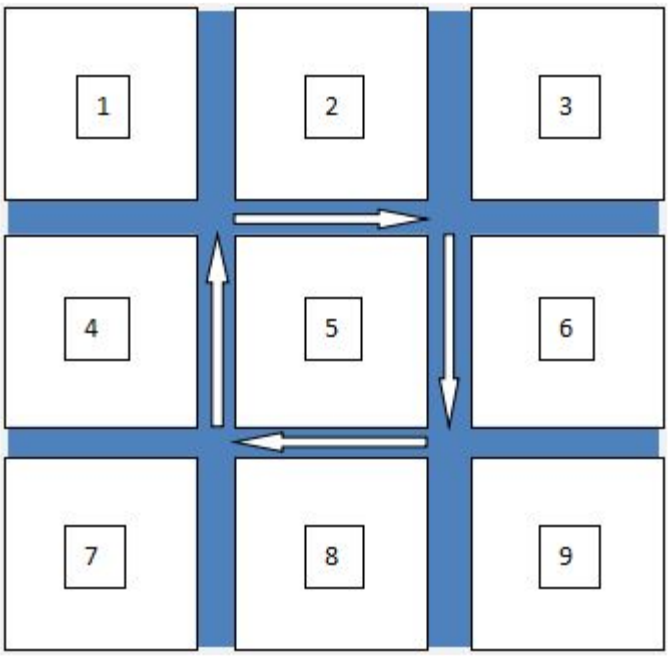
Project 4 Tic-Tac-Toe report

Progress Report

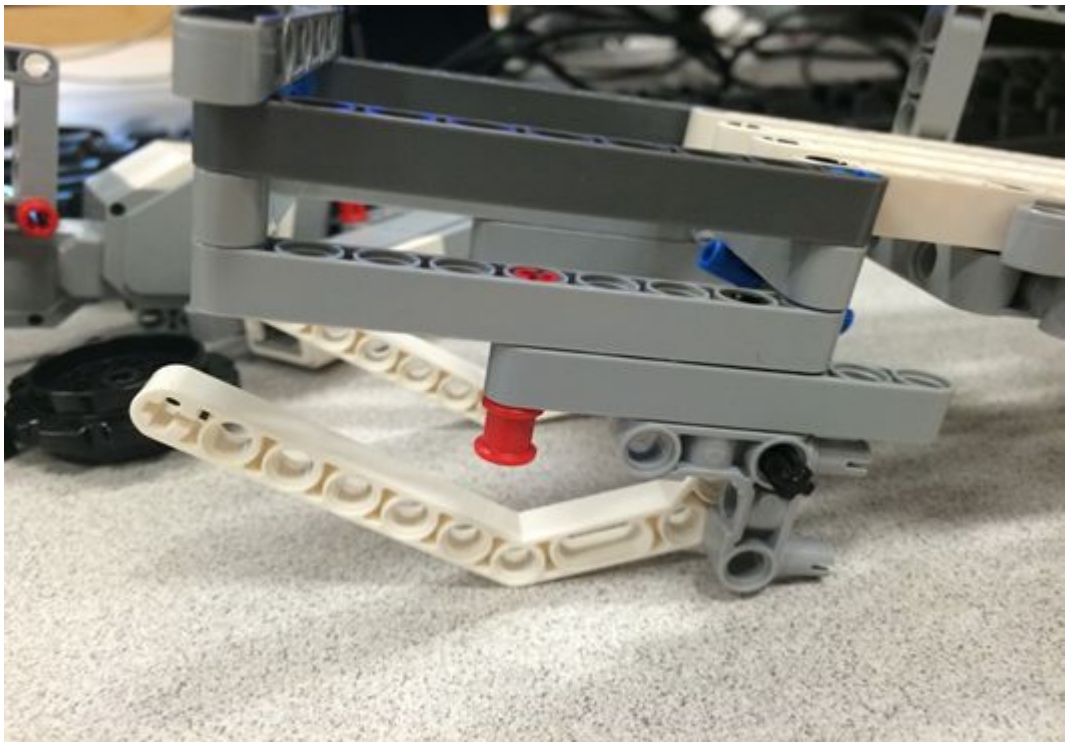
For this project, the target is winning the Tic-Tac-Toe, and the robot cannot hit an already placed ball. So we build a robot which has a small base plate. The robot should do everything on the blue line. We design a algorithm for the game, the robot can calculate the winning rate of every position which we can set, and the robot should choose the highest winning rate to set the ball.

Mechanical Design

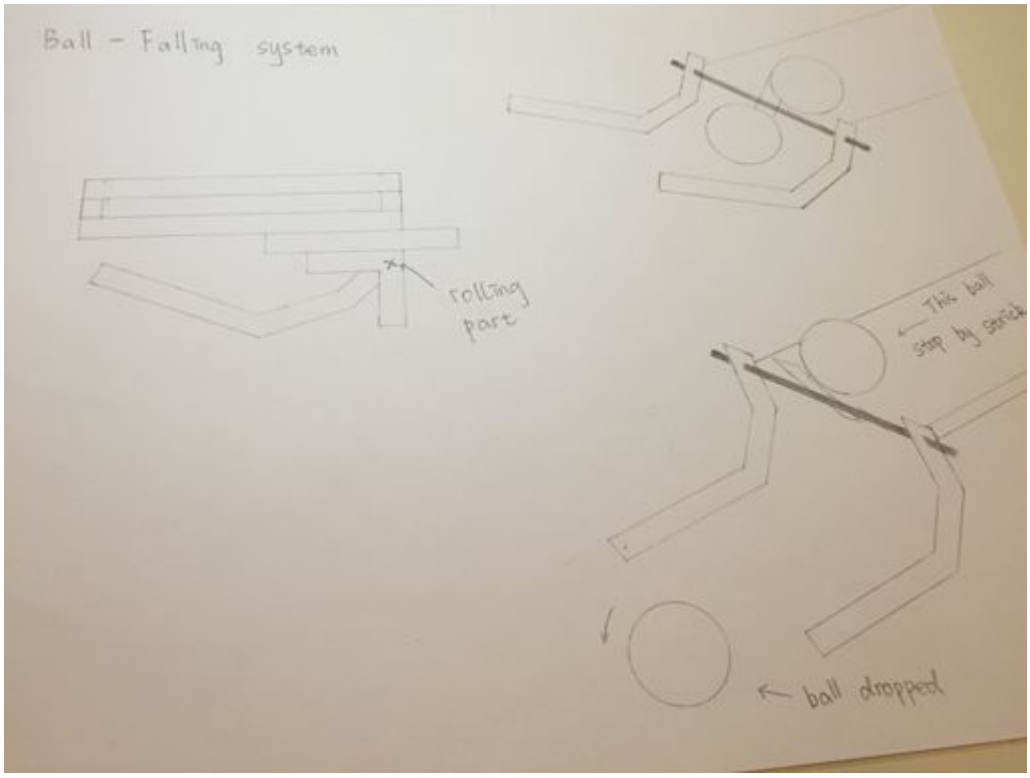


This is the place we play Tic-Tac-Toe, and we give a number for each pixel; our idea is to build a robot which is running by following blue line.

Our robot is not complicated. We use two small wheels for building a small base plate to avoid robot hitting an already placed ball, and use a steel ball at the back to reduce friction force. Steel ball is bigger than wheel, which can make robot lean a little bit to let it drop ball easily. In the whole building part, we spend most time to think that what we could do to let one ball fall at each time.



Finally, we build ball-falling system like this picture. We use a motor to control it open or close, and use a stick connect with rolling part; if it is close, it can store one ball, and if it is open, the ball that it store in the system will fall on pixel, and stick can let rest balls will stay in robot; after it finish dropping ball, the system close again, and another ball will go into system.



Then we put Ultrasonic sensor under the ball-falling system to let it detective ball; putting a touch sensor to let human player enter his choice and telling robot drop ball

after human player's move.

Because our robot should always turn 45 degrees, the weakness is encoder error could make angle become different and let robot cannot follow blue line all the time.

Software Design

For my code, I have two parts, one part is design for the robot moving, such as searching balls, setting balls and fixing the position; the other part is design for the tactics, as I said, the robot can calculate the winning rate, for this process, we use DFS(depth first search) algorithm to finish. DFS is a kind of search which uses recursion and back-track algorithm, it can count all cases, like enumeration. That means the robot can simulate the process of all cases of remaining game, and then the robot can know if I set the ball someone position, the number of winning cases and the number of failure cases. So the robot can depend these numbers to determine where it should set.

Function description

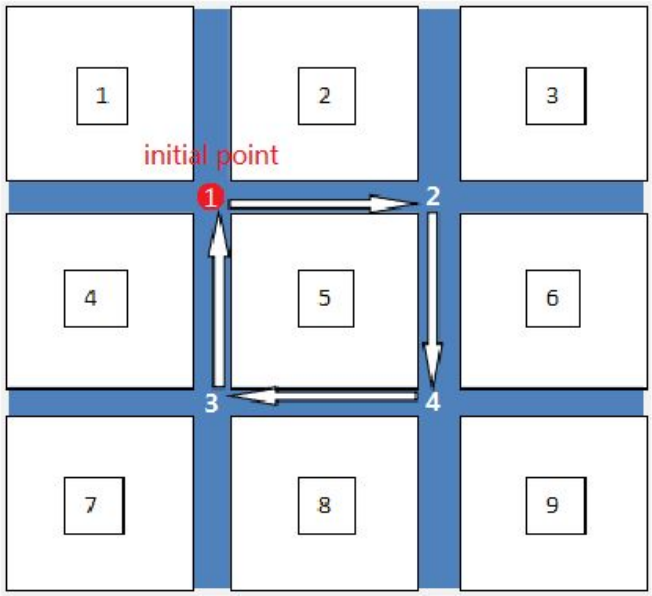
```
void Setball(int encoder)//this function can control droping balls.
void Turn45()//This function can make the robot turn 45 degrees
void Turn_45()//This function can make the robot turn -45 degrees
void Turn(int times)//This function can tell robot the number of Truning 45 degrees
void Move_position()//let robot to move the length of one unit square
void Return(int times)//let robot return to the starting position
void fix()//This function can fix the position and direction of robot
int Judge()//This function can judge who wins the game or no one wins until now
void Dfs(int depth)//This is the process of simulate the remaing games.
int Think()//This function can calculate and determine where it should set the ball
void reset_degree(int p)//Let the robot to be the correct direction
void Search()//Search the ball which the player just set
void Move_to(int x,int y)//Let the car move to the position where nears the (x,y)
```

Setting of map

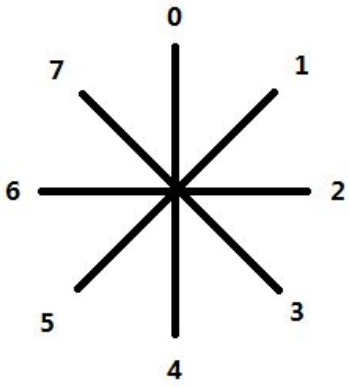
For making our code simple and convenient, we have some settings of the chess board, the following map is easy to explain the chessboard.

We assume the top left grid is No.1 grid, and the bottom right is No.9 according to priority.

We also number 4 points on the blue line, because the robot needs to move to the 4 points to set the balls. For the correct direction of every point are different, point 1 is 0, point 2 is 2, point 3 is 4, point 4 is 6.



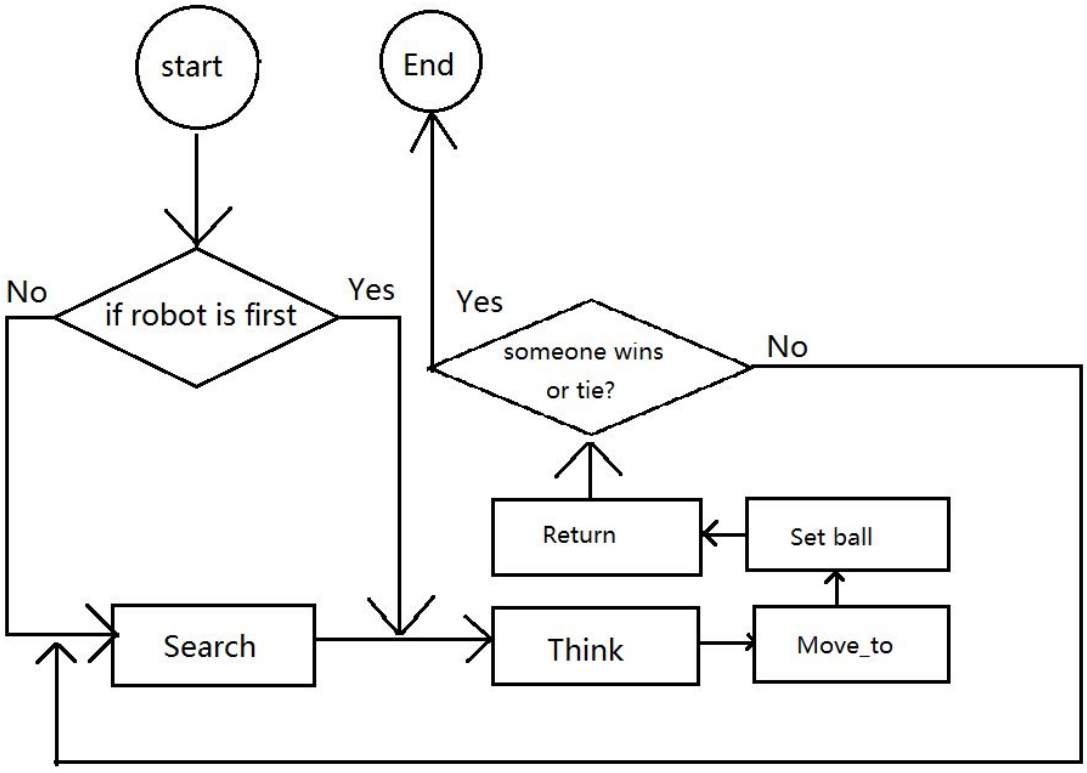
The setting of map



The direction of robot

Flowchart

For the moving part of my code, the following diagram is the flowchart:



Function parameters

The following two diagram are shows all global variables in code:

```
int mycolor,first;
//if robot is blue, mycolor is 1, else mycolor is 0.
//if robot is first, first is 1, else first is 0
int current_map[5][5];
//this 2-D array saves the current chess board data,
//if current_map[i][j]==1, that means position (i,j) has robot's ball.
//if current_map[i][j]==2, that means position (i,j) has player's ball.
//if there is no balls at position(i,j), current_map[i][j]==0;
int step;
//step represents how many balls the robot has currently.
int current_position,current_degree;
//current_position saves the current position of the robot(1 to 4) in the map.
//current_degree saves the current direction of the robot(1 to 8).
//You can see details of the map and direction in report.
int belong[4][4];
//we need to set every grid belong to which point manually.
//if we do that, we can know someone grid belong to which point easily,
//we do not need to write some useless code like
//if (x==1  &&  y==2) then
//position=1
//that can make my code shorter.
int degree[4][4],position_degree[6][6];
//we also need to set every grid the direction relative to the postion whose
//the grid belong to.
//that is the action of degree array.
//And the postion_degree array's action is saving the direction of point
//ralative to the others points.
//The 2 arrays and 'belong' array are very useful for my code.
//they let my code reduce many judgement statement.
int dx[10],dy[10];
```

```
//dx means different value x,
//dy means different value y.
//They are use for the Judge function,
//they save eight directions for 2-D,
//for example,if point 5 is (2,2),so if we add dx and dy for this point,
//dx[1]==-1, dy[1]==-1, so the top left point relative to point 5 is (2+dx[1],2+dy[1])
//it is point 1(1,1)
//we can use one loop to check 8 direction of one point.
int win,lose;
//win variable saves the number of win during the simulating(Dfs function)
//lose variable saves the number of failure during the simulating(Dfs function)
float answer;
//answer variable is use in think function,
//it saves the winning rate during Dfs function(simulating process)
//if answer==1, we can say the robot set the ball here, it must win the game,
//if answer==0, we can say the robot set the ball here, it must lose the game,
//answer value is between [0,1], the higher it changes, the better it will be.
```

The following diagrams show the meaning of some parameters and some local variables:

```
//if encoder is 1,the control is closed,
//if encoder is -1, the control is open.
void Setball(int encoder)//this function can control droping balls.
{
    if (encoder>0)
            motor[motorB]=10;
    else
            motor[motorB]=-10;

    for (int i=1;i<=100000;i++)
    {}
    motor[motorB]=0;
}
```

```
//times is a variables between [-8,8],
//if times > 0, the robot should turn clockwise 'times'(variable) times,
//else turn counterclockwise times.
void Turn(int times)//This function can tell robot the number of Truning 45 degrees
{
    if (times>0)
    {
        for (int i=0;i<times;i++)
        {
            Turn45();
            current_degree++;
        }
    }
    else
    {
        for (int i=0;i>times;i--)
        {
            Turn_45();
            current_degree--;
        }
    }
}
```

```
//the times means how many times it should turn 90 degrees and go forward.
void Return(int times)//let robot return to the starting position
{
        while (times--)
        {
            nMotorEncoder[motorA]=0;
            while (nMotorEncoder[motorA]>-650)
            {
                motor[motorA]=-10;
                motor[motorD]=-10;
            }
            motor[motorA]=0;
            motor[motorD]=0;
            Turn(-2);
        }
        current_position=1;
        //reset the current_position variable.
}
```

```
// the depth means how many steps the robot simulates,
// if depth is odd, that means the robot should set the ball
// else the depth is even, that means player should set the ball.
void Dfs(int depth)//This is the process of simulate the remaing games.
{
    if (answer==1  ||  answer==-100) return;
    int temp=Judge();
    //Simulation until now, someone wins
    //if the robot only sets one ball and wins, the robot must win the game with one step,so the
    //winning rate is 1.
    //if the player only sets one ball and wins, so the robot must lose the game with one step,
    //it cannot set the ball where it supposed, so the winning rate is very very low, so I set
    //the winning rate is -100
    //if the depth > 2, we cannot say we must win for setting the ball here, so we need to count
    //the cases of winning and failure, and calculate the winning rate at the end.
    if (temp)
    {
        if (depth==1  &&  temp==1) answer=1;
        if (depth==2  &&  temp==2) answer=-100;
        if (temp==1) win++;
        if (temp==2) lose++;
        return;
    }

    for (int i=1;i<=3;i++)
        for (int j=1;j<=3;j++)
            if (!current_map[i][j])
            {
                current_map[i][j]=1+(depth % 2);
                Dfs(depth+1);
                current_map[i][j]=0;
                //this can optimize my code, the answer represents the winning rate,
                //if answer has value, so it can win or lose with one step.
                if (answer>=0)
                    return;
            }
}
int Think()//This function can calculate and determine where it should set the ball
{
    if (step==5) return 5;
    //if the robot first, setting the ball in the center is the highest winning rate.
    if (first  &&  step==4)
        for (int i=1;i<=9;i+=2)
            if (!current_map[(i / 3)+1][i % 3]) return i;
    //if the robot first, the second step is set the ball at the corner.
    if (!first  &&  step==4)
    {
        if (!current_map[2][2]) return 5;
        if (current_map[2][2]) return 1;
    }
    //if the robot is not first, the best choice for first step is corner.
    int copy_map[5][5];
    int max_x=-1,max_y=-1;
    float max_answer;
    //save the max winning rate and position.
    max_answer=-10000;
    for (int i=1;i<=3;i++)
        for (int j=1;j<=3;j++)
            copy_map[i][j]=current_map[i][j];
    //because I have some optimization in Dfs, and I changed the current_map but
    //cannot recover the current_map, so I need a copy current_map array to recover
    //the current_map array.
```

```
    for (int i=1;i<=3;i++)
        for (int j=1;j<=3;j++)
        {
            if (current_map[i][j]) continue;
            answer=-1;
            win=0;
            lose=0;
            current_map[i][j]=1;
            Dfs(1);
            current_map[i][j]=0;
            if (answer>max_answer)
            {
                max_answer=answer;
                max_x=i;
                max_y=j;
            }
            for (int k=1;k<=3;k++)
                for (int l=1;l<=3;l++)
                    current_map[k][l]=copy_map[k][l];
        }
    return (max_x-1)*3+max_y;//convert 2-D coordinate into 1-D coordinate
}

void Search()//Search the ball which the player just set
{
    //the orderx and ordery arrays represent the robot needs to search
    //every grid with this order.
    //if someone grid has ball before, the robot does not need to search this grid,
    //it will skip this position.
```