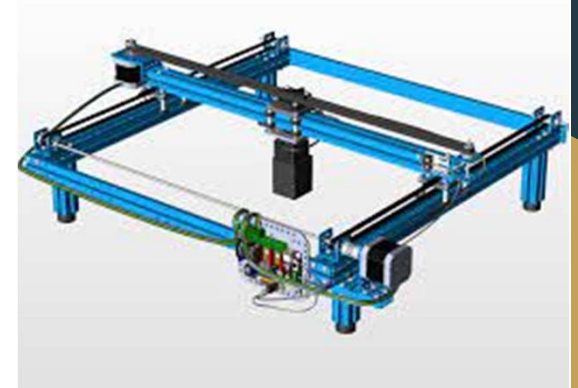
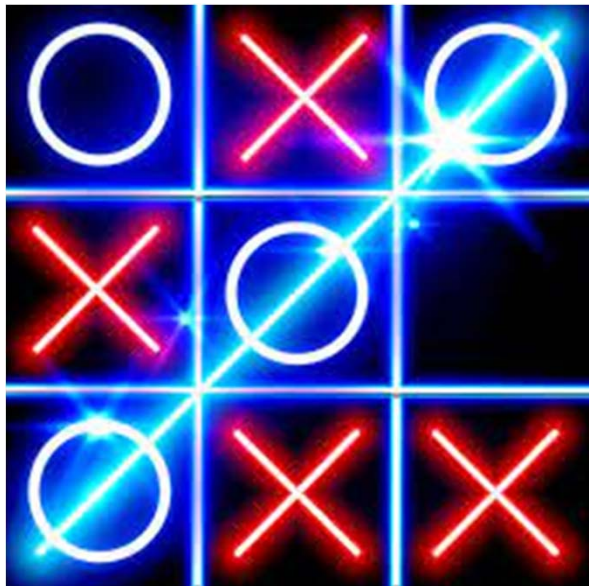


# Terminator

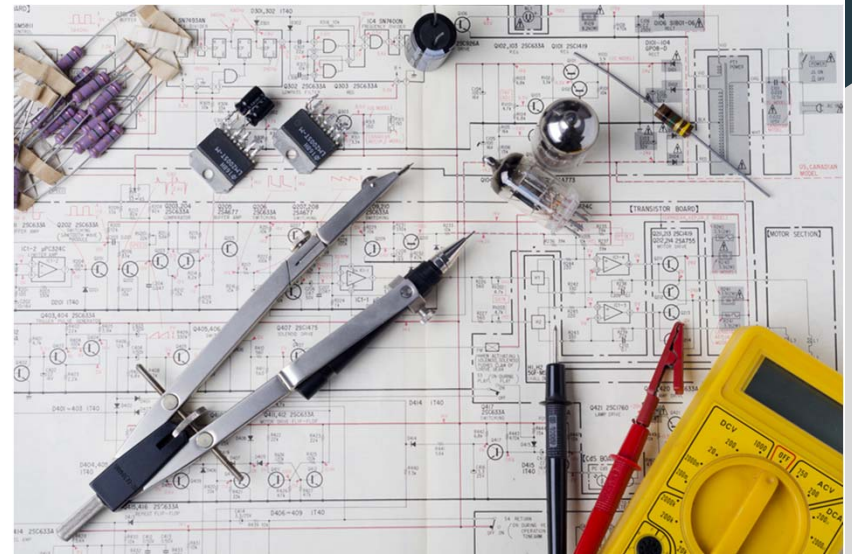


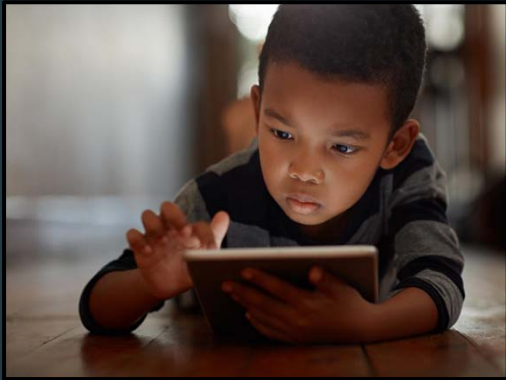
Omaretsoguwa Atsagbede  
Oshione Adams  
Bella Kuete  
Amiyah Brown

Faculty Advisor: Dr. Charles Kim  
Date: 4/14/2023

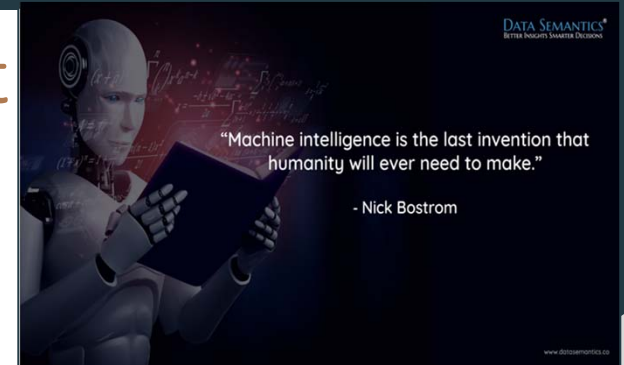
# Background

Engineering design is a process of using mathematical and engineering tools to create a plan to solve a real world problem. In our project, we decided to tackle children's productivity. We looked at a series of models and broke our design process into multiple steps to see what fit our final goal. This design solution is the culmination of those different processes from the problem statement and specifying different design models and their constraints to our final model.



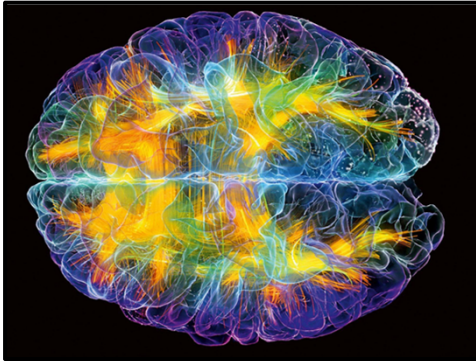


## Problem Statement

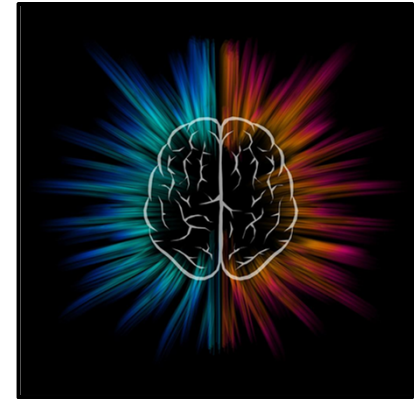


A.I is the need of today's generation are that children are always on their phone nor don't have someone to play with, which lead us to create a robot that plays tic-tac-toe, so that they can be productive and not be on their phones the whole day.





# Problem Definition



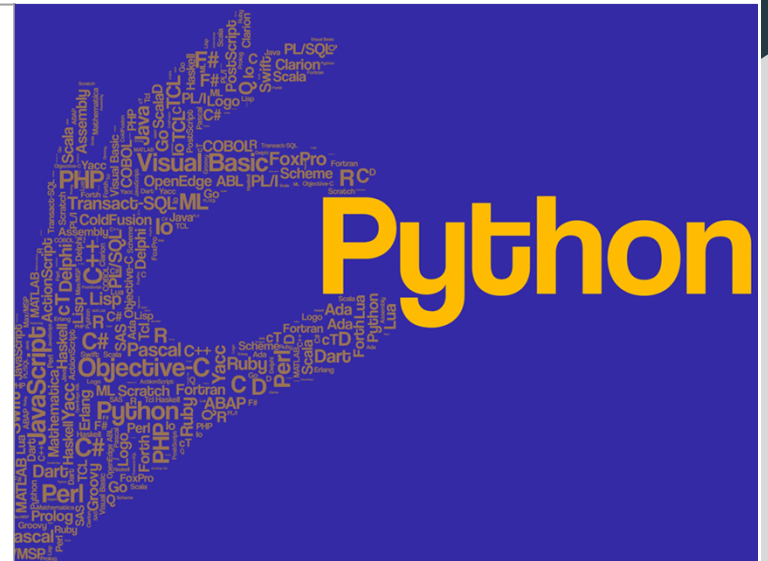
## Design Requirements

Items	Quantity
AC Input Voltage	100 - 240V
Weight of the final product	The net weight is 3.87 kg and the gross weight is 4.475 kg.
Python Google Collab	No in-built libraries(brute force)
Arduino, Laserbot	1 arduino and 1 laserbot will be used
Motors	2 Motors will be used
Online documentation of Tic-Tac-Toe Strategy	Varies documents that were used
Physical Dimensions and Maximum Working Speed	535mmx637mmx184mm are the dimensions and the speed would be 200mm/s.

# Constraint of Standards and Regulations to comply

**Compliance (Rules, Regulations, and Standards)**

Because we are using the python language, we are in compliance with the IEEE python coding style.



# Constraints of Society, Culture, and Environment

## **Environmental Constraints**

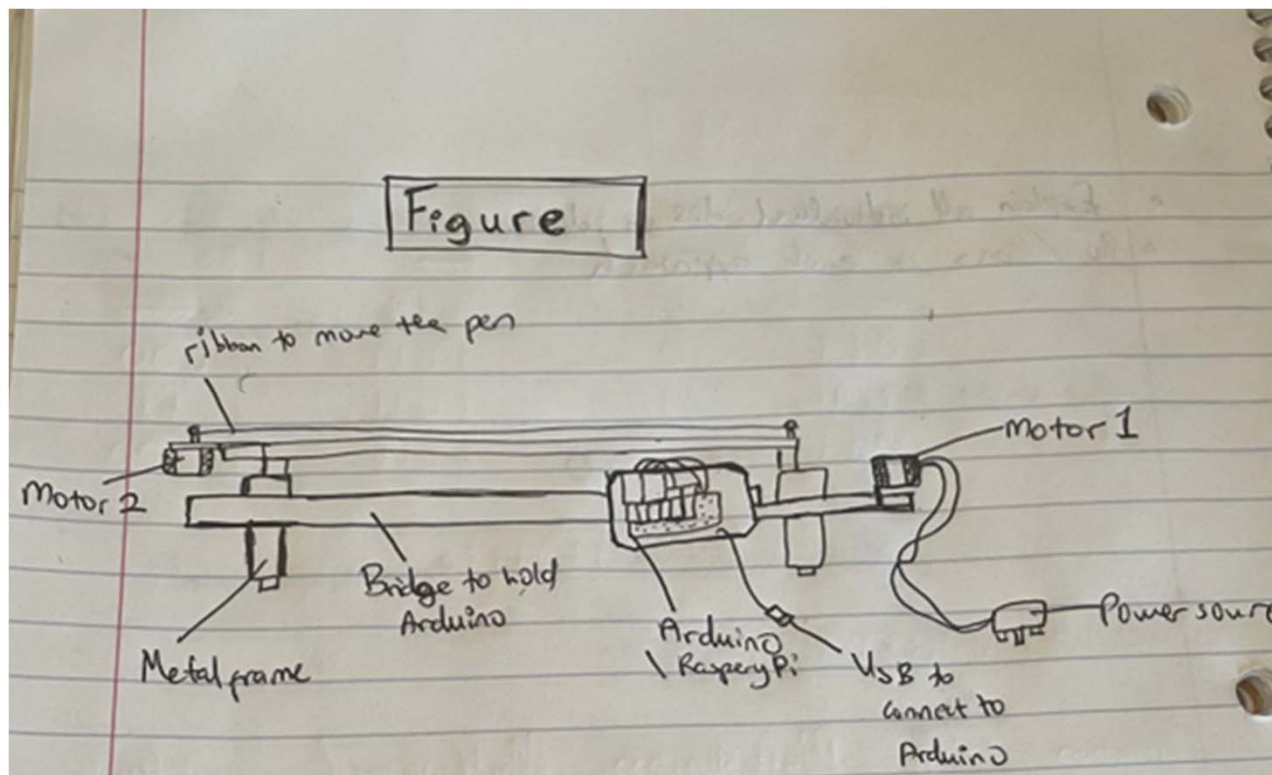
The dimensions of the Laserbot are still somewhat substantial, limiting its portability and preventing it from being as compact as desired

Previously: Invisible Laser Radiation, the Max Power would be 1600 mW and the Wavelength will be 445nm.

## **Socio-Cultural Constraints**

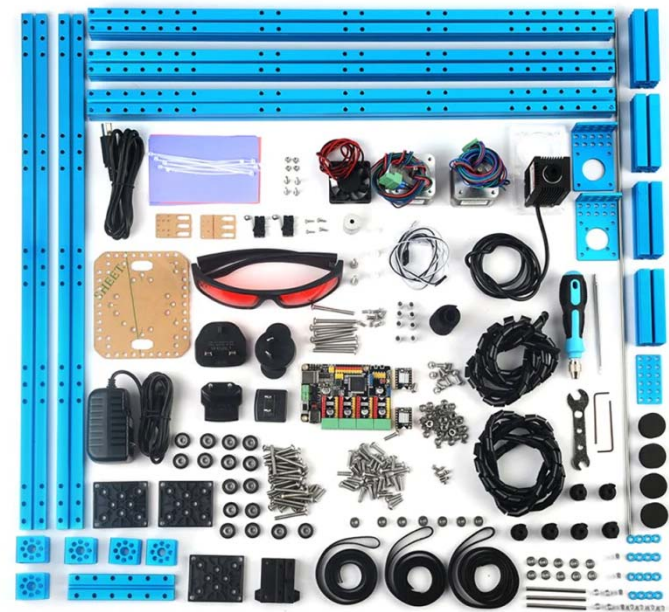
Replace the equipment producing the laser with a pencil to draw the tic-tac-toe board, which can be used to correct the mistakes.

# Schematics of the Top Solution Design



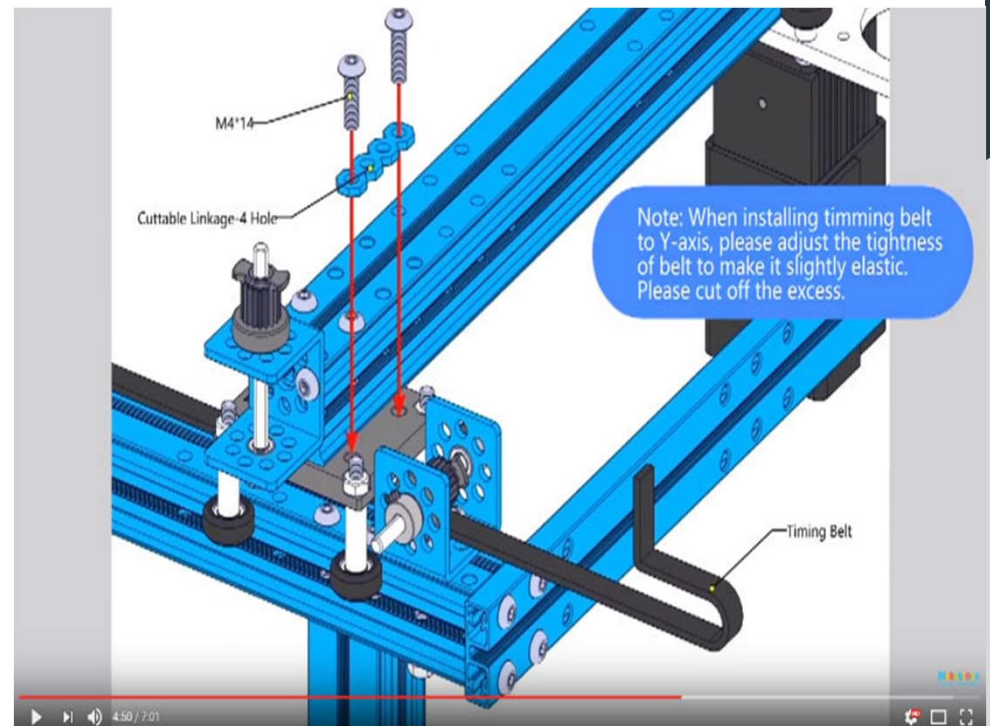
# Explanation of the Final Solution Design

- The device consists of two motors to move the pen on the x plane and y plane.
- Has a ribbon, which the pen holder moves on.
- Has a power source to ensure the device moves.
- It consists of a Mega Pi board which is compatible with Arduino and Raspberry Pi.
- It has four bridges made out of steel which hold the movable components and Mega Pi board. It has 4 metallic frames which keeps the device standing
- It contains a usb port to connect the Mega Pi board to a pc to run the code for the device

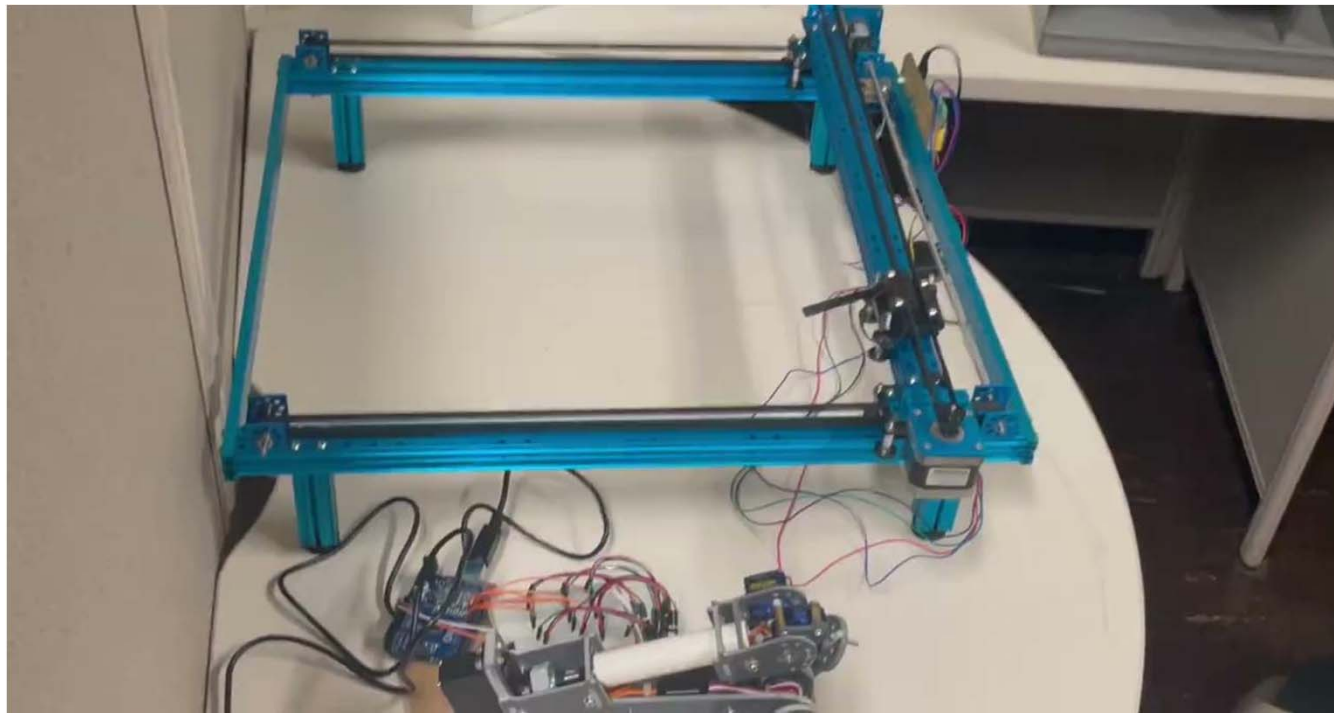


## Implementation Process

1. Assemble the frame
2. Install the motors
3. Set up the ribbon system
4. Mount the Mega Pi board
5. Connect the motors to the board
6. Power the device.
7. Attach the pen holder.
8. Implement USB connectivity.
9. Develop and upload the code.
10. Test and refine.



# Video Presentation



# SPRINT #1 [01-30-23 to 02-10-23]

Main Focus → The software mainly that the CPU, and player movements work smoothly on a 10x10 game board.

## Planned Tasks:

- Week 1:
  - ❖ Research on Tic Tac Toe boards, see a 10x10 version.
  - ❖ Create the game board design.
  - ❖ Find requirements for winning, losing and drawing conditions.
- Week 2:
  - ❖ Finalize the movements of each piece works on the tic tac toe board (software wise).
  - ❖ Work in the movements by the CPU ensuring it makes countering moves.
  - ❖ Ensure board displays at the right positions X and O movements.

```
      0  1  2
    * * * * * *
0 *   *   *   *
  * * * * * *
1 *   *   *   *
  * * * * * *
2 *   * 0 * X *
  * * * * * *
Player 1
Input the x location: 
```

# SPRINT #1 [01-30-23 to 02-10-23]

Highlights:

## What went well:

- Making the codes for the player 1 vs player 2 tests
- Making the codes for player 1 vs CPU (main code to be used)

Lowlights:

## Which tasks were not completed?

- Completing the moves for the CPU to counter the moves properly
- Using Machine Learning for CPU to move on it's own

## What was the issue in the unsuccessful tasks?

- Not enough time to teach the CPU how to block moves properly

### Player1 vs Player2

```
Input the x location: 5
Input the y location: 5
  0  1  2  3  4  5  6  7  8  9
  *  *  *  *  *  *  *  *  *  *
0 *  *  *  *  *  *  *  *  *  *
  *  *  *  *  *  *  *  *  *  *
1 *  *  *  *  *  *  *  *  *  *
  *  *  *  *  *  *  *  *  *  *
2 *  *  *  *  *  *  *  *  *  *
  *  *  *  *  *  *  *  *  *  *
3 *  *  *  *  *  *  *  *  *  *
  *  *  *  *  *  *  *  *  *  *
4 *  *  *  *  *  *  *  *  *  *
  *  *  *  *  *  *  *  *  *  *
5 *  *  *  *  *  X  *  *  *  *
  *  *  *  *  *  *  *  *  *  *
6 *  *  *  *  *  *  *  *  *  *
  *  *  *  *  *  *  *  *  *  *
7 *  *  *  *  *  *  *  *  *  *
  *  *  *  *  *  *  *  *  *  *
8 *  *  *  *  *  *  *  *  *  *
  *  *  *  *  *  *  *  *  *  *
9 *  *  *  *  *  *  *  *  *  *
```



# SPRINT #1 [01-30-23 to 02-10-23]

10 x 10 Board Player 1 vs CPU  
Showcasing the movements of 'X' and  
'O' elements

```
  0 1 2 3 4 5 6 7 8 9
* * * * *
0 * * * * *
* * * * *
1 * * * * *
* * * * *
2 * * * * X * * * *
* * * * *
3 * * * 0 * * * 0 * * *
* * * * *
4 * * * * * X * * *
* * * * *
5 * * * * *
* * * * *
6 * * * * *
* * * * *
7 * * * * *
* * * * *
8 * * * * *
* * * * *
9 * * * * *
* * * * *
Player 1
Input the x location: 
```

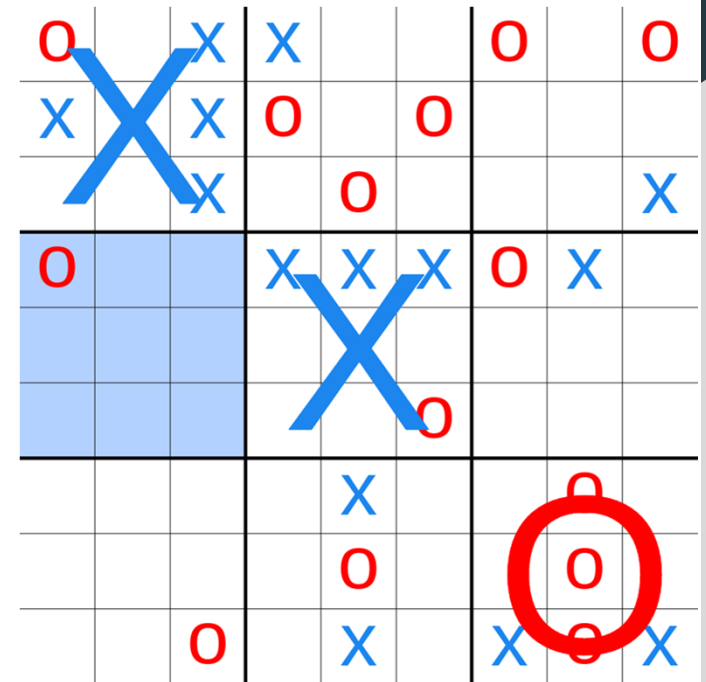
# SPRINT #1 [01-30-23 to 02-10-23]

## How to solve the issues

- Researching more and fully understanding those techniques to block the player 1 moves
- Running and testing more codes that teach the CPU the techniques to block the player 1 moves

## How to reduce failure risk

- Testing the codes more often
- Testing the code using very unique and different test cases



## SPRINT #2 [02-13-23 to 02-24-23]

Main Focus → Ensure that all the conditions in the code (winning, losing, tie) work properly.

### Planned Tasks:

- Week 1:
  - ❖ Work on the winning condition
  - ❖ Focusing on rows, columns, right & left diagonal
- Week 2:
  - ❖ Work on losing conditions
  - ❖ Work on the draw conditions (If not win || lose and board is full then it's a draw)

```
def checkWin(tttboard,x,y,tar,ch):  
    if checkrows(tttboard,x,y,tar,ch):  
        return True  
    elif checkcolumns(tttboard,x,y,tar,ch):  
        return True  
    elif checkrightdiag(tttboard,x,y,tar,ch):  
        return True  
    elif checkleftdiag(tttboard,x,y,tar,ch):  
        return True  
    else:  
        return False
```

## SPRINT #2 [02-13-23 to 02-24-23]

Highlights:

**What went well:**

- Winning condition works
- Losing condition works
- Draw condition works

Lowlights:

**Which tasks were not completed?**

- CPU function completion

**What was the issue in the unsuccessful tasks?**

- The CPU moves just not as smartly as we want it to

```
if currPlayer:
    tttboard[x][y] = "X"
    if checkWin(tttboard, x, y, thresh, "X"): # check if X has won
        drawBoard(tttboard)
        won = "X wins" # won is no longer None. While loop breaks
        continue
else:
    tttboard[x][y] = "O"
    if checkWin(tttboard, x, y, thresh, "O"): # check if Y has won
        drawBoard(tttboard)
        won = "O wins" # won is no longer None. While loop breaks
        continue

# if nobody has won
drawBoard(tttboard) # show the updated board
plays += 1 # increase the number of those who have played
currPlayer = not currPlayer #change the player to the CPU

# if people have played n^2 times, the whole board is filled.
# it's a draw
if plays == n**2:
    won = "DRAW"
```

## SPRINT #2 [02-13-23 to 02-24-23]

Increment 2: Example of “X” player winning, showing how the winning condition works.

```
  0 1 2 3 4 5 6 7 8 9
*****
0 * 0 * 0 * 0 * 0 * * * * *
*****
1 * * X * * * * * * * *
*****
2 * * * X * * * * * * *
*****
3 * * * * X * * * * * *
*****
4 * * * * * X * * * * *
*****
5 * * * * * * X * * * *
*****
6 * * * * * * * * * * *
*****
7 * * * * * * * * * * *
*****
8 * * * * * * * * * * *
*****
9 * * * * * * * * * * *
*****
X wins
```

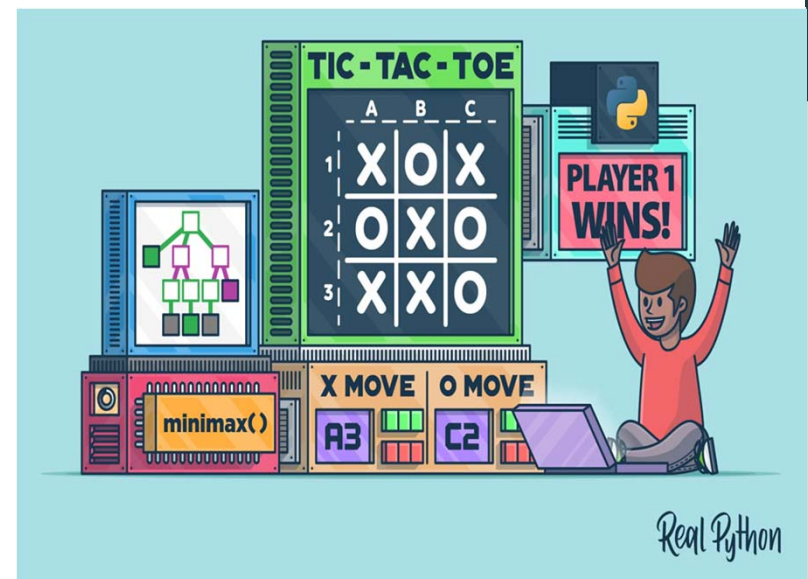
## SPRINT #2 [02-13-23 to 02-24-23]

### How to solve the issues

- Researching more to see how to train the CPU function (*Minimax Algorithm*).
- Running the code numerous times with different test cases to pinpoint errors in the code.

### How to reduce failure risk

- Testing the codes at increments to ensure each part works (*Winning conditions, Losing conditions etc*).
- Trying the code across different IDE's just as a precaution.
- Multiple testing of the code as a whole at the end.

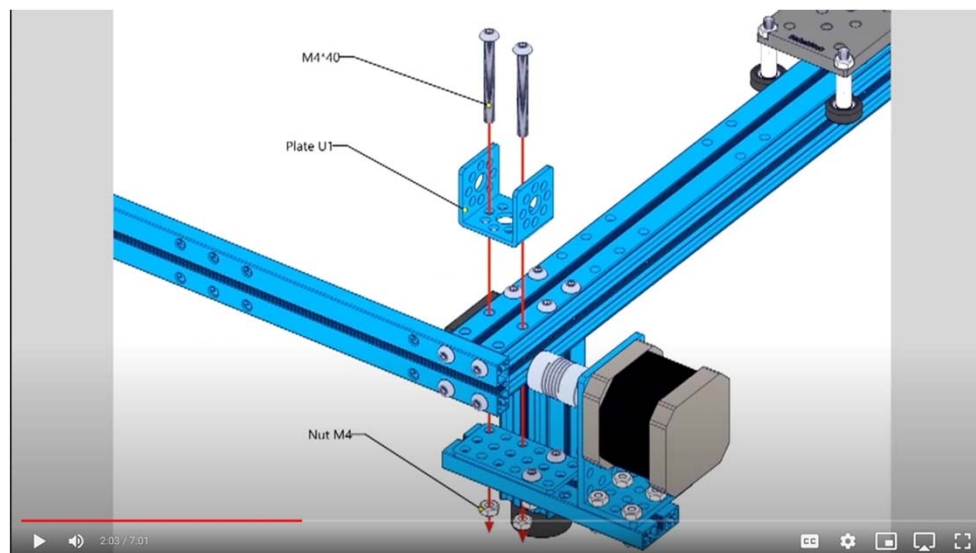


## SPRINT #3 [02-27-23 to 03-17-23]

### Piece:

Main Focus → Assembling the hardware using tutorial videos about the laser bot.

#### Tutorial Video



## SPRINT #3 [02-27-23 to 03-17-23]

Highlights:

### What went well:

- Got the motors to work fine
- All the other hardware components worked fine
- The CPU function from Sprint #2 works properly

Lowlights:

### Which tasks were not completed?

- Replacing the laserbot lasers with a pen holder

### What was the issue in the unsuccessful tasks?

- Not yet found a pen holder compatible with the laserbot

```
def smartCpuNextMove(tttboard, thresh):  
    n = len(tttboard)  
    empty_cells = []  
    immediate_threats = []  
  
    for i in range(n):
```

Function for CPU Moves (*Smart Moves Only*)

# SPRINT #3 [02-27-23 to 03-17-23]

## Increment (*From Sprint #2*)

10 x 10 Board Player 1 vs CPU

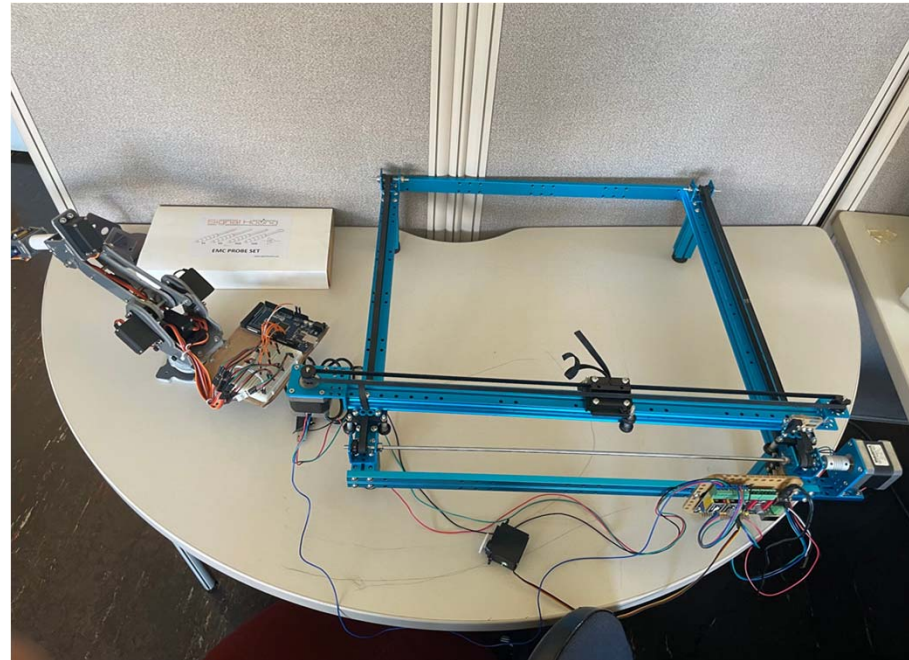
```
CPU
  0  1  2  3  4  5  6  7  8  9
* * * * *
0 * 0 * * * * * * * 0 * * *
* * * * *
1 * * X * * * O * * * * *
* * * * *
2 * * * X * * * * * * * *
* * * * *
3 * * * * X * * * * * * *
* * * * *
4 * * * * * X * * * * * *
* * * * *
5 * * * * * * O * * * * *
* * * * *
6 * * * * * * * * * * * *
* * * * *
7 * * * * * * * * * * * *
* * * * *
8 * * * * * * * * * * * *
* * * * *
9 * * * * * * * * * * * *
* * * * *
```

Player 1

Input the x location:

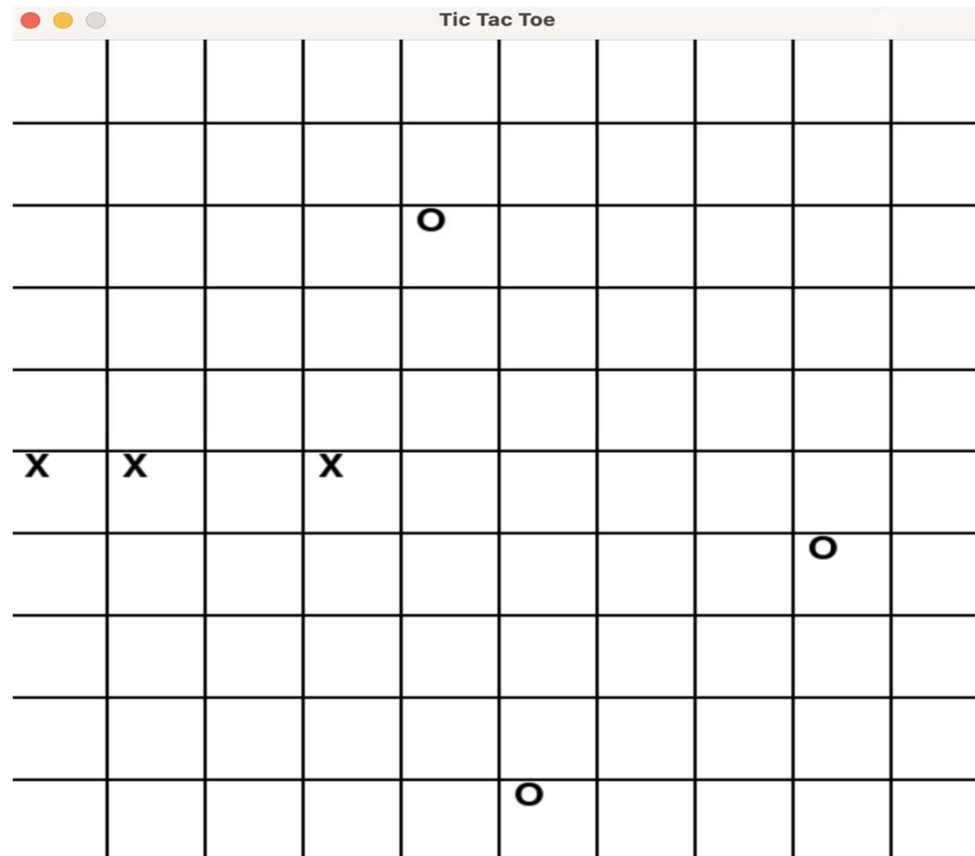
## Increment (Sprint #3)

Picture of the laserbot from LKD



# SPRINT #4 [03-20-23 to 04-11-23]

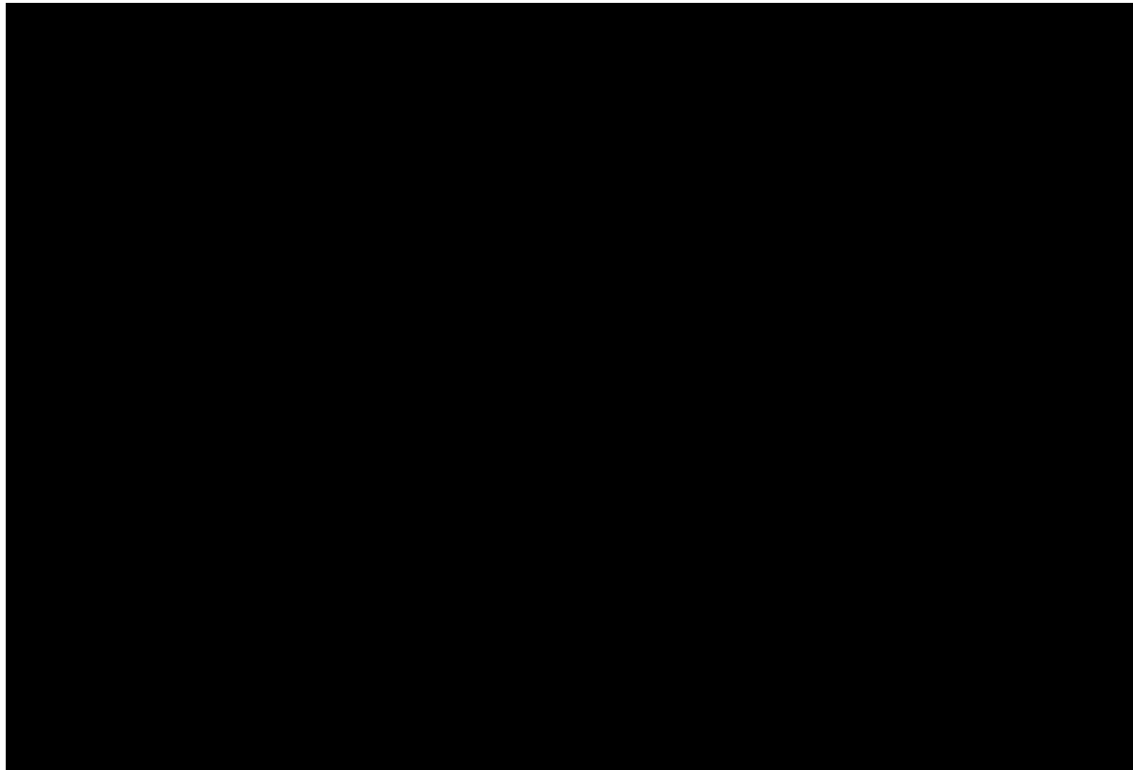
Newer Design



[Code Using Pygame](#)

# SPRINT #4 [03-20-23 to 04-11-23]

CODE WORKING ON LEVEL 1



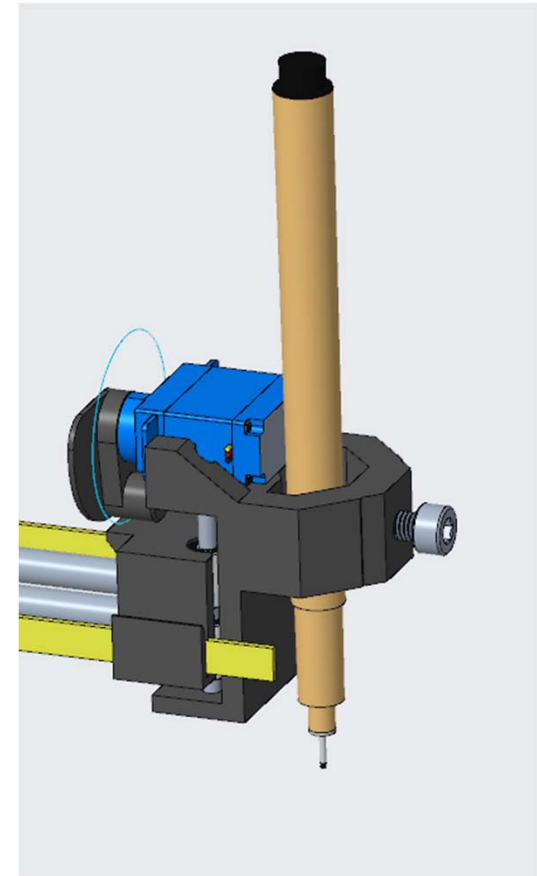
## SPRINT #3 [2/27/2023 to 3/17/2023]

### How to solve the issues

- Attaching a compatible pen holder to draw out the board from the code

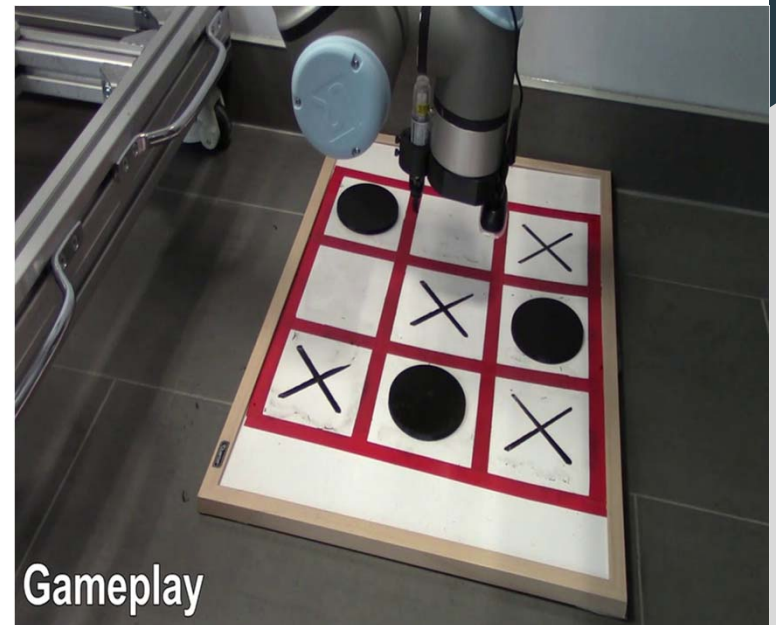
### How to reduce failure risk

- Testing the pen holders we have
- And testing different play moves on the 10 X 10 board



# Conclusion

- **Project goal:** Help children reduce excessive phone usage
- Introduce interactive tic-tac-toe playing robot
- Design components: Mega Pi board, Motors, Pen holder
- Enables play on 10 x 10 grid (*Future -> Chess*)
- **Thorough testing:**
  - Different pen holders
  - Game moves (*Strategies*)
- Ensure device reliability and minimize failure
- **Benefits:**
  - Stimulating alternative to screen time
  - Promotes cognitive development
  - Enhances social skills
  - Engaging classic game



# Resources



- S. Kelly, *Python, PyGame, and Raspberry Pi game development*. New York: Apress, 2019.
- T. Wu and C. Yu, "Tic-tac-toe prediction based on machine learning methods," 202. *International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)*, Wuhan, China, 2022, pp. 397-402, doi: 10.1109/AEMCSE55572.2022.00085.
- D. B. Fogel, "Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe," *IEEE International Conference on Neural Networks*, San Francisco, CA, USA, 1993, pp. 875-880 vol.2, doi: 10.1109/ICNN.1993.298673
- Agmed, tech\_support, and System, "Laserbot laser Makeblock," *Makeblock Forum*, 06-Aug-2020. Available: <https://forum.makeblock.com/t/laserbot-laser-upgrade/17029>.
- Real Python, "Build a tic-tac-toe game engine with an AI player in python," *Real Python*, 24-Feb-2023. [Online]. Available: <https://realpython.com/tic-tac-toe-ai-python/>.
- "About MegaPi – makeblock help center.": <https://support.makeblock.com/hc/en-us/articles/4412894483095-About-MegaPi>.



Q & A