Tic Tac Toe Robot

by

Dishan Fernando

Gornel Bah

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Bachelor Of Applied Science.

SIMON FRASER UNIVERSITY
Spring 2016

Approval

Name:	**Dishan Fernando**
	Gornel Bah
Degree:	**Mechatronic Systems Engineering**
Project:	**Tic Tac Toe Robot**
Examining Committee:	**Chair:** Amr Marzouk
	Professor

Date Approved: April 2016

Abstract

The objective of this project was to understand and develop a prototype with 2 DOF while overcoming the drawbacks of the bar code reader arm from previous assignments. More efficient mechanical design and improvement of the inverse kinematic algorithms were the basis for a successful project. for ‘Tic Tac Toe’. The Proportional Integral-Derivative controller, often known as the PID, which are commonly used in industrial control systems were also employed.

Dedication

This project and thesis is dedicated to all the underprivileged youth and their yet to be discovered inventions.

Progress Report

This project was not heavily scheduled. Because of the pure amount of time needed to complete it to specification by the deadline, most spare time was spent developing it. To reduce our reliance on all group members being available at the same time, we split the project between us and integrated all the parts during the final few days. Dishan was assigned the tic tac toe AI and algorithm design, in addition to coding its implementation, while Gornel handled the physical construction, arm control software, and graphics. The selected strategy worked out well as the different parts came together easily and without notable issue.

Table of Contents

Approval……………………………………………………………….……………………İİ

Abstract…………………………..……………………………………………………………İİİ

Dedication……………………..………………………………………………………………III

Progress Report………..……………………………………………………………………III

Table of Contents………..……………………..…………………………………………........IV

List of Figures……………………………..……………………………………………......Vİ

1.Introduction……………..………………………………………………………....……1

    1.1 Tic Tac Toe….………..…………………………………………………..…………...1

2.Mechanical Design...……..……………………………………………..………………1

    2.1 Construction………………………………………………………………………...1

3.Software Design…………………………..……………………………………..………2

    3.1 Coding Introduction………………………………………………………………...2

    3.2 Game Logics and Functions ……………………………………………………….3

Table of Contents

4.The Arm Control File…………………………..……………………………………………..9

4.1 Modularity………………………………………………………………………..9

4.2 Inverse Kinematic Calculations…………………………………………...……..9

4.3 PID Tasks………………………………………………………………………..9

4.4 Important Variable and Functions…………………………...………..…………..10

6.Conclusion ..……...……..……………………………………………..…………..…...…11

7.References …………..………………………………………..………………..……...12

8.Code – Main File……………………………………...……………………………… 16

9.Code – Arm Control File……………………………………………………………………25

List of Figures

Figure 1.1 Mechanical Design…..……………………………………..…………………...1

Figure 3.1 Winning Strategy…………………….. ………………..……………………3

Figure 3.2: Strategy Flowchart..………………………………………………………………5

1.Introduction

1.1 Tic Tac Toe

Tic-tac-toe (also known as **Noughts and crosses** or **Xs and Os**) is a paper-and-pencil game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. There are many traps that can be devised in the first two turns which need to be accounted for in order to win. The game is a lot more subtle than it appears.

2.Mechanical Design

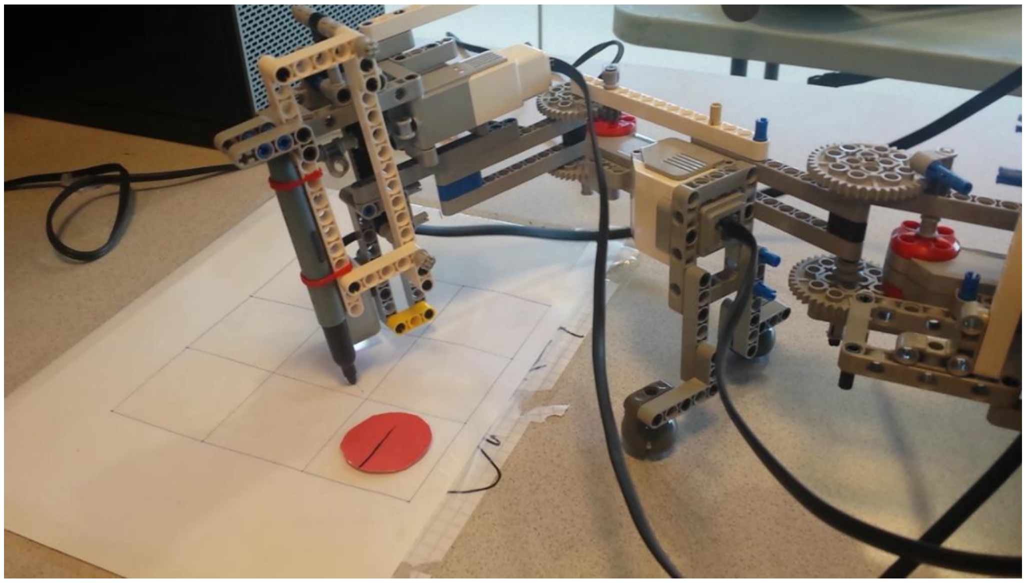


Figure 1.1: Mechanical Design

2.1 Construction

The robot consists of a lego brick, one colour sensor, 3 motors and other necessary building parts. It is a 2 Degree of freedom robot and its end point moves to a specific target by using inverse kinematics. These mathematics are used over forward kinematics because of the latter’s disadvantages in getting two possible angles for the same end point.

The robot has two arms each connected to motors where the writing tool is connected to a third motor that comes down once the target location is reached and goes back once the x is drawn. The lengths of the robot arms are carefully measured and used in the calculation in order to make the robot as precise as possible. A clamp has also been attached to the base for increased stability and to dampen any turbulence which may affect the calculations. Original designs which did not use this clamp would have the robot shift slightly and cause the arms lean heavily at certain angles, resulting in skewed degrees of freedom. The brick also had to be moved on top of the first arm motor in order to accommodate the short cord lengths, which have been left free hanging for maximum mobility. A dual-ball system was then added on the first arm to provide the necessary support for the pen control motor. The pen and light sensor were specifically designed to be as close as possible, in order to negate the need for offsets during positional calculations.

3.Software Design

3.1 Coding Introduction

The grids in the paper are represented as a matrix namely, (1,1) (1,2) and so on, thus a 3x3 character is created. In the beginning of the game, the robot asks for the players choice to play first or second. Every time the opponent plays, the arm moves through the grid and updates the current status of the board and plays accordingly, the priority being, trying to win and then stopping the opponent from winning with a draw.

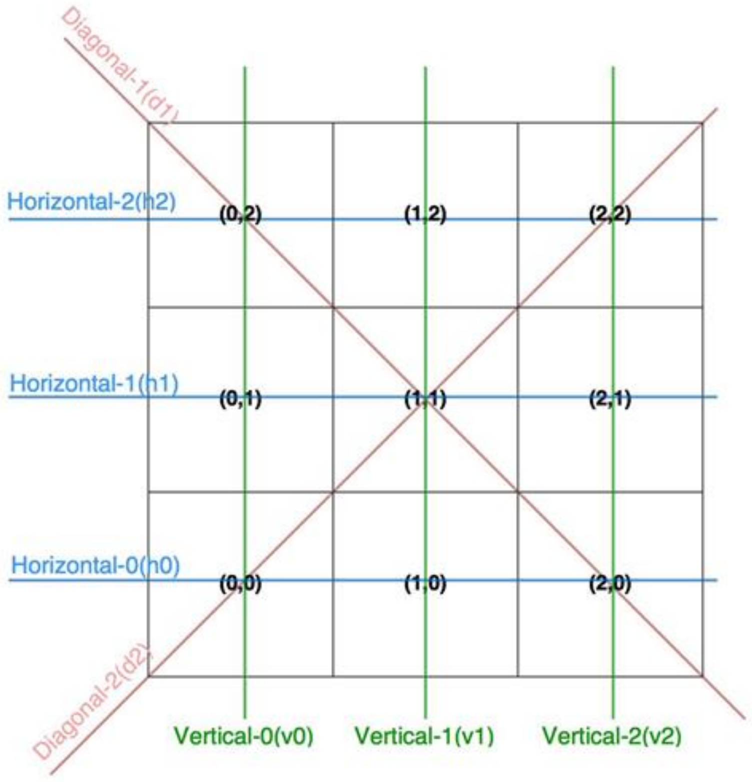


Figure3.1 : Winning Strategy

The following variables are used

Mat_char [ ][ ]: This is the character matrix that stores the status on the board.

at_x & at_y : These carry the x and y coordinates of the target location.

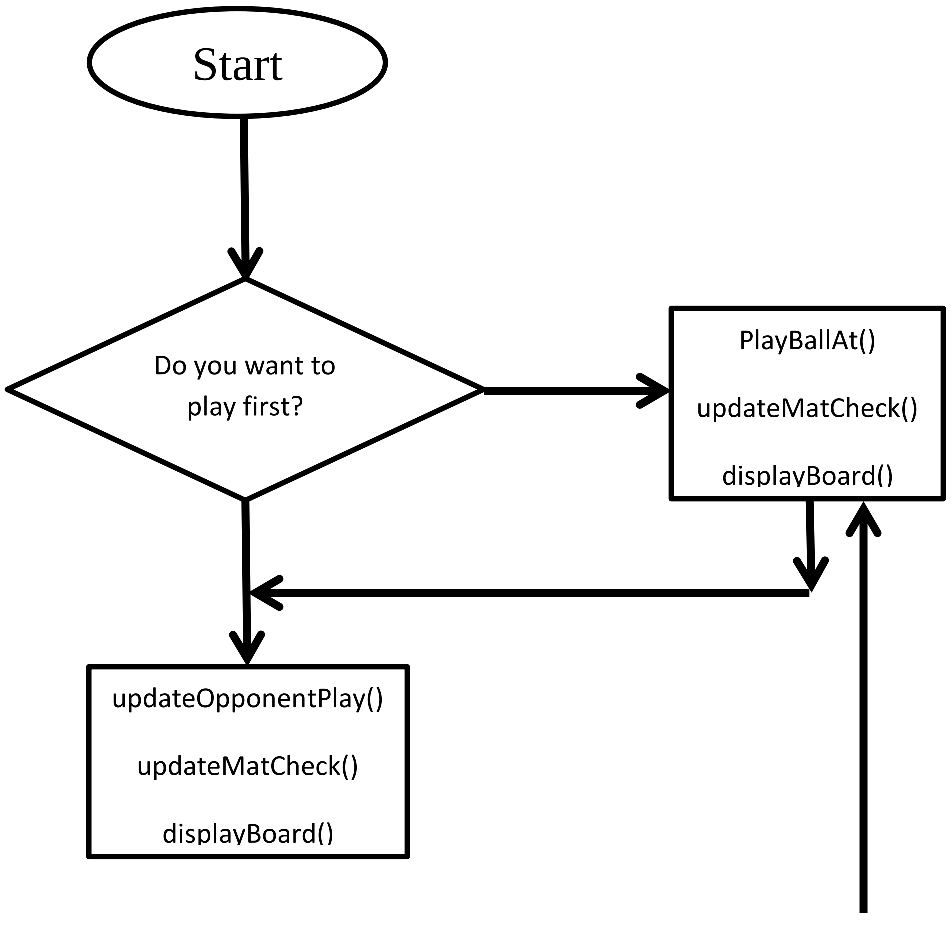
3.2 Game Logic and Functions

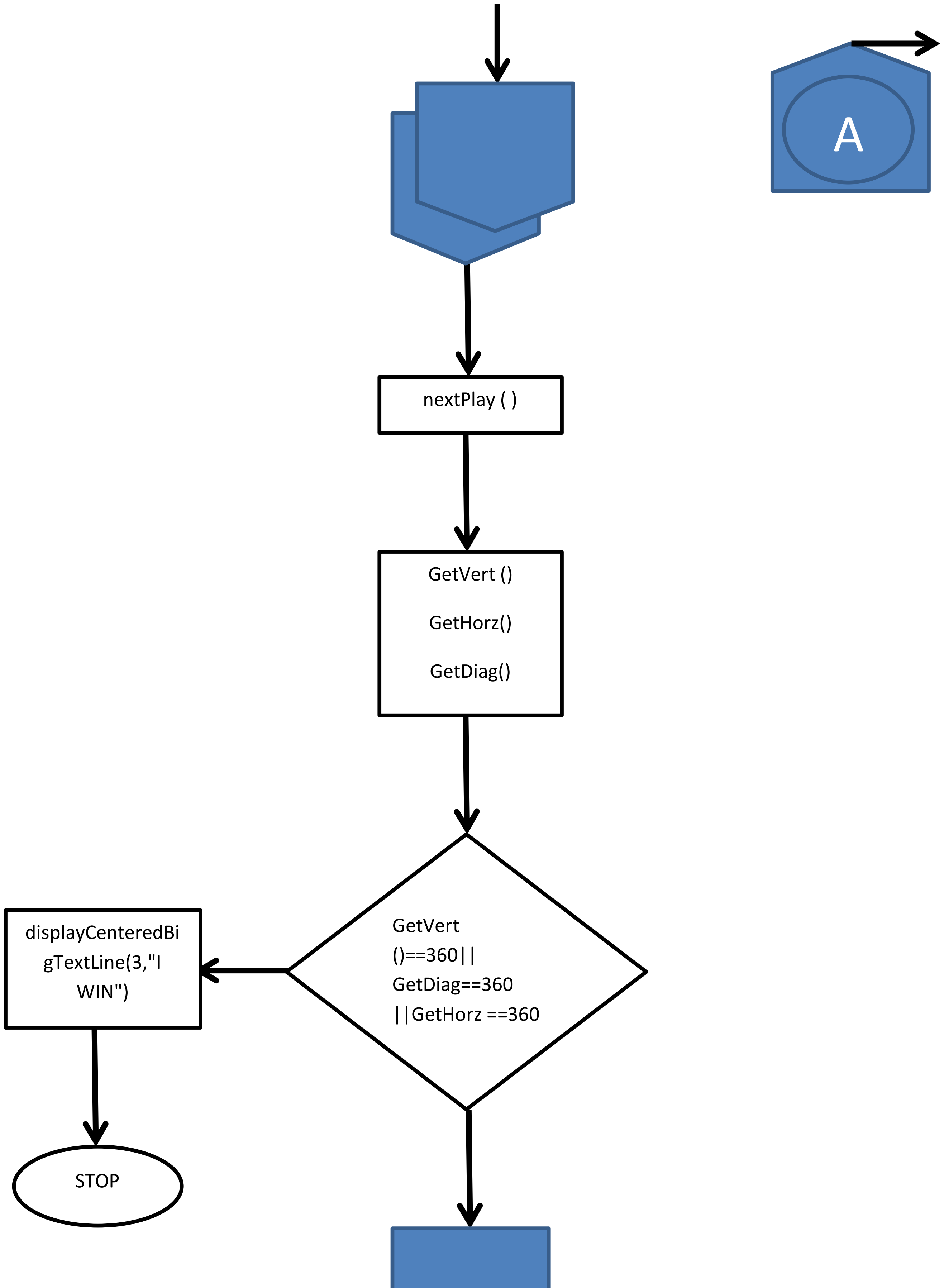
Several functions were used for making the program more user-friendly and readable, namely

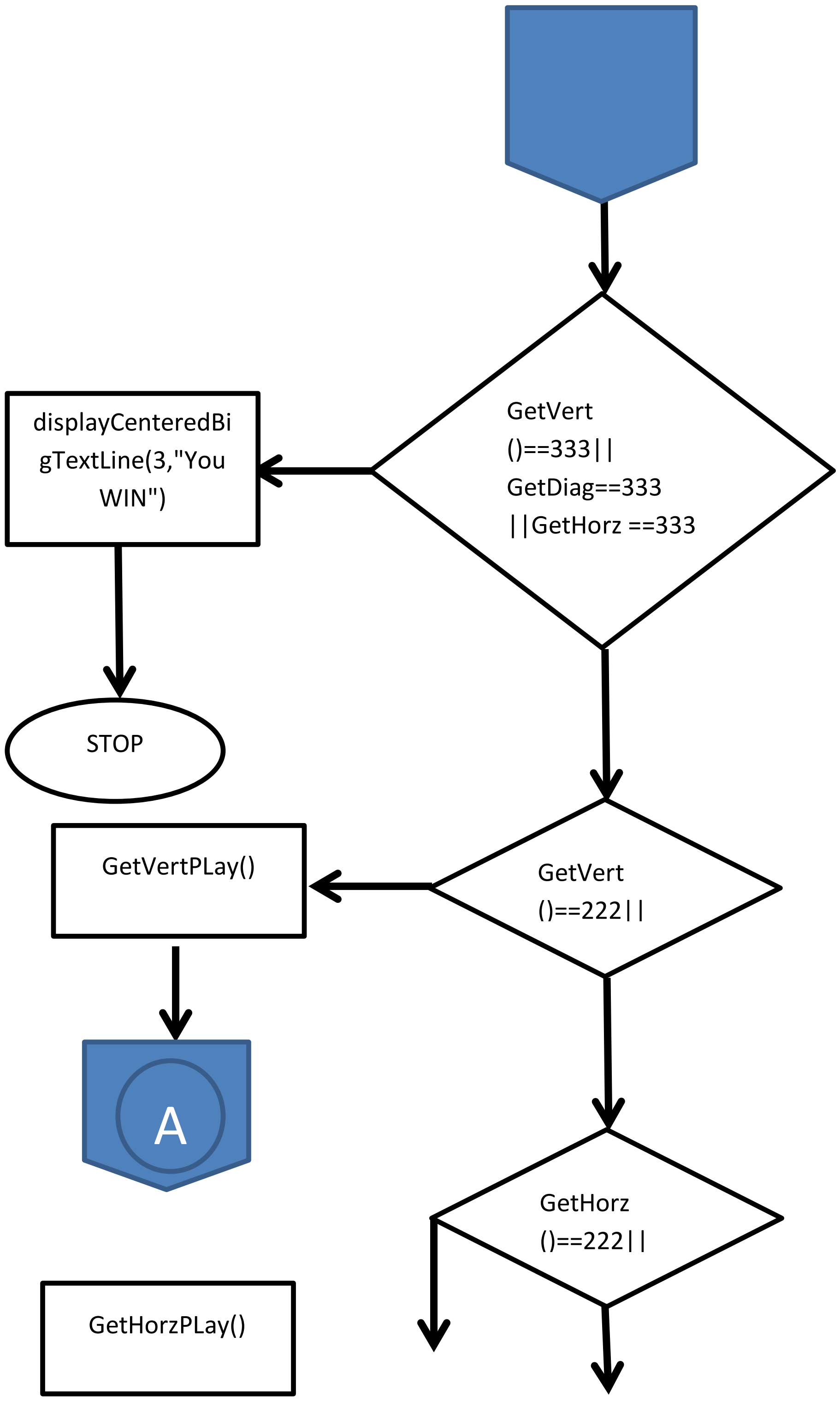
UpdateOpponentPlay( ) : This function moves the arm throughout the grid and scans for the red circle, once the red circle has been spotted it stops scanning and updates the move played by the opponent in the character matrix by assigning ‘o’.

resetMatCheck ( ) **:** This function is called by the former defined function and it updates the values in the matrix. It also assigns ‘x’ in appropriate places every time the robot plays.

nextPlay ( ) **:** This function enable the user to play the next game once the robot has finished its move. This is controlled by a Boolean that changes for every turn. This calculates the numerical values of the rows, columns and diagonals using several other functions and decides where to do the next move.




A
nextPlay ( )
GetVert ()
GetHorz()
GetDiag()
GetVert ()==360||
GetDiag==360
||GetHorz ==360
displayCenteredBigTextLine(3,"I WIN")
STOP


GetVert
()==333||
GetDiag==333
||GetHorz ==333
displayCenteredBigTextLine(3,"You WIN")
STOP
GetVert
()==222||
GetVertPLay()
A
GetHorz
()==222||
GetHorzPLay()

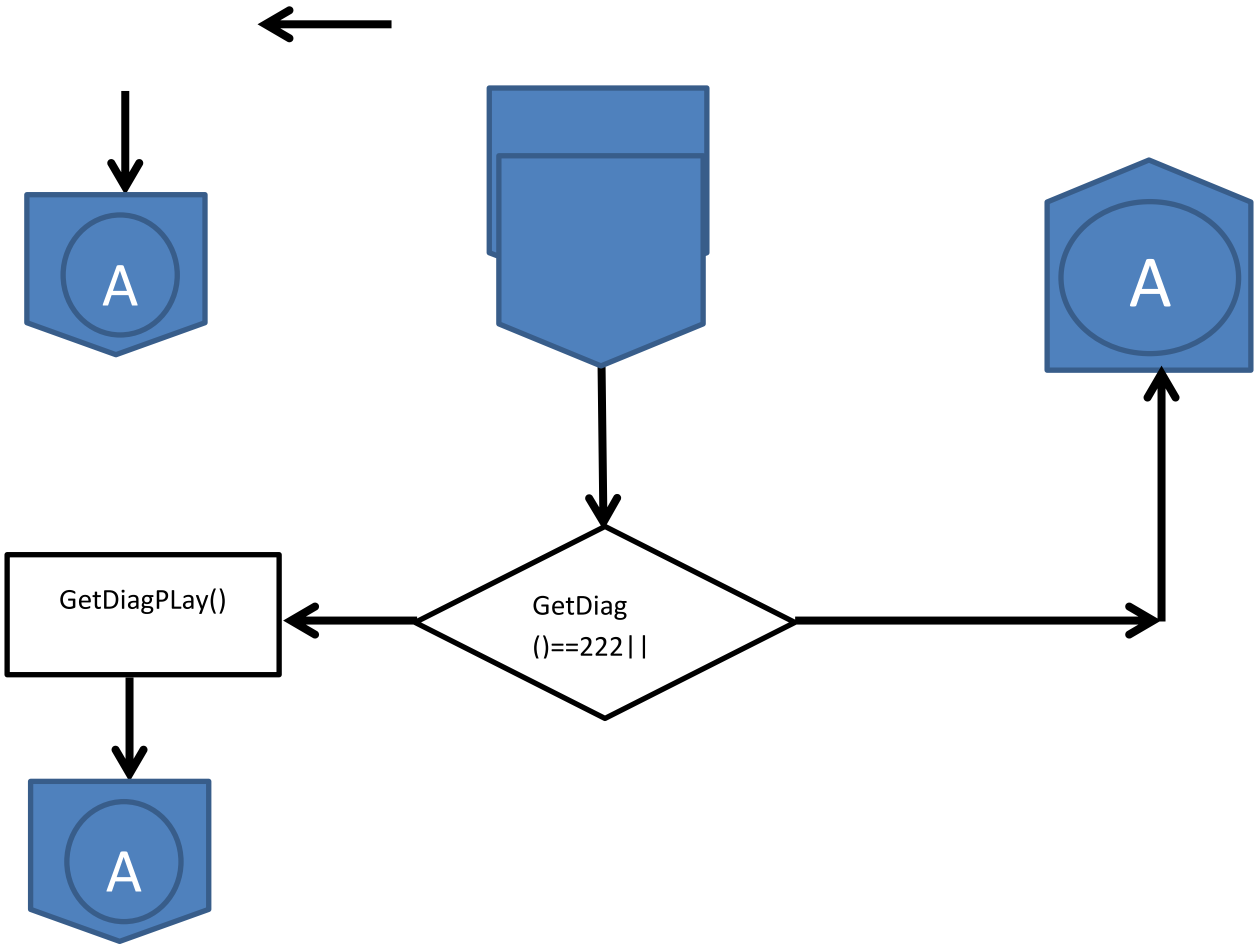


Figure 3.2: Strategy Flowchart

STRATEGY

The following functions mainly handle the strategy of the game. The robot first scans the grid, updates the matrix. Then it looks if the opponent has a winning opportunity i.e. two ‘O’s in a row or column or diagonal. If yes, then its first priority is to stop it. If not, it looks if there are two ‘X’s in a row or column or diagonal if yes it tries to draw the third ‘X’, if not it checks along the vertical, horizontal and doagonal to see if there is any hindrance and thus draws an ‘X’ accordingly.

Looking for the vertical, horizontal and diagonals are done by the following funcitons

GetVert ( ) **:** This function gets the numerical value of the vertical columns by calculating the ASCII values of ‘X’ and ‘O’. This returns the sum of the values in the grids along each vertical column.

GetHorz ( ) **:** This function gets the numerical value of the horizontal rows by calculating the ASCII values of ‘X’ and ‘O’. This returns the sum of the values in the grids along each vertical column.

GetDiag ( ) **:** This function gets the numerical value of the diagonals by calculating the ASCII values of ‘X’ and ‘O’. This returns the sum of the values in the grids along each vertical column.

PlayBallAt ( ) **:** Once the above functions are all called, the robot decides where to make the next move with accordance to the priority. Thus PlayballAt( ) function is called and the X and Y co-ordinated are passed through the function. This function in turn calls other functions to convert the XY coordinates into actual coordinates and also updates the matrix with the new status of the board.

GetVertPlay ( ) **:** After checking the status of the vertical columns using the above defined functions, if the move has to be done in one of the vertical columns, this function looks through the entire matrix and decides where to make the next move. It looks if anybody has played in a specific location and if there are two possible locations, the best choice with accordance to the given priority is chosen.

This is done with the ASCII values returned from the **GetVert( )** function. Say if it returns 111 (ASCII value of O) which means there is only one ‘O’ in the column, if its 222 then there are two ‘O’s. Similarly 120 for one ‘X’ and ‘240’ for two ‘X’s and so on. This function has algorithm to make decisions according to the retuned values.

GetHorzPlay ( ) **:** After checking the status of the horizontal rows using the above defined functions, if the move has to be done in one of the vertical columns, this function looks through the entire matrix and decides where to make the next move. It looks if anybody has played in a specific location and if there are two possible locations, the best choice with accordance to the given priority is chosen.

This is done with the ASCII values returned from the **GetHorz( )** function. Say if it returns 111 (ASCII value of O) which means there is only one ‘O’ in the column, if its 222 then there are two ‘O’s. Similarly 120 for one ‘X’ and ‘240’ for two ‘X’s and so on. This function has algorithm to make decisions according to the retuned values.

GetDiagPlay ( ) **:** After checking the status of the horizontal rows using the above defined functions, if the move has to be done in one of the vertical columns, this function looks through the entire matrix and decides where to make the next move. It looks if anybody has played in a specific location and if there are two possible locations, the best choice with accordance to the given priority is chosen.

This is done with the ASCII values returned from the **GetDiag( )** function. Say if it returns 111 (ASCII value of O) which means there is only one ‘O’ in the column, if its 222 then there are two ‘O’s. Similarly 120 for one ‘X’ and ‘240’ for two ‘X’s and so on. This function has algorithm to make decisions according to the retuned values.

PlayGame ( ) **:** This is the function that keeps track of who is playing. It asks for the player’s choice for being the first player. It also creates the character array and sets the matrix to zero in the beginning of the program and changes the Boolean values to facilitate the feasibility of the game.

foePlayCount ( ) **:** This function keeps track of the number of turns played by the player and is called upon during crucial decision making times.

DisplayBoard ( ) **:** This function is used to display the 3x3 grid on the lego brick and is updated every time there is a move, it uses several other functions like UpdateOpponentPlay ( ), resetMatCheck ( ) and so on to be on par with the game.

ExtraWinCheck ( ) **:** After checking the status of the horizontal rows, vertical columns and diagonals using the above defined functions namely **GetVert ( )**, **GetHorz ( ),GetDiag ( ).** This functions decised who the winner is.This is done with the ASCII values returned from the above functions. Say if it returns 333 (ASCII value of three O’s) which means there are three ‘O’s in a specific column or row or diagonal, it declares the winner. Similarly if it returns 360 (ASCII value of three X’s) which means there are three ‘X’s in a specific column or row or diagonal, it declares the winner.

4.The Arm Control File

4.1 Modularity

A separate file was used for all software related to controlling the arms in order to preserve an easily understood main program. This strategy also enabled different parts of the robot to be developed independently. The main software file only calls a single function, passing to it the tic tac toe location where it wants to go, and the arm control file handles the rest behind the scenes. This subsystem is an extension of the original barcode reader, but has been much improved upon by using concepts that were taught in class since that project.

4.2 Inverse Kinematic Calculations

All locations are calculated in real time using inverse kinematics. Once the game software wants a certain position, that position is translated into more precise coordinates without bothering the user, and the angle of each arm will be calculated individually. These angles are stored inside global variables to ensure that the entire control program knows what its current goal is. Only a single function is needed in order to do this, and can be called from anywhere in the file. Drawing an X on the board just tells the arm to go to a corner location, wait for the pen to be lowered and then travel to another corner location before returning to the standby position. All of this is done without the need for hard coded angles.

4.3 PID tasks

The most important feature of the arm control file are the arm control tasks. As soon as the program begins, they start running continuously in parallel with the main program and do not end until it is shut down. There is one task for each arm and each one's goal is to move their motor to the currently desired angle. This is done using PID to calculate for accuracy errors that are common in the motors, and to allow maximum speed without sacrificing that accuracy. Because of the way they are implemented, simply setting the global target angle will cause the arms to travel to that exact location without a function even having to be called. It is very convenient. This implementation also allows angles to be changed mid-travel and asynchronously with instant reaction time. Even when the arm is in a stationary position, the tasks are constantly correcting any disturbance to the arm and holding it at the correct location.

4.4 Important Variables and Functions

Void drawXat(float x, float y) – The function to draw an X. Enter the square you want and it will assume it is a 5cm by 5cm area, then decide on what locations to use and call all functions needed to control the pen and move around in creation of the X, automatically. This is the first of two functions used in the main files gameplay software.

Void Goto(float xcoord, float ycoord, bool scanmod) – The foundation for all of the simple functions in the program. It translates the square you want to go to into coordinates for the backend to use. This is the second of the two functions used in the main files gameplay software.

Void calculateInstructions(float x, float y) – Contains all math for inverse kinematics. This is usually called by GoTo(x, y) after it has translated the users location into coordinates to pass into this function. Once given coordinates, this will convert them to an angle for each arm and store them in global variables for the entire file to see.

float arm1Target, float arm2Target – These are the global variables that hold target angles for each arm, which the entire program can see.

Task moveArm1(), task moveArm2() – The PID control tasks for the motors. A work of art. They will instantly move to the new angles stored in the arm target variables when they are changed, and correct for any motor inaccuracies along the way.

Conclusion

The concept of 2 and 3 degree of motion was very well understood. We were able to find and overcome several barriers using inverse kinematics. When the groundwork had been laid, we were able to meet every criterion in the given specifications and found very efficient ways to handle certain procedures. We were able to explore and acquire numerous new programming skills while practically implementing on a real physical robot. MATLAB was also very useful in simulating the outcomes while coding. We became very familiar with several algorithms and also made full use of PID to enhance the robots its efficiency. This was a very good opportunity to combine mechanical and software design in a way that complement each other. Overall the project was a great learning experience and helped develop team work and time management.

References

1. Ivanov G.E , Kazeev V.A (2011)**.** Minimax algorithm for constructing an optimal control strategy in differential games with a Lipschitz payoff, *Computational Mathematics and Mathematical Physics***,** *51*(4), 550-574
2. Tic Tac Toe: Understanding the Minimax Algorithm. (2013). *NeverstopBuilding*.com
3. https://en.wikipedia.org/wiki/Tic-tac-toe

Main Code – ttt_main

```
/*
Gornel Bah, 301235845
Dishan Fernando, 301270036
video:
https://www.youtube.com/watch?v=pKBV_TG_RIs

Our project uses two files, this is the main. We will also give you the armcontrol(the second file).
The project does everything asked, including bonuses. Nothing is hard coded
and all calculations and algorithms are done in real time.


This file can do:
tic tac toe algorithm in - void nextPlay();
efficiently scan board with algorithm in - void updateOpponentPlay()
display board and other information on screen in - void displayBoard()
track state of the board in char array with x and o's with - char char_mat[3][3]
and any other non-kinematic functions


The second file can do:
PID - task moveArm1(), task moveArm2()
draw X's - void drawXat(float x, float y)
go to any grid location using - void Goto(float xcoord, float ycoord, bool scanmod)
^all this using inverse kinematics at - void calculateInstructions(float x, float y)
and any other arm control related functions

Pretty much everything else in that file is for translating simple user inputs of board spots into coordinates
for the kinematics to use.

What this code can't do: Brew your coffee.

*/

#pragma config(Sensor, S4,     colorSensor,    sensorEV3_Color, modeEV3Color_Color)
#pragma config(Motor,  motorA,          arm1,          tmotorEV3_Large, openLoop)
#pragma config(Motor,  motorC,          arm2,          tmotorEV3_Large, openLoop)
#pragma config(Motor,  motorD,          pen,           tmotorEV3_Medium, PIDControl, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

#include "Group37_armcontrol.c"
//------------------------------------------------------------------
//
            ||DECLARATIONS|| ROBOT NAME = gmb
//------------------------------------------------------------------
//

bool foundOppPlay = false;
bool playedTurn = false;
bool restartGame = true;
bool matCheck[3][3];
int emptyBox = 0;
int foe = 5;
int robot = 7;
void updateOpponentPlay();
//void playBallAt(int x, int y);
void updateMatCheck();
void resetMatCheck();
void nextPlay();
//void resetMatCheck();
int getVert(int col);
int getHorz(int row);
int getDiag(int oneTwo);
void playBallAt(int x, int y);
void vertPlay(int col);
void horzPlay(int row);
void diagPlay(int oneTwo);
void playGame();
bool gmbPlaysFirst();
bool gmbFirst = false;
int foePlayCount = 0;
void displayBoard();
void extraWinCheck();


//GLOBAL VARIABLES
/************************************************************
**************X'S AND O'S ARE STORED HERE*********************
************************************************************/
char char_mat[3][3];
int atX = 0;
int atY = 0;
int fwd =  100000000;
int bkwd = -100000000;
int mat[3][3];

void displayBoard(){
drawLine(50, 10, 50, 120);
drawLine(100, 10, 100, 120);
drawLine(10, 50, 150, 50);
drawLine(10, 90, 150, 90);

for(int i=0; i<3; i++){
                for(int j=0; j<3;j++){
                        if(mat[i][j]== 5){
                                drawEllipse(i*50+10, j*50, 25+(i*50)+10, 25+(j*50));
                        }else if (mat[i][j]== 7){
                                drawLine(i*50+10, j*55, (i*50)+40, 25+(j*50)+10);
                                drawLine(i*50+40, j*55, (i*50)+10, 25+(j*50)+10);
                                //drawRect(i*50+10, j*50, 25+(i*50)+10, 25+(j*50));
    sleep(100);
                        }
                }
        }
}
        ///////////////////////////////////////////////////
        ////////***********MAIN FUNCTION***********//////////
        ///////////////////////////////////////////////////
```

```
task main()
{

                //SETUP:
        //LINE UP INSIDE SUPPORT WITH 0,2 AND FINAL WITH 3,1.5
        //////////////////////////////////////////////////////
        resetMotorEncoder(arm1);
        resetMotorEncoder(arm2);
        //sets the starting point where the arm is
        goHome();
        startTask (moveArm1);
        startTask (moveArm2);

        playGame();

        while (true){
                if (restartGame == true){
                        //wait 5 seconds to restart game
                sleep(5000);
                playGame();
                }
        }

}

void playGame(){
        restartGame = false;
        //initialize array to zeros
        for(int i=0; i<3; i++){
                for(int j=0; j<3;j++){
                        mat[i][j] = 0;
                }
        }
        sleep(50);


        for(int i=0; i<3; i++){
                for(int j=0; j<3;j++){
                        char_mat[i][j] = NULL;
                }
        }
        sleep(50);

        gmbFirst = gmbPlaysFirst();
        if(gmbFirst){
                playBallAt(1,1);
        }
        eraseDisplay();
        while (restartGame == false){
                //eraseDisplay();
                displayBoard();
                displayCenteredTextLine(12," Your Turn!   ");
                displayCenteredTextLine(14,"Press Middle Button to End.");
                //Enter button for opponent play
                if (getButtonPress(buttonEnter)){
                        eraseDisplay();
```

```
                              displayBoard();
                              //scan for red circle and put it into array
                              updateOpponentPlay();
                              //show where opponent went
                              displayBoard();
                              //update turn counter
                              foePlayCount = foePlayCount + 1;
                              //Opponent has played, my turn
                              playedTurn = false;
                              //look for spot to place my x
                              updateMatCheck();
                              int count = 0;
                              for(int i=0;i<3;i++){
                                    for(int j=0;j<3;j++){
                                          if(mat[i][j]!=0){
                                                count++;
                                                }
                                          }
                              }
                                    if (count == 9){
                                    displayCenteredTextLine(2," It's a DRAW !! ");
                                    }
                               // If none of the cells are empty
                                    else if (restartGame == false){
                                          nextPlay();//PLAY GAME.. Keep a boolean that lets the program
run only when its true
                                          extraWinCheck();
                                    }


                        }
            }
}

//-------------------------------------------------------------------
//
FUNCTIONS

      //Reset matCheck
void resetMatCheck(){
      for (int i = 0; i<3; i++){
            for (int j = 0; j<3;j++){
                  matCheck[i][j] = false;
            }
      }
      sleep(50);
}
//-----------------------------------------------------------------
// SCANS FOR A RED DOT
//-----------------------------------------------------------------
void updateOpponentPlay(){
      /*if(mat[atX][atY] == emptyBox){// && !(matCheck[atX][atY])){//
            */
            bool stopScanning = false;
while (stopScanning == false){
      for (float j = 2; j >= 0; j-=1)
```

```
        {
                for (float i = 2; i >= 0;i-=1)
                {
                        Goto(i+0.5,j+0.5, true);
                        if (scanForRed() == true && mat[i][j] != foe){
                                mat[i][j] = foe;
                                char_mat[i][j]='o';
                                matCheck[i][j] = true;
                                foundOppPlay = true;
                                //trigger breaks in the for loops containing this
                                stopScanning = true;
                        }
                        //break the for loop
                        if (stopScanning == true)
                                break;
                }
                //break the for loop
                if (stopScanning == true)
                                break;
        }
        goHome();
        playTone(30,100);
        sleep(3000);
}
        displayCenteredTextLine(10,"You played at (%d,%d)",atX,atY);
}

void playBallAt(int x, int y){

        displayCenteredTextLine(5,"going  to  play  at  (%d,%d)",x,y);
        drawXat(x, y);

        //because of errors dont use mat[atX][atY]
        mat[x][y] = robot;
        char_mat[x][y]='x';
        playedTurn = true;
        eraseDisplay();
        displayBoard();
}

void diagPlay(int oneTwo){
        if(oneTwo == 1){
                if(mat[0][2] == emptyBox && !(playedTurn)){playBallAt(0,2);}
                else if(mat[1][1] == emptyBox && !(playedTurn)){playBallAt(1,1);}
                else if(mat[2][0]== emptyBox && !(playedTurn)){playBallAt(2,0);}
        }
        else if(oneTwo == 2){
                if(mat[0][0] == emptyBox){playBallAt(0,0);}
                else if(mat[1][1] == emptyBox){playBallAt(1,1);}
                else if(mat[2][2]== emptyBox){playBallAt(2,2);}
        }
}

void horzPlay(int row){
        for (int i = 0; i<3; i++){
                if(mat[i][row] == 0 && !(playedTurn)){
```

```
				playBallAt(i,row);
			}
		}
}

void vertPlay(int col){
		for (int i = 0; i<3; i++){
			if(mat[col][i] == 0 && !(playedTurn)){
				playBallAt(col,i);
			}
		}
}

int getDiag(int oneTwo){
		int result = 0;
		if(oneTwo == 1){
			result = (int)char_mat[0][2] + (int)char_mat[1][1] + (int)char_mat[2][0];
		}
		else{
			result = (int)char_mat[0][0] + (int)char_mat[1][1] + (int)char_mat[2][2];
		}

		return result;
}

int getVert(int col){
		int result = 0;
		for (int i = 0; i<3; i++){
			result = result + (int)char_mat[col][i];
		}
		return result;
}

int getHorz(int row){
		int result = 0;
		for (int i = 0; i<3; i++){
			result = result + (int)char_mat[i][row];
		}
		return result;
}


void updateMatCheck(){
		//Reset matCheck
		resetMatCheck();

		//look for where to play
		for (int i = 0; i<3; i++){
			for (int j = 0; j<3;j++){
				if(!(matCheck[i][j]) && mat[i][j]==emptyBox && !(foundOppPlay)){
				//if It has not check this spot on the mat and
				//		There is no ball on that spot of the map and
				//		It still hasn't found the opponent's play
					//Goto(i,j);
					//a break statement could make a tiny difference here
				}
		}
	}

	//Reset foundPlay
	foundOppPlay = false;
}

void extraWinCheck(){
	updateMatCheck();
	//displayCenteredTextLine(7,"Performing nextPlay at")
	int d1 = getDiag(1); int d2 = getDiag(2);
	int v0 = getVert(0);int v1 = getVert(1);int v2 = getVert(2);
	int h0 = getHorz(0);int h1 = getHorz(1);int h2 = getHorz(2);

	if(mat[1][1]==emptyBox && foePlayCount==1 &&!(playedTurn)){
		//playBallAt(1,1);
	}
	else{

		if(v0==360 || v1==360 || v2 == 360 || h0==360 || h1==360 || h2 == 360|| d1 ==
360||d2==360){
			eraseDisplay();
			displayCenteredBigTextLine(3,"I WIN");
			displayCenteredBigTextLine(5,"New game in 5 seconds.");
			restartGame = true;
		}
		else if(v0==333 || v1==333 || v2 == 333 || h0==333 || h1==333 || h2 == 333|| d1 ==
333||d2==333){
			eraseDisplay();
			displayCenteredBigTextLine(3,"YOU WIN");
			displayCenteredBigTextLine(5,"New game in 5 seconds.");
			restartGame = true;
		}
	}
}

void nextPlay(){
	//displayCenteredTextLine(7,"Performing nextPlay at")
	int d1 = getDiag(1); int d2 = getDiag(2);
	int v0 = getVert(0);int v1 = getVert(1);int v2 = getVert(2);
	int h0 = getHorz(0);int h1 = getHorz(1);int h2 = getHorz(2);

	if(mat[1][1]==emptyBox && foePlayCount==1 &&!(playedTurn)){
		playBallAt(1,1);
	}
	else{

		if(v0==360 || v1==360 || v2 == 360 || h0==360 || h1==360 || h2 == 360|| d1 ==
360||d2==360){
			eraseDisplay();
			displayCenteredBigTextLine(3,"I WIN");
			displayCenteredBigTextLine(5,"New game in 5 seconds.");
			restartGame = true;
		}
		else if(v0==333 || v1==333 || v2 == 333 || h0==333 || h1==333 || h2 == 333|| d1 ==
333||d2==333){
        eraseDisplay();
        displayCenteredBigTextLine(3,"YOU WIN");
        displayCenteredBigTextLine(5,"New game in 5 seconds.");
        restartGame = true;
}

//1---1)CHECK WINS opportunities first
if(v0==14){vertPlay(0);}
else if(v1==240 ){vertPlay(1);}
else if(v2==240 ){vertPlay(2);}

//horizontals
else if (h0==240){horzPlay(0);}
else if (h1==240 ){horzPlay(1);}
else if (h2==240){horzPlay(2);}

//diagonals
else if(d1==240){diagPlay(1);}
else if(d2==240){diagPlay(2);}

//2---2)Then BLOCKING OPPONENT
//DIRECT BLOCKING
//diagonals
else if(d1==222){diagPlay(1);}
else if(d2==222){diagPlay(2);}

//vertical
else if(v0==222){vertPlay(0);}
else if(v1==222){vertPlay(1);}
else if(v2==222){vertPlay(2);}

//horizontals
else if (h0==222){horzPlay(0);}
else if (h1==222){horzPlay(1);}
else if (h2==222){horzPlay(2);}


//INDIRECT BLOCKING:

//diagonals
else if((d1%foe) == 0){diagPlay(1);}
else if((d2%foe)==0){diagPlay(2);}

//verticals
else if((v0%foe)==0){vertPlay(0);}
else if((v1%foe)==0){vertPlay(1);}
else if((v2%foe)==0){vertPlay(2);}

//horizontals
else if ((h0%foe) == 0){horzPlay(0);}
else if ((h1%foe) == 0){horzPlay(1);}
else if ((h2%foe) == 0){horzPlay(2);}

//play anywhere
else{
        for (int i = 0; i<3; i++){
```

```
                        for(int j = 0; j<3; j++){
                            if(mat[i][j] == emptyBox && !(playedTurn)){
                                playBallAt(i,j);
                                break;
                            }
                        }
                    }
                    sleep(50);
                }

                sleep(50);
        }
}

bool gmbPlaysFirst(){
        bool result = false;
        while (true){
                eraseDisplay();
                displayCenteredTextLine(4,"        RButton = You First------->");
                displayCenteredTextLine(5,"<-------LButton = Me First");

                if (getButtonPress(buttonRight)){
                        displayCenteredTextLine(3,"PLAY!!!!");
                        result = false;
                        break;
                }
                else if (getButtonPress(buttonLeft)){
                        displayCenteredTextLine(3,"I PLAY!!!!");
                        result = true;
                        break;
                }
        }

        return result;

}
```

Main Code – ttt_armcontrol

```
#pragma config(Sensor, S4,     colorSensor,    sensorEV3_Color, modeEV3Color_Color)
#pragma config(Motor,  motorA,          arm1,          tmotorEV3_Large, openLoop)
#pragma config(Motor,  motorC,          arm2,          tmotorEV3_Large, openLoop)
#pragma config(Motor,  motorD,          pen,           tmotorEV3_Medium, PIDControl, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

//arm lengths
	//arm1 = 18 holes
	//arm2 = 16 and 1 holes
float L1=19;
float L2=14;

//More options
float inGearSize = 8;
float outGearSize = 40;
float gearRatio;

float arm1Target = 0;
float arm2Target = 0;



////////////////
//DECLARATIONS//
////////////////

//for calculating the multiple theta options
float c2;
float Part_1a;
float Part_1b;
float Part_2;
float Theta2_Option_A_rad;
float Theta2_Option_B_rad;
float theta2_rad;
float theta2_deg;
float k1;
float k2;
float theta1_rad;
float theta1_deg;

void calculateInstructions(float x, float y)
{
	gearRatio = (outGearSize/inGearSize);
	//adding 30 starts the calculation from zero?
		//calculate theta2
		c2 = (pow(x, 2) + pow(y, 2) - pow(L1, 2) - pow(L2, 2))/(2 * L1 * L2);
   Part_1a = sqrt(1 - pow(c2, 2));
   Part_1b = -sqrt(1 - pow(c2, 2));
```

```
    Part_2 =(c2);

    Theta2_Option_A_rad = atan2(Part_1a,Part_2);
    Theta2_Option_B_rad = atan2(Part_1b,Part_2);

    //use the theta2 option that has a smaller distance
    if (Theta2_Option_A_rad < Theta2_Option_B_rad){
      theta2_rad=atan2(Part_1a,Part_2);
     }
    else{
      theta2_rad=atan2(Part_1b,Part_2);
                    }

    theta2_deg = radiansToDegrees(theta2_rad);

    k1=L1+L2*cos(theta2_rad);
    k2=L2*sin(theta2_rad);

    //find theta1
    theta1_rad = atan2(y,x)-atan2(k2,k1);
    theta1_deg = radiansToDegrees(theta1_rad);

    //////////////////////////////////////
    //update the arm targets with the new instructions
    //////////////////////////////////////
    if (abs(theta1_deg) < 150){
    arm1Target = -theta1_deg;
          }
          else{
                    arm1Target = -theta1_deg + 360;
                    //playTone(10, 10);

          }

                    if (theta2_deg > 90){
                    arm2Target = -theta2_deg + 360;
                    }
                    else{
                              arm2Target = -theta2_deg;
                    }
}

////////////////////////////////
//MOVE THE ARMS TO GIVEN ANGELS//
///////////////////////////////

//ARM1
task moveArm1(){

float error;
float error_diff;
float speed;
float error_past;
float errorintg;
float current = 0;
float mostRecentTarget = 0;
float Kp = 5; //resolution control
float Kd = 0.1;//error correction
float Ki = 40;

while (true){
            if (mostRecentTarget != arm1Target)
            {
            error_diff = 0;
            speed = 0;
            error_past = 0;
            errorintg = 0;
            mostRecentTarget = arm1Target;
            error = 0;
            }

current = getMotorEncoder(arm1)/gearRatio;
error = arm1Target - current;
error_diff = (error - error_past)/0.01;
speed = (error*Kp + error_diff*Kd + errorintg*Ki);
error_past=error;
errorintg = (errorintg +  error)*0.01;
motor[arm1]= speed;
}
}

//ARM2
task moveArm2{

float error;
float error_diff;
float speed;
float error_past;
float errorintg;
float mostRecentTarget = 0;
float current = 0;

float Kp = 5; //resolution control
float Kd = 0.1;//error correction
float Ki = 40;

while (true){
            if (mostRecentTarget != arm2Target)
            {
            error = 0;
            error_diff = 0;
            speed = 0;
            error_past = 0;
            errorintg = 0;
            mostRecentTarget = arm2Target;
            }
current = getMotorEncoder(arm2)/gearRatio;
error = arm2Target - current;
error_diff = (error - error_past)/0.01;
speed = (error*Kp + error_diff*Kd + errorintg*Ki);
error_past=error;
```

```
errorintg = (errorintg +  error)*0.01;
motor[arm2]= speed;
}
}

//the arm tips starting coords are 30, 0. 30 is the maximum reach.
//x is along the arms starting axis, y is perpendicular
void goHome(){
	calculateInstructions(L1+L2, 0);
}


float xdest;
float ydest;
void translateCoords(float xcoord, float ycoord, bool scanmod){

	float ymod = 0;
	if (scanmod)
		ymod = 1;


	switch(xcoord){
		case 0: xdest = 12; break;//8//
		case 1: xdest = 19; break;//14
		case 2: xdest = 25; break;//19//
		case 3: xdest = 30; break;//24//
	}

	switch(ycoord){
		case 0: ydest = 10; break;//17//
		case 1: ydest = 4; break;//12//
		case 2: ydest = -2; break;//7//
		case 3: ydest = -7; break;//1//
	}

	if (xcoord == 0 && ycoord == 3)
	{
		xdest = 14;
		ydest = -7;
	}

		if (xcoord == 0 && ycoord == 2)
	{
		xdest = 13;
		ydest = -1;
	}

	if (xcoord == 0 && ycoord == 1)
	{
		xdest = 13;
		ydest = 5;
	}

	if (xcoord == 0 && ycoord == 0)
	{
		xdest = 12;
```

```
        ydest = 11;
}

if (xcoord == 1 && ycoord == 3)
{
        xdest = 20;
        ydest = -6;
}

if (xcoord == 1 && ycoord == 2)
{
        xdest = 20;
        ydest = 0;
}

if (xcoord == 1 && ycoord == 1)
{
        xdest = 18;
        ydest = 5;
}

if (xcoord == 1 && ycoord == 0)
{
        xdest = 18;
        ydest = 11;
}

if (xcoord == 2 && ycoord == 3)
{
        xdest = 25;
        ydest = -8;
}

if (xcoord == 2 && ycoord == 2)
{
        xdest = 25;
        ydest = -1;
}

if (xcoord == 2 && ycoord == 1)
{
        xdest = 24.5;
        ydest = 5;
}

if (xcoord == 2 && ycoord == 0)
{
        xdest = 23;
        ydest = 11;
}

if (xcoord == 3 && ycoord == 3)
{
        xdest = 30;
        ydest = -11;
}
```

```
if (xcoord == 3 && ycoord == 2)
{
		xdest = 31;
		ydest = -2;
}

if (xcoord == 3 && ycoord == 1)
{
		xdest = 30;
		ydest = 4;
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
if (xcoord == 0.5 && ycoord == 0.5)
{
		xdest = 16 - ymod;
		ydest = 7 - ymod*2;
}

if (xcoord == 0.5 && ycoord == 1.5)
{
		xdest = 16;
		ydest = -1 + ymod;
}

if (xcoord == 0.5 && ycoord == 2.5)
{
		xdest = 16;
		ydest = -5 - ymod;
}

if (xcoord == 1.5 && ycoord == 0.5)
{
		xdest = 22 + ymod;
		ydest = 4 + ymod;
}

if (xcoord == 1.5 && ycoord == 1.5)
{
		xdest = 22;
		ydest = 0;
}

if (xcoord == 1.5 && ycoord == 2.5)
{
		xdest = 22;
		ydest = -6 - ymod;
}

if (xcoord == 2.5 && ycoord == 0.5)
{
			xdest = 29;
			ydest = 5 - ymod*7;
		}

		if (xcoord == 2.5 && ycoord == 1.5)
		{
			xdest = 29 - ymod;
			ydest = -2 - ymod*6;
		}

		if (xcoord == 2.5 && ycoord == 2.5)
		{
			xdest = 29 - ymod*1;
			ydest = -9 - ymod*2;
		}

	}

	void Goto(float xcoord, float ycoord, bool scanmod){
		//seperating movement from translation function for other uses
		translateCoords(xcoord, ycoord, scanmod);
		//move to coords
		calculateInstructions(xdest, ydest);
		sleep(1000);
	}

	bool scanForRed(){
		int color = SensorValue[colorSensor];
		if ( color == 5){
			playTone(10, 10);
			return true;
		}
		else
			return false;
	}


	void activatePen(){
		sleep(500);
		setMotorTarget(pen, -200, 10);
		sleep(2000);
	}

	void deactivatePen(){
		sleep(1000);
		setMotorTarget(pen, 0, 100);
		sleep(1000);
	}


	void x00(){

		xdest = 16;
		ydest = 7;

		//first stroke
```

```
        //xdest -= 1;
        ydest -= 1;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest += 4;
        ydest += 5;
        calculateInstructions(xdest, ydest);
        deactivatePen();
        //goHome();

        //second stroke
        ydest -= 8;
        //xdest -= 3;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest -= 6;
        ydest += 10;
        calculateInstructions(xdest, ydest);
        sleep(1000);
        //xdest -= 2;
        //ydest += 4;
        //calculateInstructions(xdest, ydest);
        deactivatePen();
        goHome();
}

void x10(){
        xdest = 22;
        ydest = 4;
        //first stroke
        xdest -= 1;
        ydest -= 0;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest += 4;
        ydest += 5;
        calculateInstructions(xdest, ydest);
        deactivatePen();
        //goHome();

        //second stroke
        ydest -= 9;
        xdest -= 0;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest -= 3;
        ydest += 13;
        calculateInstructions(xdest, ydest);
        sleep(1000);
        //xdest -= 2;
        //ydest += 4;
        //calculateInstructions(xdest, ydest);
        deactivatePen();
        goHome();
}
void x20(){
        xdest = 29;
        ydest = 5;

        //first stroke
        xdest -= 2;
        ydest -= 0;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest += 3;
        ydest += 5;
        calculateInstructions(xdest, ydest);
        deactivatePen();
        //goHome();

        //second stroke
        ydest -= 9;
        xdest += 1;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest -= 5;
        ydest += 13;
        calculateInstructions(xdest, ydest);
        sleep(1000);
        //xdest -= 2;
        //ydest += 4;
        //calculateInstructions(xdest, ydest);
        deactivatePen();
        goHome();
}
void x01(){
        xdest = 16;
        ydest = -1;

        //first stroke
        xdest -= 0;
        ydest -= 0;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest += 3;
        ydest += 7;
        calculateInstructions(xdest, ydest);
        deactivatePen();
        //goHome();

        //second stroke
        ydest -= 9;
        //xdest -= 3;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest -= 3;
        ydest += 7;
        calculateInstructions(xdest, ydest);
        sleep(1000);
        //xdest -= 2;
        //ydest += 4;
        //calculateInstructions(xdest, ydest);
        deactivatePen();
        goHome();
}
void x11(){
        xdest = 22;
        ydest = 0;

        //first stroke
        xdest -= 0;
        ydest -= 1;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest += 4;
        ydest += 6;
        calculateInstructions(xdest, ydest);
        deactivatePen();
        //goHome();

        //second stroke
        ydest -= 11;
        xdest += 1;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest -= 3;
        ydest += 13;
        calculateInstructions(xdest, ydest);
        sleep(1000);
        //xdest -= 2;
        //ydest += 4;
        //calculateInstructions(xdest, ydest);
        deactivatePen();
        goHome();
}
void x21(){
        xdest = 29;
        ydest = -2;

        //first stroke
        xdest -= 1;
        ydest -= 0;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest += 3;
        ydest += 7;
        calculateInstructions(xdest, ydest);
        deactivatePen();
        //goHome();

        //second stroke
        ydest -= 11;
        //xdest -= 3;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest -= 3;
        ydest += 13;
        calculateInstructions(xdest, ydest);
        sleep(1000);
        //xdest -= 2;
        //ydest += 4;
        //calculateInstructions(xdest, ydest);
        deactivatePen();
        goHome();
}

void x02(){
        xdest = 16;
        ydest = -5;

        //first stroke
        xdest += 1;
        ydest -= 1;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest += 4;
        ydest += 7;
        calculateInstructions(xdest, ydest);
        deactivatePen();
        //goHome();

        //second stroke
        ydest -= 11;
        xdest -= 1;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest -= 1;
        ydest += 10;
        calculateInstructions(xdest, ydest);
        sleep(1000);
        //xdest -= 2;
        //ydest += 4;
        //calculateInstructions(xdest, ydest);
        deactivatePen();
        goHome();
}
void x12(){
        xdest = 22;
        ydest = -6;

        //first stroke
        xdest += 1;
        ydest -= 0;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest += 4;
        ydest += 8;
        calculateInstructions(xdest, ydest);
        deactivatePen();
        //goHome();

        //second stroke
        ydest -= 13;
        xdest -= 1;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest -= 2;
        ydest += 9;
        calculateInstructions(xdest, ydest);
        sleep(1000);
        //xdest -= 2;
        //ydest += 4;
        //calculateInstructions(xdest, ydest);
        deactivatePen();
        goHome();
}
void x22(){
        xdest = 29;
        ydest = -9;

        //first stroke
        xdest -= 2;
        ydest -= 1;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest += 4;
        ydest += 5;
        calculateInstructions(xdest, ydest);
        deactivatePen();
        //goHome();

        //second stroke
        ydest = -15;
        xdest = 29;
        calculateInstructions(xdest, ydest);
        activatePen();
        xdest += 1;
        ydest += 13;
        calculateInstructions(xdest, ydest);
        sleep(1000);
        //xdest -= 2;
        //ydest += 4;
        //calculateInstructions(xdest, ydest);
        deactivatePen();
        goHome();
}
void drawXat(float x, float y){
        goHome();
        sleep(1000);
        if(x == 0 && y == 0)
                x00();
        else if(x == 1 && y == 0)
                x10();
        else if(x == 2 && y == 0)
                x20();
        else if(x == 0 && y == 1)
                x01();
        else if(x == 1 && y == 1)
                x11();
        else if(x == 2 && y == 1)
```

```
                x21();
        else if(x == 0 && y == 2)
                x02();
        else if(x == 1 && y == 2)
                x12();
        else if(x == 2 && y == 2)
                x22();
}
```