackPawnArray[i][0] == MissingRow && BlackPawnArray[i][1] == MissingCol)
                            {
                                BlackPawnArray[i][0] = NewRow;//New square
                                BlackPawnArray[i][1] = NewCol;//
                            }
                        }
                        else
                        {
                            if (WhitePawnArray[i][0] == MissingRow && WhitePawnArray[i][1] == MissingCol)
//Piece moving to a taken square
                            {
                                WhitePawnArray[i][0] = ColorChangeRow;//Outside board
                                WhitePawnArray[i][1] = ColorChangeCol;//Outside board

                            }
                            else if (BlackPawnArray[i][0] == MissingRow && BlackPawnArray[i][1] == MissingCol)
                            {
                                BlackPawnArray[i][0] = ColorChangeRow;//Outside board
                                BlackPawnArray[i][1] = ColorChangeCol;//Outside board

                            }
                        }
                }
                        //-------------------------------------------------------------------

                //General form for change in piece location-----------------------------Knight
                for(int i=0;i<2;i++)
                {
                        if (WhiteKnightArray[i][0] == ColorChangeRow && WhiteKnightArray[i][1] == ColorChangeCol)
//Piece removed from game
                        {
                            WhiteKnightArray[i][0] = -1;//Outside board
                            WhiteKnightArray[i][1] = -1;//Outside board
                            WhiteKnightArray[i][2] = 0;//Does not exist

                        }
				else if (BlackKnightArray[i][0] == ColorChangeRow && BlackKnightArray[i][1] == ColorChangeCol)
				{
					BlackKnightArray[i][0] = -1;//Outside board
					BlackKnightArray[i][1] = -1;//Outside board
					BlackKnightArray[i][2] = 0;//Does not exist

				}

				if (take == 0)
				{
					if (WhiteKnightArray[i][0] == MissingRow && WhiteKnightArray[i][1] ==
MissingCol)//Piece moving to a nontaken square
					{
						WhiteKnightArray[i][0] = NewRow;//New square
						WhiteKnightArray[i][1] = NewCol;//
					}
					else if (BlackKnightArray[i][0] == MissingRow && BlackKnightArray[i][1] ==
MissingCol)
					{
						BlackKnightArray[i][0] = NewRow;//New square
						BlackKnightArray[i][1] = NewCol;//
					}
				}
				else
				{
					if (WhiteKnightArray[i][0] == MissingRow && WhiteKnightArray[i][1] ==
MissingCol)//Piece moving to a taken square
					{
						WhiteKnightArray[i][0] = ColorChangeRow;//Outside board
						WhiteKnightArray[i][1] = ColorChangeCol;//Outside board

					}
					else if (BlackKnightArray[i][0] == MissingRow && BlackKnightArray[i][1] ==
MissingCol)
					{
						BlackKnightArray[i][0] = ColorChangeRow;//Outside board
						BlackKnightArray[i][1] = ColorChangeCol;//Outside board

					}
				}
			}
				//-------------------------------------------------------------------

			//General form for change in piece location-----------------------------Bishop
			for(int i=0;i<2;i++)
			{
				if (WhiteBishopArray[i][0] == ColorChangeRow && WhiteBishopArray[i][1] == ColorChangeCol)
//Piece removed from game
				{
					WhiteBishopArray[i][0] = -1;//Outside board
					WhiteBishopArray[i][1] = -1;//Outside board
					WhiteBishopArray[i][2] = 0;//Does not exist

				}
				else if (BlackBishopArray[i][0] == ColorChangeRow && BlackBishopArray[i][1] == ColorChangeCol)
				{
					BlackBishopArray[i][0] = -1;//Outside board
					BlackBishopArray[i][1] = -1;//Outside board
					BlackBishopArray[i][2] = 0;//Does not exist

				}

				if (take == 0)
				{
					if (WhiteBishopArray[i][0] == MissingRow && WhiteBishopArray[i][1] ==
MissingCol)//Piece moving to a nontaken square
					{
						WhiteBishopArray[i][0] = NewRow;//New square
						WhiteBishopArray[i][1] = NewCol;//
					}
					else if (BlackBishopArray[i][0] == MissingRow && BlackBishopArray[i][1] ==
MissingCol)
					{
						BlackBishopArray[i][0] = NewRow;//New square
						BlackBishopArray[i][1] = NewCol;//
					}
				}
				else
				{
					if (WhiteBishopArray[i][0] == MissingRow && WhiteBishopArray[i][1] ==
MissingCol)//Piece moving to a taken square
					{
						WhiteBishopArray[i][0] = ColorChangeRow;//Outside board
						WhiteBishopArray[i][1] = ColorChangeCol;//Outside board

					}
					else if (BlackBishopArray[i][0] == MissingRow && BlackBishopArray[i][1] ==
MissingCol)
					{
						BlackBishopArray[i][0] = ColorChangeRow;//Outside board
						BlackBishopArray[i][1] = ColorChangeCol;//Outside board

					}
				}
			}
			//-------------------------------------------------------------------
```

```
                                            //General form for change in piece location-----------------------------Rook
                                            for(int i=0;i<2;i++)
                                            {
                                                        if (WhiteRookArray[i][0] == ColorChangeRow && WhiteRookArray[i][1] == ColorChangeCol)
//Piece removed from game
                                                        {
                                                                    WhiteRookArray[i][0] = -1;//Outside board
                                                                    WhiteRookArray[i][1] = -1;//Outside board
                                                                    WhiteRookArray[i][2] = 0;//Does not exist

                                                        }
                                                        else if (BlackRookArray[i][0] == ColorChangeRow && BlackRookArray[i][1] == ColorChangeCol)
                                                        {
                                                                    BlackRookArray[i][0] = -1;//Outside board
                                                                    BlackRookArray[i][1] = -1;//Outside board
                                                                    BlackRookArray[i][2] = 0;//Does not exist

                                                        }

                                                        if (take == 0)
                                                        {
                                                                    if (WhiteRookArray[i][0] == MissingRow && WhiteRookArray[i][1] == MissingCol)
//Piece moving to a nontaken square
                                                                    {
                                                                                WhiteRookArray[i][0] = NewRow;//New square
                                                                                WhiteRookArray[i][1] = NewCol;//
                                                                    }
                                                                    else if (BlackRookArray[i][0] == MissingRow && BlackRookArray[i][1] == MissingCol)
                                                                    {
                                                                                BlackRookArray[i][0] = NewRow;//New square
                                                                                BlackRookArray[i][1] = NewCol;//
                                                                    }
                                                        }
                                                        else
                                                        {
                                                                    if (WhiteRookArray[i][0] == MissingRow && WhiteRookArray[i][1] == MissingCol)
//Piece moving to a taken square
                                                                    {
                                                                                WhiteRookArray[i][0] = ColorChangeRow;//Outside board
                                                                                WhiteRookArray[i][1] = ColorChangeCol;//Outside board

                                                                    }
                                                                    else if (BlackRookArray[i][0] == MissingRow && BlackRookArray[i][1] == MissingCol)
                                                                    {
                                                                                BlackRookArray[i][0] = ColorChangeRow;//Outside board
                                                                                BlackRookArray[i][1] = ColorChangeCol;//Outside board

                                                                    }
                                                        }
                                            }
                                            //------------------------------------------------------------------------

                                            //General form for change in piece location-----------------------------Queen
                                            for(int i=0;i<1;i++)
                                            {
                                                        if (WhiteQueenArray[i][0] == ColorChangeRow && WhiteQueenArray[i][1] == ColorChangeCol)
//Piece removed from game
                                                        {
                                                                    WhiteQueenArray[i][0] = -1;//Outside board
                                                                    WhiteQueenArray[i][1] = -1;//Outside board
                                                                    WhiteQueenArray[i][2] = 0;//Does not exist

                                                        }
                                                        else if (BlackQueenArray[i][0] == ColorChangeRow && BlackQueenArray[i][1] == ColorChangeCol)
                                                        {
                                                                    BlackQueenArray[i][0] = -1;//Outside board
                                                                    BlackQueenArray[i][1] = -1;//Outside board
                                                                    BlackQueenArray[i][2] = 0;//Does not exist

                                                        }

                                                        if (take == 0)
                                                        {
                                                                    if (WhiteQueenArray[i][0] == MissingRow && WhiteQueenArray[i][1] == MissingCol)
//Piece moving to a nontaken square
                                                                    {
                                                                                WhiteQueenArray[i][0] = NewRow;//New square
                                                                                WhiteQueenArray[i][1] = NewCol;//
                                                                    }
                                                                    else if (BlackQueenArray[i][0] == MissingRow && BlackQueenArray[i][1] == MissingCol)
                                                                    {
                                                                                BlackQueenArray[i][0] = NewRow;//New square
                                                                                BlackQueenArray[i][1] = NewCol;//
                                                                    }
                                                        }
                                                        else
                                                        {
                                                                    if (WhiteQueenArray[i][0] == MissingRow && WhiteQueenArray[i][1] == MissingCol)
//Piece moving to a taken square
                                                                    {
                                                                                WhiteQueenArray[i][0] = ColorChangeRow;//Outside board
                                                                                WhiteQueenArray[i][1] = ColorChangeCol;//Outside board

                                                                    }
                                                                    else if (BlackQueenArray[i][0] == MissingRow && BlackQueenArray[i][1] == MissingCol)
```

```
                                                {
                                                            BlackQueenArray[i][0] = ColorChangeRow;//Outside board
                                                            BlackQueenArray[i][1] = ColorChangeCol;//Outside board

                                                }
                                    }
                        }
                        //-------------------------------------------------------------------------

                        //General form for change in piece location-----------------------------King
                        for(int i=0;i<1;i++)
                        {
                                    if (WhiteKingArray[i][0] == ColorChangeRow && WhiteKingArray[i][1] == ColorChangeCol)
//Piece removed from game
                                    {
                                                WhiteKingArray[i][0] = -1;//Outside board
                                                WhiteKingArray[i][1] = -1;//Outside board
                                                WhiteKingArray[i][2] = 0;//Does not exist

                                    }
                                    else if (BlackKingArray[i][0] == ColorChangeRow && BlackKingArray[i][1] == ColorChangeCol)
                                    {
                                                BlackKingArray[i][0] = -1;//Outside board
                                                BlackKingArray[i][1] = -1;//Outside board
                                                BlackKingArray[i][2] = 0;//Does not exist

                                    }

                                    if (take == 0)
                                    {
                                                if (WhiteKingArray[i][0] == MissingRow && WhiteKingArray[i][1] == MissingCol)
//Piece moving to a nontaken square
                                                {
                                                            WhiteKingArray[i][0] = NewRow;//New square
                                                            WhiteKingArray[i][1] = NewCol;//
                                                }
                                                else if (BlackKingArray[i][0] == MissingRow && BlackKingArray[i][1] == MissingCol)
                                                {
                                                            BlackKingArray[i][0] = NewRow;//New square
                                                            BlackKingArray[i][1] = NewCol;//
                                                }
                                    }
                                    else
                                    {
                                                if (WhiteKingArray[i][0] == MissingRow && WhiteKingArray[i][1] == MissingCol)
//Piece moving to a taken square
                                                {
                                                            WhiteKingArray[i][0] = ColorChangeRow;//Outside board
                                                            WhiteKingArray[i][1] = ColorChangeCol;//Outside board

                                                }
                                                else if (BlackKingArray[i][0] == MissingRow && BlackKingArray[i][1] == MissingCol)
                                                {
                                                            BlackKingArray[i][0] = ColorChangeRow;//Outside board
                                                            BlackKingArray[i][1] = ColorChangeCol;//Outside board

                                                }
                                    }
                        }
                        //-------------------------------------------------------------------------

                        //Changes accounted for. Will now keep track of all pieces


            for(int i=0;i<8;i++)//After calculating and implementing, change board
            {
                        for(int j=0;j<8;j++)
                        {
                                    OldBoard[i][j] = NewBoard[i][j];
                        }
            }
}
```

Movement.h

```
//Sean Vibbert
//Chess Playing Robot
//Movement.h
//Last Modified: 4/17/19
//This header file contains the algorithms for moving pieces
#include "stm32f446.h"
#include "PC6PWM.h"
#include "PC1234StepperMotors.h"
#include "PB1_EMAGNET.h"
#include "PB2to9Keypad.h"
#define OneSquareYPulses 190//Y Steps for a single square
#define OneSquareXPulses 190//X Steps for a single square
#define BoardDistanceY 0//Distance to the Board
#define BoardDistanceX 800//Distance to the Board
#define KingHeight 0.44
#define QueenHeight 0.62
#define RookHeight 0.94
#define BishopHeight 0.74
#define KnightHeight 0.81
#define PawnHeight 0.93
```

```
int GlobalYLoc = 0;//Assume starting at 0
int GlobalXLoc = 0;
int GlobalPlayerTurn = 1;//1 = white's turn, -1 = black's turn
int ActuateRetractEmpty = 20;
int ActuateRetractHolding = 30;
int ActuateExtend = 28;
void YMove(double squares, int record);
void XMove(double squares, int record);
void ToBoardY(int offset);
void ToBoardX(int offset);
void ExtendActuator(unsigned int PieceType);
void Return(void);
void TestSequence(void);

void LogicWait(int steps);
void MovePiece(int row1, int col1, int piecetype, int row2, col2, int capture, int capturepiece);
void TakePiece(int row, int col, int endrow, int endcol, int piecetype, int color);
void GoClock(int rowshift, int colshift);
void Ymove(double squares, int record)
{
           GoStep3((int)(OneSquareYPulses*squares));
           if (record == 1)
           GlobalYLoc += (int)(OneSquareYPulses*squares);
           Wait(100000);
}
void Xmove(double squares, int record)
{
           GoStep4((int)(OneSquareXPulses*squares));
           if (record == 1)
           GlobalXLoc += (int)(OneSquareYPulses*squares);
           Wait(100000);
}
void ToBoardY(int offset)
{
           GoStep3((int)BoardDistanceY + offset);
           GlobalYLoc += (int)BoardDistanceY + offset;
           Wait(100000);
}
void ToBoardX(int offset)
{
           GoStep4((int)BoardDistanceX + offset);
           GlobalXLoc += (int)BoardDistanceX + offset;
           Wait(100000);
}
void ExtendActuator(unsigned int PieceType)
{
           for(int i=4;i>0;i--)
           {
           switch(PieceType)
           {

   case 0:
      PC6_PWM_SETDUTY((unsigned int)(PawnHeight*(10000/i)));
                                 LogicWait(1);
      break;

            case 1:
                                 PC6_PWM_SETDUTY((unsigned int)(KnightHeight*10000/i));
                                 LogicWait(1);
                                 break;

            case 2:
                                 PC6_PWM_SETDUTY((unsigned int)(BishopHeight*10000/i));
                                 LogicWait(1);
                                 break;

            case 3:
                                 PC6_PWM_SETDUTY((unsigned int)(RookHeight*10000/i));
                                 LogicWait(1);
                                 break;

            case 4:
                                 PC6_PWM_SETDUTY((unsigned int)(QueenHeight*10000/i));
                                 LogicWait(1);
                                 break;

            case 5:
                                 PC6_PWM_SETDUTY((unsigned int)(KingHeight*10000/i));
                                 LogicWait(1);
                                 break;
            case 6:
                                 PC6_PWM_SETDUTY((unsigned int)(3800/i));
                                            LogicWait(1);
                                 break;
            case 7:
                                 PC6_PWM_SETDUTY((unsigned int)(3800/i));
                                 LogicWait(1);
                                 break;
                      case 8:
                                 PC6_PWM_SETDUTY((unsigned int)(10000));
                                 break;
                      case 9:
                                 if(i==4)
                                                                 PC6_PWM_SETDUTY((unsigned int)(200));
                                 if(i==3)
                                                                 PC6_PWM_SETDUTY((unsigned int)(4000));
```

```
                              if(i==2)
                                                                PC6_PWM_SETDUTY((unsigned int)(7500));
                              if(i==1)
                                                                PC6_PWM_SETDUTY((unsigned int)(10000));
                              break;

   default:
                              break;
          }
          }
}

void Return()
{
          GoStep3((-1)*GlobalYLoc);
          GoStep4((-1)*GlobalXLoc);
          GlobalYLoc = 0;
          GlobalXLoc = 0;
}
void TestSequence()
{
          AllOn();
          Wait(1000000);
          int i,j,dir = 1;
          ToBoardX(0);
          ToBoardY(0);
          for(j=0;j<8;j++)
          {
                    for(i=0;i<7;i++)
            {
                    Xmove(1*dir,1);
            }
                    dir = dir*(-1);
                    Ymove(1,1);
  }
          Wait(1000000);
          Return();
          AllOff();
}
void MovePiece(int row1, int col1, int piecetype, int row2, int col2, int capture, int capturepiece)
{
          AllOn();
          ExtendActuator(7);//7 is  partial
          LogicWait(3);
          ToBoardX(0);//Go to board
          ToBoardY(0);
          if (capture == 1)
                    TakePiece(row2,col2,row1,col1,capturepiece,GlobalPlayerTurn*(-1));
          else
          {
                    Xmove(row1,1);//Move to square if no capture. Capture function will end on that square.
                    Ymove(col1,1);
          }
          AllOff();
          LogicWait(1);//This will fix it
          ExtendActuator(piecetype);//Extend to piece height
          LogicWait(ActuateExtend);
          MagOn();//Magnet On
          LogicWait(1);
          ExtendActuator(6);//6 is retract full
          AllOn();
          LogicWait(ActuateRetractHolding);
          Xmove(0.5,0);//Shift to offset square
          Ymove(0.5,0);
          Xmove(row2-row1,1);//Move to second square
          Ymove(col2-col1,1);
          Xmove(-0.5,0);//Shift to offset square
          Ymove(-0.5,0);
          AllOff();
          ExtendActuator(piecetype);//Extend to piece height
          LogicWait(ActuateExtend);
          MagOff();//Magnet Off
          LogicWait(1);
          ExtendActuator(8);//Extend to place
          LogicWait(2);
          MagOff();//Magnet Off
          ExtendActuator(7);//6 is retract
          AllOn();
          LogicWait(ActuateRetractEmpty);

          GoClock(row2,col2);

          Return();
          AllOff();
}
void TakePiece(int row, int col, int endrow, int endcol, int piecetype, int color)
{
                    Xmove(row,1);//Go to piece that will get captured
                    Ymove(col,1);
                    AllOff();
                    LogicWait(1);//This will fix it maybe
                    ExtendActuator(piecetype);//Extend to piece height
                    LogicWait(ActuateExtend);
                    MagOn();
                    LogicWait(1);
            ExtendActuator(6);
```

```
                        AllOn();
                        LogicWait(ActuateRetractHolding);
              //For now I will dispose of piece in the same spot
                        int TrashDumpX = -2;//2 squares right of board
              int TrashDumpY = 4;
                        Xmove(0.5,0);//Shift to offset square
                        Ymove(0.5,0);
              Xmove(TrashDumpX - row,1);//Go to piece that will get captured
                        Ymove(TrashDumpY - col,1);
                        Xmove(-0.5,0);//Shift to offset square
                        Ymove(-0.5,0);
                        AllOff();
              ExtendActuator(piecetype);
              LogicWait(ActuateExtend);
                        MagOff();
                        LogicWait(1);
                        ExtendActuator(8);//Extend to place
                        LogicWait(2);
                        MagOff();
                        AllOn();
                        ExtendActuator(7);
                        LogicWait(ActuateRetractEmpty);

                        Xmove(endrow-row + (row-TrashDumpX),1);
                        Ymove(endcol-col + (col-TrashDumpY),1);
}

void GoClock(int rowshift, int colshift)
{
           double clocklocx = -1;
           double clocklocy = 2.6;
                      Xmove(clocklocx-rowshift,1);//Go to piece that will get captured
                      Ymove(clocklocy-colshift,1);
           LogicWait(1);//This will fix it
           ExtendActuator(9);//Extend to piece height
           LogicWait(ActuateExtend);
           LogicWait(1);
           ExtendActuator(7);//7 is retract partial
}
void LogicWait(int steps)
{
           int i;
                      for(i=0;i<steps;i++)//wait
                      Wait(1000000);
}
```

PB1_EMAGNET.h

```
//Sean Vibbert
//Chess Playing Robot
//PB1_EMAGNET.h
//Last Modified: 4/17/19
//This header file is simply an on/off for the electromagnet
#include "stm32f446.h"
void MagnetEnable(void);
void MagOn(void);
void MagOff(void);
void MagnetEnable()
{
           RCC_AHB1ENR |= 2; //Bit 1 is GPIOB clock enable bit
           GPIOB_MODER |= (1<<1*2);//Output on PB1
           GPIOB_OTYPER |= (1<<1);//Set to open drain for DC averaging
}
void MagOn()
{
           GPIOB_ODR |= 1<<1;
}
void MagOff()
{
           GPIOB_ODR &= ~(1<<1);
}
```

PB2to9Keypad.h

```
//Sean Vibbert
//Chess Playing Robot
//PB2to9Keypad.h
//Last Modified: 4/17/19
//This header file is used to communicate with the keypad
#include "stm32f446.h"
void LogicWait2(int steps);
void Wait2(unsigned long k);
void KeypadEnable(void);
int GetKeypad(void);
int KeyPad[16];
int NoPress = 0;
void KeypadEnable()
{
           GPIOB_MODER &= ~(3<<2*0);
           GPIOB_MODER &= ~(3<<2*2);
           GPIOB_MODER &= ~(3<<3*2);
           GPIOB_MODER &= ~(3<<4*2);
           GPIOB_MODER &= ~(3<<5*2);
           GPIOB_MODER &= ~(3<<6*2);
           GPIOB_MODER &= ~(3<<7*2);
           GPIOB_MODER &= ~(3<<8*2);
```

```
        GPIOB_MODER &= ~(3<<9*2);
        GPIOB_MODER |= (1<<2*2);//Output on PB2 = row1
        GPIOB_MODER |= (1<<3*2);//Output on PB3 = row2
        GPIOB_MODER |= (1<<4*2);//Output on PB4 = row3
        GPIOB_MODER |= (1<<5*2);//Output on PB5 = row4
        int i;
        for(i=0;i<16;i++)
                KeyPad[i] = 0;
        //PB6-9 input by default
        //6 = col1
        //7 = col2
        //8 = col3
        //9 = col4
        //PB0 is also input for the clock button
}
int GetKeypad(void)
{
        LogicWait2(1);
        //upper left is 1, lower right is 16
        GPIOB_ODR |= (1<<2);//first row
        if ((GPIOB_IDR & (1<<6)) == (1<<6))
                return 1;
        if ((GPIOB_IDR & (1<<7)) == (1<<7))
                return 2;
        if ((GPIOB_IDR & (1<<8)) == (1<<8))
                return 3;
        if ((GPIOB_IDR & (1<<9)) == (1<<9))
                return 4;
        GPIOB_ODR &= ~(1<<2);//disable first row.
        GPIOB_ODR |= (1<<3);//second row
        if ((GPIOB_IDR & (1<<6)) == (1<<6))
                return 5;
        if ((GPIOB_IDR & (1<<7)) == (1<<7))
                return 6;
        if ((GPIOB_IDR & (1<<8)) == (1<<8))
                return 7;
        if ((GPIOB_IDR & (1<<9)) == (1<<9))
                return 8;
        GPIOB_ODR &= ~(1<<3);//disable second row.
        GPIOB_ODR |= (1<<4);//third row
        if ((GPIOB_IDR & (1<<6)) == (1<<6))
                return 9;
        if ((GPIOB_IDR & (1<<7)) == (1<<7))
                return 10;
        if ((GPIOB_IDR & (1<<8)) == (1<<8))
                return 11;
        if ((GPIOB_IDR & (1<<9)) == (1<<9) || (GPIOB_IDR & (1<<0)) == (0))//Also let clock button count as this press
                return 12;
        GPIOB_ODR &= ~(1<<4);//disable third row.
        GPIOB_ODR |= (1<<5);//fourth row
        if ((GPIOB_IDR & (1<<6)) == (1<<6))
                return 13;
        if ((GPIOB_IDR & (1<<7)) == (1<<7))
                return 14;
        if ((GPIOB_IDR & (1<<8)) == (1<<8))
                return 15;
        if ((GPIOB_IDR & (1<<9)) == (1<<9))
                return 16;
        GPIOB_ODR &= ~(1<<5);//disable fourth row.
                return 0;
        }

void LogicWait2(int steps)
{
        int i;
        for(i=0;i<steps;i++)//wait
                Wait2(1000000);
}
void Wait2(unsigned long k)
{
        unsigned long i;
        for (i=0;i<k;i++)
        {
                ;//Do nothing
        }
}
```

PB10to15Communication.h

```
//Sean Vibbert
//Chess Playing Robot
//PB10to15Communication.h
//Last Modified: 4/17/19
//This header file is used to communicate with the Raspberry PI 3
#include "stm32f446.h"
unsigned int CommBit = 0;
void CommunicateSetup(void);//PB11 is missing from the pinout of the board. Likely used.
unsigned long Get32Bits(void);
void Send32Bits(unsigned long OutData);
void CommWait(int type);
void CommunicateSetup()
{
                GPIOB_MODER &= ~(3<<12*2);//Clear bit
                GPIOB_MODER &= ~(3<<13*2);
                GPIOB_MODER &= ~(3<<14*2);
                GPIOB_MODER &= ~(3<<15*2);
```

```
                      GPIOB_MODER |= (1<<12*2);        //12 is output data
                      GPIOB_MODER |= (1<<13*2);        //13 is output clocking
                      //14 is input data, already cleared
             //15 is input clocking, already cleared
}
void Send32Bits(unsigned long OutData)//OutData will be a 32 bit long
{
           GPIOB_ODR &= ~(1<<12);//Start of signal 00 then 11
           GPIOB_ODR &= ~(1<<13);
           CommWait(1);
           CommWait(1);
           CommWait(1);
           GPIOB_ODR |= (1<<12);
           GPIOB_ODR |= (1<<13);

           CommBit = 1;
           CommWait(1);
           CommWait(1);
           CommWait(1);
           for(int i=0;i<32;i++)
           {
                      GPIOB_ODR &= ~(1<<12);//Clear
                      GPIOB_ODR |= ((OutData & 1)<<12);//Set to first bit of OutData
                      OutData = OutData - (OutData&1);//Clear first bit
                      OutData = OutData >> 1;//Right shift 1
                      CommWait(0);
                      if (CommBit == 1)
                      {
                                 GPIOB_ODR &= ~(1<<13);
                                 CommBit = 0;
                      }
                      else
                      {
                                 GPIOB_ODR |= (1<<13);
                                 CommBit = 1;
                      }
                      CommWait(0);
                      CommWait(1);
           }
}
unsigned long Get32Bits()//OutData will be a 32 bit long
{
           unsigned int LastClock = GPIOB_IDR & (1<<15);
           unsigned int ThisClock = GPIOB_IDR & (1<<15);
           unsigned int StartCheck1 = GPIOB_IDR & (1<<14);
           unsigned int StartCheck2 = GPIOB_IDR & (1<<15);
           unsigned long Data = 0;
           unsigned long tmp;
           while((StartCheck1<<1) == 1<<15 || StartCheck2 == 1<<15)//Wait for both equal signal
           {
                      StartCheck1 = GPIOB_IDR & (1<<14);
                      StartCheck2 = GPIOB_IDR & (1<<15);
           }

           while((StartCheck1<<1) != 1<<15 || StartCheck2 != 1<<15)//Wait for both equal signal
           {
                      StartCheck1 = GPIOB_IDR & (1<<14);
                      StartCheck2 = GPIOB_IDR & (1<<15);
           }
           //unsigned int InBit
           for(int i=0;i<32;i++)
           {
                      while(ThisClock == LastClock)
                      {
                                 ThisClock = GPIOB_IDR & (1<<15);
                                 CommWait(1);
                                 LastClock = GPIOB_IDR & (1<<15);
                      }
                      tmp = (GPIOB_IDR & (1<<14)) >> 14;//Get bit into first position
                      tmp = tmp << i;
                      Data |= (tmp);
           LastClock = GPIOB_IDR & (1<<15);
           ThisClock = GPIOB_IDR & (1<<15);
           }

           return Data;
}
void CommWait(int type)
{
           if (type == 0)
           {
                      for(int i = 0;i<5000;i++)
                      {
                                 ;
                      }
           }
           else
           {
                      for(int i = 0;i<50000;i++)
                      {
                                 ;
                      }
           }
}
```

PC6PWM.h

```
//Sean Vibbert
//Chess Playing Robot
//PC6PWM.h
//Last Modified: 4/17/19
//This header file is used to control the voltage of the linear actuator
#include "stm32f446.h"
void PC6_PWM_INIT(void);
void PC6_PWM_SETDUTY(unsigned int duty);
void PC6_PWM_INIT()
{
//PWM output on PC6. 12 bit resolution
RCC_AHB1ENR |= 4; //Bit 2 is GPIOC clock enable bit
RCC_APB1ENR |= 2; //Enable peripheral timer for timer 3 (bit 1)
//I/O bits
GPIOC_MODER |= 0x2000; //Bits 13-12 = 10 for Alt Funct Mode on PC6
GPIOC_OTYPER |= (1<<6);//Set to open drain for DC averaging
GPIOC_OSPEEDER |= 0x3000; //Bits 13-12 = 11 for high speed on PC6
//PUPDR defaults to no pull up no pull down
//Timer 3 bits
GPIOC_AFRL = 0x02000000; //Sets PC6 to Timer 3
TIM3_CCMR1 |= 0x60; //Timer 3 in PWM Mode bits 6,5,4 = 110
TIM3_CCMR1 |= 0x0C; //Timer 3 Preload enable and fast enable
TIM3_CR1 |= (1 << 7); //Auto reload is buffered
TIM3_PSC = 0;
TIM3_ARR = 10000; //16 MHz/(40000*8) = 50 Hz = 20ms
TIM3_CCR1 = 0; //Duty cycle starts at 0
TIM3_CCER |= 1; //Compare and capture output enable
TIM3_EGR |= 1; //Enable event
TIM3_CR1 |= 1; //Enable Timer 3
}
void PC6_PWM_SETDUTY(unsigned int duty)
{
TIM3_CCR1 = duty;//0 to TIM3_ARR
}
```

PC1234StepperMotors.h

```
//Sean Vibbert
//Chess Playing Robot
//PC1234StepperMotors.h
//Last Modified: 4/17/19
//This header file is used to control stepper motors. More port pins than 1234 are used
//GPIOC
//11 is pulse bit for driver 1
//2 is the dir bit for driver 1
//3 is the pulse bit for driver 2
//4 is the dir bit for driver 2
//7 is the pulse bit for driver 3
//8 is the dir bit for driver 3
//10 is enable or disable stepper motors
#include "stm32f446.h"
void StepperInit(void);
void GoStep1(int steps);
void GoStep2(int steps);
void GoStep3(int steps);
void GoStep4(int steps);
void AllOff(void);
void AllOn(void);
void AllOff()
{
          GPIOC_ODR |= 1<<10;
}
void AllOn()
{
          GPIOC_ODR &= ~(1<<10);
}
void StepperInit()//Initialize all bits to output
{
          GPIOC_MODER |= (1<<11*2);
          GPIOC_MODER |= (1<<2*2);
          GPIOC_MODER |= (1<<3*2);
          GPIOC_MODER |= (1<<4*2);
          GPIOC_MODER |= (1<<7*2);
          GPIOC_MODER |= (1<<8*2);
          GPIOC_MODER |= (1<<10*2);
}
void GoStep1(int steps)//Driver 1 alone
{
          if (steps > 0)
                    GPIOC_ODR |= (1<<2);
          else
          {
                    GPIOC_ODR &= ~(1<<2);
                    steps = steps*(-1);
          }
          while(steps > 0)
          {
                    GPIOC_ODR |= (1<<11);
                    Wait(10000);
                    GPIOC_ODR &= ~(1<<11);
                    Wait(10000);
                    steps--;
          }
}
```

```
void GoStep2(int steps)//Driver 2 alone
{
          if (steps > 0)
                    GPIOC_ODR |= (1<<4);
          else
          {
                    GPIOC_ODR &= ~(1<<4);
                    steps = steps*(-1);
          }
          while(steps > 0)
          {
                    GPIOC_ODR |= (1<<3);
                    Wait(10000);
                    GPIOC_ODR &= ~(1<<3);
                    Wait(10000);
                    steps--;
          }
}
void GoStep3(int steps)//Driver 1 and 2
{
          if (steps > 0)
          {
                    GPIOC_ODR |= (1<<4);
                    GPIOC_ODR |= (1<<2);
          }
          else
          {
                    GPIOC_ODR &= ~(1<<4);
                    GPIOC_ODR &= ~(1<<2);
                    steps = steps*(-1);
          }
          while(steps > 0)
          {
                    GPIOC_ODR |= (1<<11);
                    GPIOC_ODR |= (1<<3);
                    Wait(10000);
                    GPIOC_ODR &= ~(1<<11);
                    GPIOC_ODR &= ~(1<<3);
                    Wait(10000);
                    steps--;
          }
}
void GoStep4(int steps)//Driver 3 alone
{
          if (steps > 0)
                    GPIOC_ODR |= (1<<8);
          else
          {
                    GPIOC_ODR &= ~(1<<8);
                    steps = steps*(-1);
          }
          while(steps > 0)
          {
                    GPIOC_ODR |= (1<<7);
                    Wait(10000);
                    GPIOC_ODR &= ~(1<<7);
                    Wait(10000);
                    steps--;
          }
}
```

stm32f446.h

```
//This header file was written by Dr. Dick Blandford and has been used to properly address the stm32f446 port pins for some time
//stm32f446.h
//Updated: July 1, 2017
//D to A Addresses
#define DAC_CR (*((volatile unsigned long *) 0x40007400))   //DAC Control Register
#define DAC_SWTRIGR (*((volatile unsigned long *) 0x40007404))   //DAC Software trigger Reg
#define DAC_DHR12R1 (*((volatile unsigned long *) 0x40007408))   //DAC 12-bit Right align data Reg
#define DAC_DHR12R2 (*((volatile unsigned long *) 0x40007414))   //DAC2 12-bit Right align data Reg
#define DAC_DOR1 (*((volatile unsigned long *) 0x4000742C))   //DAC Ch1 data output Reg
#define DAC_DOR2 (*((volatile unsigned long *) 0x40007430))   //DAC Ch2 data output Reg
//GPIO Port A Addresses
#define GPIOA_MODER    (*((volatile unsigned long *) 0x40020000))   //GPIO A Mode Reg
#define GPIOA_PUPDR    (*((volatile unsigned long *) 0x4002000C))   //GPIO A Pull Up/Pull Dn Reg
#define GPIOA_OSPEEDER (*((volatile unsigned long *) 0x40020008))   //GPIO A Speed register
#define GPIOA_OTYPER   (*((volatile unsigned long *) 0x40020004))   //GPIO A Output type register
#define GPIOA_IDR      (*((volatile unsigned long *) 0x40020010))   //GPIO A Input Data register
#define GPIOA_ODR      (*((volatile unsigned long *) 0x40020014))   //GPIO A Output Data register
#define GPIOA_BSRR     (*((volatile unsigned long *) 0x40020018))   //GPIO A Output Bit set/reset reg
#define GPIOA_AFRL     (*((volatile unsigned long *) 0x40020020))   //GPIO A Alt Funct reg bits 0-7
#define GPIOA_AFRH     (*((volatile unsigned long *) 0x40020024))   //GPIO A Alt Funct reg bits 8-15
//GPIO Port B Addresses
#define GPIOB_MODER    (*((volatile unsigned long *) 0x40020400))   //GPIO B Mode Reg
#define GPIOB_PUPDR    (*((volatile unsigned long *) 0x4002040C))   //GPIO B Pull Up/Pull Dn Reg
#define GPIOB_OSPEEDER (*((volatile unsigned long *) 0x40020408))   //GPIO B Speed register
#define GPIOB_OTYPER   (*((volatile unsigned long *) 0x40020404))   //GPIO B Output type register
#define GPIOB_IDR      (*((volatile unsigned long *) 0x40020410))   //GPIO B Input Data register
#define GPIOB_ODR      (*((volatile unsigned long *) 0x40020414))   //GPIO B Output Data register
#define GPIOB_BSRR     (*((volatile unsigned long *) 0x40020418))   //GPIO B Output Bit set/reset reg
#define GPIOB_AFRL     (*((volatile unsigned long *) 0x40020420))   //GPIO B Alt Funct reg bits 0-7
#define GPIOB_AFRH     (*((volatile unsigned long *) 0x40020424))   //GPIO B Alt Funct reg bits 8-15
//GPIO Port C Addresses
#define GPIOC_MODER    (*((volatile unsigned long *) 0x40020800))   //GPIO C Mode Reg
#define GPIOC_PUPDR    (*((volatile unsigned long *) 0x4002080C))   //GPIO C Pull Up/Pull Dn Reg
```

```
#define GPIOC_OSPEEDER (*((volatile unsigned long *) 0x40020808))   //GPIO C Speed register
#define GPIOC_OTYPER   (*((volatile unsigned long *) 0x40020804))   //GPIO C Output type register
#define GPIOC_IDR      (*((volatile unsigned long *) 0x40020810))   //GPIO C Input Data register
#define GPIOC_ODR      (*((volatile unsigned long *) 0x40020814))   //GPIO C Output Data register
#define GPIOC_BSRR     (*((volatile unsigned long *) 0x40020818))   //GPIO C Output Bit set/reset reg
#define GPIOC_AFRL     (*((volatile unsigned long *) 0x40020820))   //GPIO C Alt Funct reg bits 0-7
#define GPIOC_AFRH     (*((volatile unsigned long *) 0x40020824))   //GPIO C Alt Funct reg bits 8-15
//GPIO Port D Addresses
#define GPIOD_MODER    (*((volatile unsigned long *) 0x40020C00))   //GPIO D Mode Reg
#define GPIOD_PUPDR    (*((volatile unsigned long *) 0x40020C0C))   //GPIO D Pull Up/Pull Dn Reg
#define GPIOD_OSPEEDER (*((volatile unsigned long *) 0x40020C08))   //GPIO D Speed register
#define GPIOD_OTYPER   (*((volatile unsigned long *) 0x40020C04))   //GPIO D Output type register
#define GPIOD_IDR      (*((volatile unsigned long *) 0x40020C10))   //GPIO D Input Data register
#define GPIOD_ODR      (*((volatile unsigned long *) 0x40020C14))   //GPIO D Output Data register
#define GPIOD_BSRR     (*((volatile unsigned long *) 0x40020C18))   //GPIO D Output Bit set/reset reg
#define GPIOD_AFRL     (*((volatile unsigned long *) 0x40020C20))   //GPIO D Alt Funct reg bits 0-7
#define GPIOD_AFRH     (*((volatile unsigned long *) 0x40020C24))   //GPIO D Alt Funct reg bits 8-15
//GPIO Port H Addresses
#define GPIOH_MODER    (*((volatile unsigned long *) 0x40021C00))   //GPIO H Mode Reg
#define GPIOH_PUPDR    (*((volatile unsigned long *) 0x40021C0C))   //GPIO H Pull Up/Pull Dn Reg
#define GPIOH_OSPEEDER (*((volatile unsigned long *) 0x40021C08))   //GPIO H Speed register
#define GPIOH_OTYPER   (*((volatile unsigned long *) 0x40021C04))   //GPIO H Output type register
#define GPIOH_IDR      (*((volatile unsigned long *) 0x40021C10))   //GPIO H Input Data register
#define GPIOH_ODR      (*((volatile unsigned long *) 0x40021C14))   //GPIO H Output Data register
#define GPIOH_BSRR     (*((volatile unsigned long *) 0x40021C18))   //GPIO H Output Bit set/reset reg
#define GPIOH_AFRL     (*((volatile unsigned long *) 0x40021C20))   //GPIO H Alt Funct reg bits 0-7
#define GPIOH_AFRH     (*((volatile unsigned long *) 0x40021C24))   //GPIO H Alt Funct reg bits 8-15
//Clock addresses
#define RCC_CR      (*((volatile unsigned long *) 0x40023800))   //Clock Control Register
#define RCC_PLLCFGR (*((volatile unsigned long *) 0x40023804))   //PLL Config Register
#define RCC_CFGR    (*((volatile unsigned long *) 0x40023808))   //Clock Config Register
#define RCC_CIR     (*((volatile unsigned long *) 0x4002380C))   //Clock Interrupt Register

#define RCC_APB1RSTR (*((volatile unsigned long *) 0x40023820))   //Periph Reset Register
#define RCC_APB1ENR (*((volatile unsigned long *) 0x40023840))   //DAC Periph Clock Enable Reg
#define RCC_AHB1ENR (*((volatile unsigned long *) 0x40023830))   //GPIO Enable Reg
#define RCC_APB2ENR (*((volatile unsigned long *) 0x40023844))   //ADC Periph Clock Enable Reg

//Flash Memory
#define FLASH_ACR (*((volatile unsigned long *) 0x40023C00))   //Flash Access Control Register

//A to D Addresses
//ADC1
#define ADC1_SR (*((volatile unsigned long *) 0x40012000))      //ADC1 Status Register
#define ADC1_CR1 (*((volatile unsigned long *) 0x40012004))     //ADC1 Control Register 1
#define ADC1_CR2 (*((volatile unsigned long *) 0x40012008))     //ADC1 Control Register 2
#define ADC1_SQR1 (*((volatile unsigned long *) 0x4001202C))     //ADC1 Regular Seq Register 1
#define ADC1_SQR2 (*((volatile unsigned long *) 0x40012030))     //ADC1 Regular Seq Register 2
#define ADC1_SQR3 (*((volatile unsigned long *) 0x40012034))     //ADC1 Regular Seq Register 3
#define ADC1_DR (*((volatile unsigned long *) 0x4001204C))      //ADC1 Data Register
//ADC2
#define ADC2_SR (*((volatile unsigned long *) 0x40012100))      //ADC2 Status Register
#define ADC2_CR1 (*((volatile unsigned long *) 0x40012104))     //ADC2 Control Register 1
#define ADC2_CR2 (*((volatile unsigned long *) 0x40012108))     //ADC2 Control Register 2
#define ADC2_SQR1 (*((volatile unsigned long *) 0x4001212C))     //ADC2 Regular Seq Register 1
#define ADC2_SQR2 (*((volatile unsigned long *) 0x40012130))     //ADC2 Regular Seq Register 2
#define ADC2_SQR3 (*((volatile unsigned long *) 0x40012134))     //ADC2 Regular Seq Register 3
#define ADC2_DR (*((volatile unsigned long *) 0x4001214C))      //ADC2 Data Register
//ADC3
#define ADC3_SR (*((volatile unsigned long *) 0x40012200))      //ADC3 Status Register
#define ADC3_CR1 (*((volatile unsigned long *) 0x40012204))     //ADC3 Control Register 1
#define ADC3_CR2 (*((volatile unsigned long *) 0x40012208))     //ADC3 Control Register 2
#define ADC3_SQR1 (*((volatile unsigned long *) 0x4001222C))     //ADC3 Regular Seq Register 1
#define ADC3_SQR2 (*((volatile unsigned long *) 0x40012230))     //ADC3 Regular Seq Register 2
#define ADC3_SQR3 (*((volatile unsigned long *) 0x40012234))     //ADC3 Regular Seq Register 3
#define ADC3_DR (*((volatile unsigned long *) 0x4001224C))      //ADC3 Data Register
//ADC Common
#define ADC_CCR (*((volatile unsigned long *) 0x40012304))     //ADC Common Control Register
//Timer 2 addresses
#define TIM2_CR1  (*((volatile unsigned long *) 0x40000000))   //Timer 2 Control Reg 1
#define TIM2_CR2  (*((volatile unsigned long *) 0x40000004))   //Timer 2 Control Reg 2
#define TIM2_SR   (*((volatile unsigned long *) 0x40000010))   //Timer 2 Status Reg
#define TIM2_DIER (*((volatile unsigned long *) 0x4000000C))   //Timer 2 Interrupt Enable
#define TIM2_EGR  (*((volatile unsigned long *) 0x40000014))   //Timer 2 Event Generation Reg
#define TIM2_CNT  (*((volatile unsigned long *) 0x40000024))   //Timer 2 Count Reg
#define TIM2_PSC  (*((volatile unsigned long *) 0x40000028))   //Timer 2 Prescale Reg
#define TIM2_ARR  (*((volatile unsigned long *) 0x4000002C))   //Timer 2 Auto-Reload Reg
//Timer 3 addresses
#define TIM3_CR1  (*((volatile unsigned long *) 0x40000400))   //Timer 3 Control Reg 1
#define TIM3_CR2  (*((volatile unsigned long *) 0x40000404))   //Timer 3 Control Reg 2
#define TIM3_SR   (*((volatile unsigned long *) 0x40000410))   //Timer 3 Status Reg
#define TIM3_DIER (*((volatile unsigned long *) 0x4000040C))   //Timer 3 Interrupt Enable
#define TIM3_EGR  (*((volatile unsigned long *) 0x40000414))   //Timer 3 Event Generation Reg
#define TIM3_CCMR1 (*((volatile unsigned long *) 0x40000418))   //Timer 3 Capture/Compare Mode
#define TIM3_CCER (*((volatile unsigned long *) 0x40000420))   //Timer 3 Compare/Capture Reg
#define TIM3_CNT  (*((volatile unsigned long *) 0x40000424))   //Timer 3 Count Reg
#define TIM3_PSC  (*((volatile unsigned long *) 0x40000428))   //Timer 3 Prescale Reg
#define TIM3_ARR  (*((volatile unsigned long *) 0x4000042C))   //Timer 3 Auto-Reload Reg
#define TIM3_CCR1 (*((volatile unsigned long *) 0x40000434))   //Timer 3 Capture/Compare Reg
//Timer 6 addresses
#define TIM6_CR1 (*((volatile unsigned long *) 0x40001000))   //Timer 6 Control Reg 1
#define TIM6_CR2 (*((volatile unsigned long *) 0x40001004))   //Timer 6 Control Reg 2
#define TIM6_SR (*((volatile unsigned long *) 0x40001010))    //Timer 6 Status Reg
#define TIM6_DIER (*((volatile unsigned long *) 0x4000100C))  //Timer 6 Interrupt Enable
#define TIM6_CNT (*((volatile unsigned long *) 0x40001024))   //Timer 6 Count Reg
#define TIM6_PSC (*((volatile unsigned long *) 0x40001028))   //Timer 6 Prescale Reg
```

```
#define TIM6_ARR (*((volatile unsigned long *) 0x4000102C))   //Timer 6 Auto-Reload Reg
#define TIM6_EGR (*((volatile unsigned long *) 0x40001014))   //Timer 6 Event Generation Reg
//Interrupt controller addresses
#define NVICISER0 (*((volatile unsigned long *) 0xE000E100))   //Interrupt Enable 0-31
#define NVICISER1 (*((volatile unsigned long *) 0xE000E104))   //Interrupt Enable 32-63
#define NVICICER0 (*((volatile unsigned long *) 0xE000E180))   //Interrupt Clear Enable 0-31
#define NVICICER1 (*((volatile unsigned long *) 0xE000E184))   //Interrupt Clear Enable 32-63
//USART1 Registers
#define USART1_SR   (*((volatile unsigned long *) 0x40011000))   //USART 1 Status Register
#define USART1_DR   (*((volatile unsigned long *) 0x40011004))   //USART 1 Data Register
#define USART1_BRR  (*((volatile unsigned long *) 0x40011008))  //USART 1 Baud Rate Register
#define USART1_CR1  (*((volatile unsigned long *) 0x4001100C))  //USART 1 Control Register 1
#define USART1_CR2  (*((volatile unsigned long *) 0x40011010))  //USART 1 Control Register 2
#define USART1_CR3  (*((volatile unsigned long *) 0x40011014))  //USART 1 Control Register 3
#define USART1_GTPR (*((volatile unsigned long *) 0x40011018)) //USART 1 Gd Time & Prescale Register
//USART4 Registers
#define USART4_SR   (*((volatile unsigned long *) 0x40004C00))   //USART 4 Status Register
#define USART4_DR   (*((volatile unsigned long *) 0x40004C04))   //USART 4 Data Register
#define USART4_BRR  (*((volatile unsigned long *) 0x40004C08))  //USART 4 Baud Rate Register
#define USART4_CR1  (*((volatile unsigned long *) 0x40004C0C))  //USART 4 Control Register 1
#define USART4_CR2  (*((volatile unsigned long *) 0x40004C10))  //USART 4 Control Register 2
#define USART4_CR3  (*((volatile unsigned long *) 0x40004C14))  //USART 4 Control Register 3
#define USART4_GTPR (*((volatile unsigned long *) 0x40004C18)) //USART 4 Gd Time & Prescale Register
//USART6 Registers
#define USART6_SR   (*((volatile unsigned long *) 0x40011400))   //USART 6 Status Register
#define USART6_DR   (*((volatile unsigned long *) 0x40011404))   //USART 6 Data Register
#define USART6_BRR  (*((volatile unsigned long *) 0x40011408))  //USART 6 Baud Rate Register
#define USART6_CR1  (*((volatile unsigned long *) 0x4001140C))  //USART 6 Control Register 1
#define USART6_CR2  (*((volatile unsigned long *) 0x40011410))  //USART 6 Control Register 2
#define USART6_CR3  (*((volatile unsigned long *) 0x40011414))  //USART 6 Control Register 3
#define USART6_GTPR (*((volatile unsigned long *) 0x40011418)) //USART 6 Gd Time & Prescale Register
//I2C1
#define I2C1_CR1   (*((volatile unsigned long *) 0x40005400)) //I2C1 Control Reg 1
#define I2C1_CR2   (*((volatile unsigned long *) 0x40005404)) //I2C1 Control Reg 2
#define I2C1_OAR1  (*((volatile unsigned long *) 0x40005408)) //I2C1 Own Addr Reg 1
#define I2C1_OAR2  (*((volatile unsigned long *) 0x4000540C)) //I2C1 Own Addr Reg 2
#define I2C1_DR    (*((volatile unsigned long *) 0x40005410)) //I2C1 Data Reg
#define I2C1_SR1   (*((volatile unsigned long *) 0x40005414)) //I2C1 Status Reg 1
#define I2C1_SR2   (*((volatile unsigned long *) 0x40005418)) //I2C1 Status Reg 2
#define I2C1_CCR   (*((volatile unsigned long *) 0x4000541C)) //I2C1 Clock Control Reg
#define I2C1_TRISE (*((volatile unsigned long *) 0x40005420)) //I2C1 Rise Time Reg
#define I2C1_FLTR  (*((volatile unsigned long *) 0x40005424)) //I2C1 Filter Reg
//I2C2
#define I2C2_CR1   (*((volatile unsigned long *) 0x40005800)) //I2C2 Control Reg 1
#define I2C2_CR2   (*((volatile unsigned long *) 0x40005804)) //I2C2 Control Reg 2
#define I2C2_OAR1  (*((volatile unsigned long *) 0x40005808)) //I2C2 Own Addr Reg 1
#define I2C2_OAR2  (*((volatile unsigned long *) 0x4000580C)) //I2C2 Own Addr Reg 2
#define I2C2_DR    (*((volatile unsigned long *) 0x40005810)) //I2C2 Data Reg
#define I2C2_SR1   (*((volatile unsigned long *) 0x40005814)) //I2C2 Status Reg 1
#define I2C2_SR2   (*((volatile unsigned long *) 0x40005818)) //I2C2 Status Reg 2
#define I2C2_CCR   (*((volatile unsigned long *) 0x4000581C)) //I2C2 Clock Control Reg
#define I2C2_TRISE (*((volatile unsigned long *) 0x40005820)) //I2C2 Rise Time Reg
#define I2C2_FLTR  (*((volatile unsigned long *) 0x40005824)) //I2C2 Filter Reg
//I2C3
#define I2C3_CR1   (*((volatile unsigned long *) 0x40005C00)) //I2C3 Control Reg 1
#define I2C3_CR2   (*((volatile unsigned long *) 0x40005C04)) //I2C3 Control Reg 2
#define I2C3_OAR1  (*((volatile unsigned long *) 0x40005C08)) //I2C3 Own Addr Reg 1
#define I2C3_OAR2  (*((volatile unsigned long *) 0x40005C0C)) //I2C3 Own Addr Reg 2
#define I2C3_DR    (*((volatile unsigned long *) 0x40005C10)) //I2C3 Data Reg
#define I2C3_SR1   (*((volatile unsigned long *) 0x40005C14)) //I2C3 Status Reg 1
#define I2C3_SR2   (*((volatile unsigned long *) 0x40005C18)) //I2C3 Status Reg 2
#define I2C3_CCR   (*((volatile unsigned long *) 0x40005C1C)) //I2C3 Clock Control Reg
#define I2C3_TRISE (*((volatile unsigned long *) 0x40005C20)) //I2C3 Rise Time Reg
#define I2C3_FLTR  (*((volatile unsigned long *) 0x40005C24)) //I2C3 Filter Reg
//SPI1
#define SPI1_CR1     (*((volatile unsigned long *) 0x40013000)) //SPI1 Control Reg 1
#define SPI1_CR2     (*((volatile unsigned long *) 0x40013004)) //SPI1 Control Reg 2
#define SPI1_SR      (*((volatile unsigned long *) 0x40013008)) //SPI1 Status Reg
#define SPI1_DR      (*((volatile unsigned long *) 0x4001300C)) //SPI1 Data Reg
#define SPI1_CRCPR   (*((volatile unsigned long *) 0x40013010)) //SPI1 CRC Polynomial Reg
#define SPI1_RXCRCR  (*((volatile unsigned long *) 0x40013014)) //SPI1 CRS receive
#define SPI1_TXCRCR  (*((volatile unsigned long *) 0x40013018)) //SPI1 CRC transmit
#define SPI1_I2SCFGR (*((volatile unsigned long *) 0x4001301C)) //SPI1 I2S Config Reg
#define SPI1_I2SCPR  (*((volatile unsigned long *) 0x40013020)) //SPI1 I2S Clk Prescale
//SPI2
#define SPI2_CR1     (*((volatile unsigned long *) 0x40003800)) //SPI2 Control Reg 1
#define SPI2_CR2     (*((volatile unsigned long *) 0x40003804)) //SPI2 Control Reg 2
#define SPI2_SR      (*((volatile unsigned long *) 0x40003808)) //SPI2 Status Reg
#define SPI2_DR      (*((volatile unsigned long *) 0x4000380C)) //SPI2 Data Reg
#define SPI2_CRCPR   (*((volatile unsigned long *) 0x40003810)) //SPI2 CRC Polynomial Reg
#define SPI2_RXCRCR  (*((volatile unsigned long *) 0x40003814)) //SPI2 CRS receive
#define SPI2_TXCRCR  (*((volatile unsigned long *) 0x40003818)) //SPI2 CRC transmit
#define SPI2_I2SCFGR (*((volatile unsigned long *) 0x4000381C)) //SPI2 I2S Config Reg
#define SPI2_I2SCPR  (*((volatile unsigned long *) 0x40003820)) //SPI2 I2S Clk Prescale
//SPI3
#define SPI3_CR1     (*((volatile unsigned long *) 0x40003C00)) //SPI3 Control Reg 1
#define SPI3_CR2     (*((volatile unsigned long *) 0x40003C04)) //SPI3 Control Reg 2
#define SPI3_SR      (*((volatile unsigned long *) 0x40003C08)) //SPI3 Status Reg
#define SPI3_DR      (*((volatile unsigned long *) 0x40003C0C)) //SPI3 Data Reg
#define SPI3_CRCPR   (*((volatile unsigned long *) 0x40003C10)) //SPI3 CRC Polynomial Reg
#define SPI3_RXCRCR  (*((volatile unsigned long *) 0x40003C14)) //SPI3 CRS receive
#define SPI3_TXCRCR  (*((volatile unsigned long *) 0x40003C18)) //SPI3 CRC transmit
#define SPI3_I2SCFGR (*((volatile unsigned long *) 0x40003C1C)) //SPI3 I2S Config Reg
#define SPI3_I2SCPR  (*((volatile unsigned long *) 0x40003C20)) //SPI3 I2S Clk Prescale
```

X. Appendix C – Raspberry PI Source Code

The Raspberry PI code is compiled using g++ and requires one external header file called lwiringpi.

MAIN.c

```
//Sean Vibbert
//Chess Playing Robot
//MAIN.c (Raspberry PI)
//Last Modified: 4/17/19
//The entire Raspberry PI code, including communication, visual recognition, etc.
//Requires explicit handling when compiling
#include <wiringPi.h>
//Built-in
#include <iostream>
#include <stdlib.h>
#include <unistd.h>
#include <fstream>
#include <string>
#include <sstream>
#include <stdio.h>
using namespace std;
void CommunicateSetup(void);
void CommWait(int type);
void Send32Bits(unsigned long OutData);
unsigned long Get32Bits(void);
unsigned int CommBit = 0;

int main (int argc, char** argv)
{
  //Set up wiring.
  CommunicateSetup();
  system("gpio readall");

  while(1)
  {
  unsigned long myData = 0;

  myData = Get32Bits();

  while (myData != 0xABABABAB)
  {
  if (myData == 0xABCDABCD)//Exit program
    return 0;
  else
    {
    myData = Get32Bits();
    }
  }
  CommWait(1);
  CommWait(1);
  CommWait(1);
  CommWait(1);
  CommWait(1);

  cout << hex << myData << endl;

  system("raspistill -vf -hf -t 2000 -o ChessImage.jpg");
  usleep(300000);
  system("convert ChessImage.jpg -resize 520x400 ChessImage2.magick");
  usleep(300000);
  ifstream file("ChessImage2.magick");
  string line;
  string content;
  while(getline(file,line))
  {
    content += line;
    //content.push_back('\n');
  }
  int width = 520, height = 400;
  int RGB [520][400][3];
  int RGB2 [520][400][3];
  int shift = 0;
  int row = 0,col = 0, color = 0;
  int errors = 0;
  //Gathering data from text file
  for (int i = 83;i<content.length();i+=6)//81 = beginning after the first 0x
  {
    if (i+shift < content.length()-96)
    {
    stringstream ss;
    ss << hex << content.substr(i+shift,2);
    ss >> RGB[col][row][color];
    //if (RGB[col][row][color] > 255 || RGB[col][row][color] < 0)

    color += 1;
    if (color > 2)
    {
      color = 0;
      col += 1;
    }

    if (col > width-1)
    {
      col = 0;
      row += 1;
    }

    if (((i-83+6) % 72 ) == 0)
    {
      shift += 4;
    }

    }
  }

  //Find locations where pixel is not RGB and erase it
  for (int i=0;i<width;i++)
    for (int j = 0;j<height;j++)
      for (int k = 0;k<3;k++)
        {
          if (RGB[i][j][k] > 255 || RGB [i][j][k] < 0)
          {
            errors += 1;
            RGB[i][j][k] = 0;
          }
        }

  cout << errors << endl;

  for (int i=0;i<width;i++)//Filter out all that is not red/blue
    for (int j = 0;j<height;j++)
        {
          RGB2[i][j][0] = RGB[i][j][0];
          RGB2[i][j][1] = RGB[i][j][1];
          RGB2[i][j][1] = RGB[i][j][2];
          double r = RGB[i][j][0],g = RGB[i][j][1], b = RGB[i][j][2];

          if (r > b*0.5 && g > b*0.5)
          {
            RGB[i][j][0] = 0;
            RGB[i][j][1] = 0;
            RGB[i][j][2] = 0;
          }

          else if (g > b*0.8)
          {
            RGB[i][j][0] = 0;
            RGB[i][j][1] = 0;
            RGB[i][j][2] = 0;
          }

          if (b > r*0.5 && g > r*0.5)
          {
            RGB2[i][j][0] = 0;
            RGB2[i][j][1] = 0;
            RGB2[i][j][2] = 0;
          }

          else if (b > r*0.8)
          {
            RGB2[i][j][0] = 0;
            RGB2[i][j][1] = 0;
            RGB2[i][j][2] = 0;
          }

          if (r+b+g < 125)
          {
            RGB[i][j][0] = 0;
            RGB[i][j][1] = 0;
            RGB[i][j][2] = 0;
            RGB2[i][j][0] = 0;
            RGB2[i][j][1] = 0;
            RGB2[i][j][2] = 0;
          }

          //RGB[i][j][0] = 0;
          //RGB[i][j][1] = 0;

          //RGB2[i][j][1] = 0;
          //RGB2[i][j][2] = 0;
        }
        //520 400
        int leftshift = 70;
        int squarelength = 41;
        int lineend = 520-115;
        int topshift = 35;
        for(int line = 0;line < 8;line++)
        {
          for(int i = leftshift;i<lineend;i++)
          {
            RGB[i][line*squarelength+topshift][1] = 255;
            RGB[line*squarelength+leftshift][i-leftshift+topshift][1] = 255;

            RGB2[i][line*squarelength+topshift][1] = 255;
            RGB2[line*squarelength+leftshift][i-leftshift+topshift][1] = 255;
          }
        }

        int Board[8][8];

        int sum1 = 0;
        int sum2 = 0;
        int thresh = 150;

        for(int ii=0;ii<8;ii++)
        {
          for(int jj = 0;jj<8;jj++)
          {
        //Get locations. Threshhold will be very small
            for (int i = leftshift+ii*squarelength;i<leftshift+ii*squarelength+squarelength;i++)
            {
              for (int j = topshift+jj*squarelength;j<topshift+jj*squarelength+squarelength;j++)
              {
                if (RGB[i][j][2] > 0)
                  sum1++;
                if (RGB2[i][j][0] > 0)
                  sum2++;
                //RGB[i][j][1] = 20;
                //RGB2[i][j][1] = 20;
              }
            }
            if (sum1 > 5)
              Board[ii][jj] = 1;
            else if (sum2 > 5)
              Board[ii][jj] = 2;
            else
              Board[ii][jj] = 0;
         sum1 = 0;
         sum2 = 0;
          }
        }
        unsigned long long Blue = 0;
        unsigned long long Red = 0;
        //Print out board
        for(int i=7;i>=0;i--)
        {
          for(int j=7;j>=0;j--)
          {
            int ind = i*8 + j;
          cout << Board[7-j][7-i] << " ";

            if (Board[7-j][7-i] == 1)//Write to two variables
            {
              Blue |= 1;
            }
            else if (Board[7-j][7-i] == 2)
            {
              Red |= 1;
            }
            if(ind != 0)
            {
            Blue = Blue << 1;
            Red = Red << 1;
            }
          }
        cout << endl;
        }

        cout << Blue << endl;
        cout << Red << endl;

  FILE *myfile = fopen("a.ppm", "wb");
  {
    fprintf(myfile,"P3\n");
    fprintf(myfile,"520 400\n");
    fprintf(myfile,"255\n");
    fprintf(myfile,"# This is my header\n");
    for(int i = 0;i<height;i++)
    {
      for(int j = 0;j<width;j++)
      {
        for(int k = 0;k<3;k++)
        {
          fprintf(myfile,"%d ",RGB[j][i][k]);
        }
        fprintf(myfile,"\n");
      }
    }
    fclose(myfile);
  }

    FILE *myfile2 = fopen("b.ppm", "wb");
  {
    fprintf(myfile2,"P3\n");
```

```
    fprintf(myfile2,"520 400\n");
    fprintf(myfile2,"255\n");
    fprintf(myfile2,"# This is my header\n");
    for(int i = 0;i<height;i++)
    {
      for(int j = 0;j<width;j++)
      {
        for(int k = 0;k<3;k++)
        {
          fprintf(myfile2,"%d ",RGB2[j][i][k]);
        }
        fprintf(myfile2,"\n");
      }
    }
    fclose(myfile2);
  }

  //return would go here
  //instead, send data
  unsigned long tmp2 = 0;
  unsigned long long mask = 0xFFFFFFFF;
  tmp2 = Blue & mask;
  cout << tmp2 << endl;
  Send32Bits(tmp2);
  Blue &= ~(mask);
  Blue = Blue >> 32;
  tmp2 = Blue & mask;
  cout << tmp2 << endl;
  Send32Bits(tmp2);
  CommWait(1);

  tmp2 = Red & mask;
  cout << tmp2 << endl;
  Send32Bits(tmp2);
  Red &= ~(mask);
  Red = Red >> 32;
  tmp2 = Red & mask;
  cout << tmp2 << endl;
  Send32Bits(tmp2);


  }
  return 0;
}

void CommunicateSetup()
{
  //0 is input data
  //1 is input clock
  //2 is output data
  //3 is output clock

  wiringPiSetup();


  pinMode(0,INPUT);//Pin 0 input
  pinMode(1,INPUT);//Pin 1 input
  pinMode(2,OUTPUT);//Pin 2 output
  pinMode(3,OUTPUT);//Pin 3 output

}

void CommWait(int type)//Waits will need to be adjusted via the clock
{
  if (type == 0)
  {
    usleep(1500);
  }
  else
  {
    usleep(15000);
  }

}

unsigned long Get32Bits()
{
  int LastClock = digitalRead(1);//bit 0
  int ThisClock = digitalRead(1);

  int StartCheck1 = digitalRead(0);
  int StartCheck2 = digitalRead(1);

  unsigned long Data = 0;
  unsigned long tmp;

  while((StartCheck1 == 1) || (StartCheck2 == 1))
  {
    StartCheck1 = digitalRead(0);
    StartCheck2 = digitalRead(1);
  }

  while((StartCheck1 != 1) || (StartCheck2 != 1))
  {
    StartCheck1 = digitalRead(0);
    StartCheck2 = digitalRead(1);
  }

  for(int i=0;i<32;i++)
  {
    while(ThisClock == LastClock)
    {
      ThisClock = digitalRead(1);
      CommWait(1);
      LastClock = digitalRead(1);
    }

    tmp = digitalRead(0);
    tmp = tmp << i;
    Data |= tmp;

    LastClock = digitalRead(1);
    ThisClock = digitalRead(1);
  }

  return Data;
}

void Send32Bits(unsigned long OutData)
{
  digitalWrite(2,0);//Start of signal 00 then 11
  digitalWrite(3,0);
  CommWait(1);
  CommWait(1);
  CommWait(1);
  digitalWrite(2,1);
  digitalWrite(3,1);
  CommBit = 1;


  CommWait(1);
  CommWait(1);
  CommWait(1);
  for(int i=0;i<32;i++)
  {
    digitalWrite(2,0);//clear
    int tmp = (OutData & 1);
    digitalWrite(2,tmp);
    OutData = OutData - (OutData&1);
    OutData = OutData >> 1;

    CommWait(0);
    if (CommBit == 1)
    {
        digitalWrite(3,0);
        CommBit = 0;
    }
    else
    {
      digitalWrite(3,1);
      CommBit = 1;
    }
    CommWait(1);//Wait
    CommWait(0);
  }
}
```

XI. References

[1] "IEEE Guide on Shielding Practice for Low Voltage Cables," IEEE Std. 1143-2012, Mar. 4, 2013. [Online]. Available: https://standards.ieee.org/standard/1143-2012.html. [Accessed Dec. 7, 2018].

[2] Wiringpi.com, "Wiring Pi – GPIO Interface Library for the Raspberry PI," [Online]. Available: http://wiringpi.com/. [Accessed April 24, 2019].

[3] Imagemagick.org "ImageMagick," [Online]. Available: https://imagemagick.org/index.php. [Accessed April 24, 2019].

[4] Raspberrypi.org "Raspicam Commands," [Online]. Available: https://www.raspberrypi.org/documentation/usage/camera/raspicam/README.md. [Accessed April 24, 2019].

[5] keil.com "µVision IDE," [Online]. Available: http://www2.keil.com/mdk5/uvision/. [Accessed April 24, 2019].