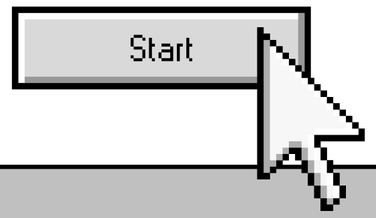
ODT A1

Home Samruddhi Kavya

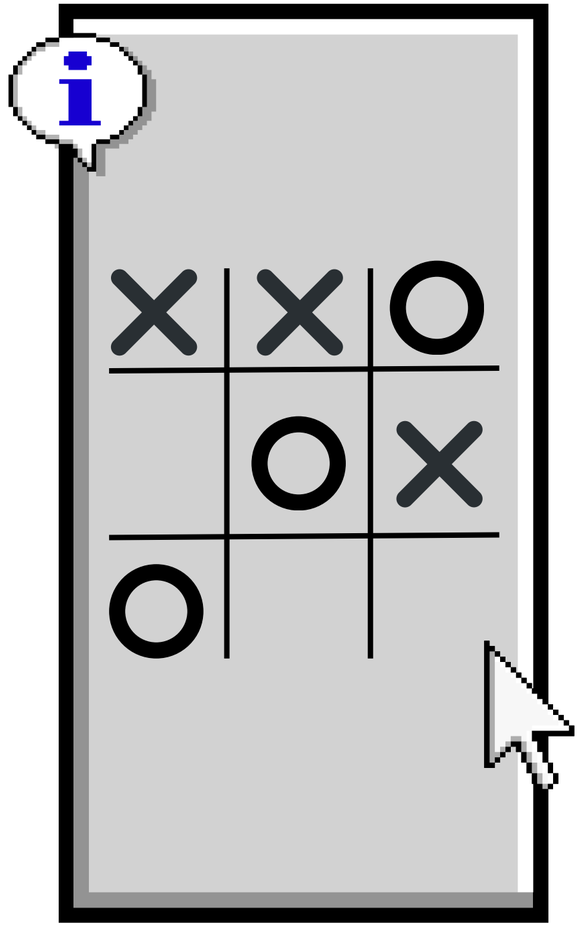
Bored in class?
Grab a friend and come play

TIC TAC GLOW



ODT A1

Home



COMPONENTS

* ESP32 → machine module
* NeoPixel ring → the game board
* Button 1 → moves cursor
* Button 2 → places move
* Buzzer → win indication (yay!)

ODT A1

Home

HOW IT WORKS

When the move button is pressed, a green LED moves across the 9 squares.

When the select button is pressed, the current player marks that square.

* Player 1 = Red
* Player 2 = Blue
* Green = current selection

Board list:
* 0 → empty
* 1 → Player 1
* 2 → Player 2

ODT A1

Home

HOW IT WORKS

List of all possible winning combinations. After each move, we check whether any combination contains the same non-zero value.

For example:

[0,1,2]
[3,4,5]
[6,7,8]

When a win or draw is detected, the buzzer activates and the LEDs animate as a celebration.

(yay!)

ODT A1

Home

CODE

Let’s go 2 Thonny

```
from machine import Pin
import time
import neopixel

pb1 = Pin(14, Pin.IN, Pin.PULL_UP)
pb2 = Pin(19, Pin.IN, Pin.PULL_UP)

buzzer = Pin(25, Pin.OUT)


raju = neopixel.NeoPixel(Pin(23), 16)


board = [0,0,0,0,0,0,0,0,0]


wins = [[0,1,2],[3,4,5],[6,7,8],[0,3,6],[1,4,7],[2,5,8],[0,4,8],[2,4,6]]

current_square = 0
current_player = 1
game_over = False
```

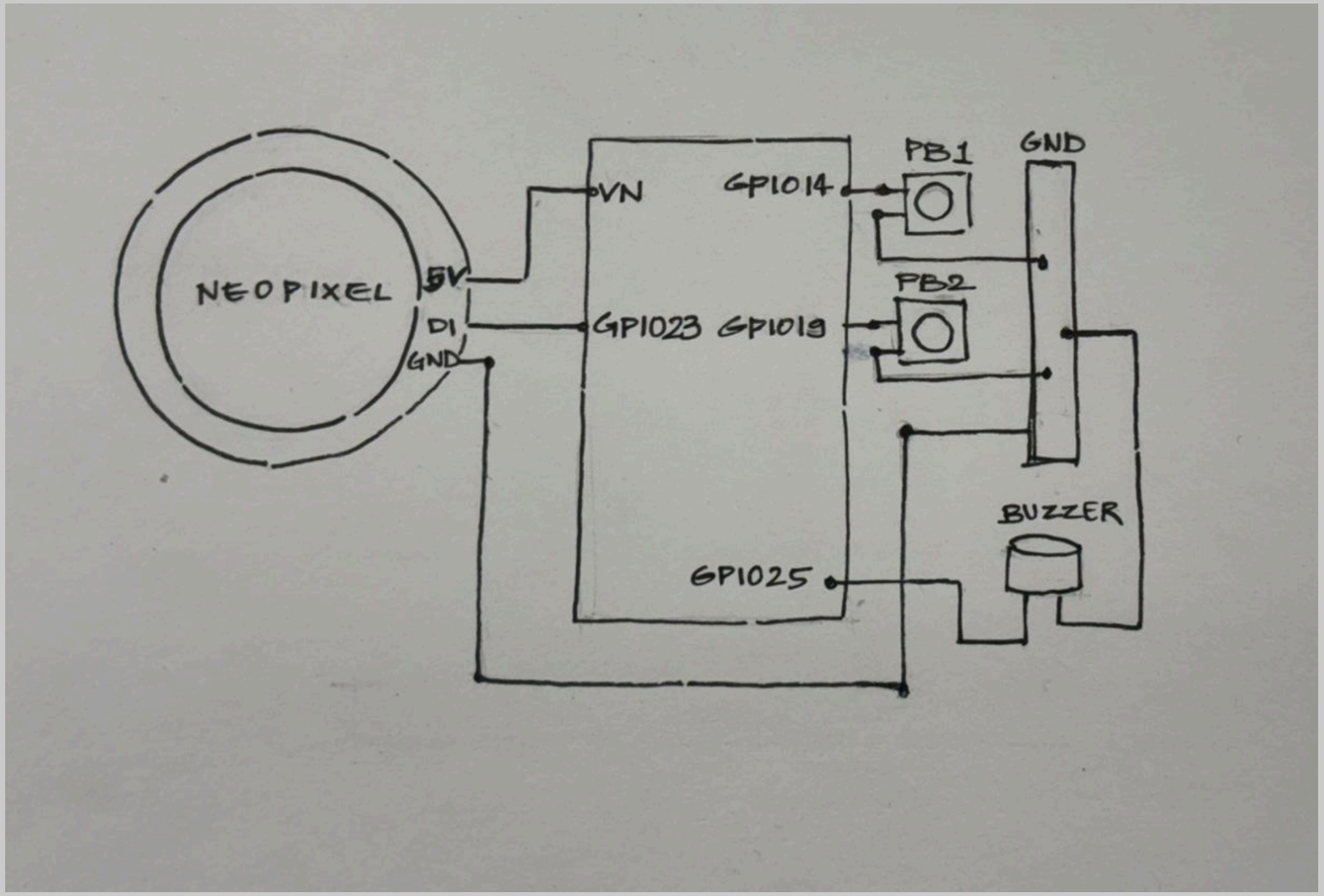
```
while True:

    if not game_over:


        for i in range(16):
            raju[i] = (0,0,0)


        for i in range(9):
            if board[i] == 1:
                raju[i] = (255,0,0)
            if board[i] == 2:
                raju[i] = (0,0,255)


        if board[current_square] == 0:
            raju[current_square] = (0,255,0)

        raju.write()
        time.sleep(0.2)
```

```
        if pb1.value() == 0:
            print(pb1.value())
            current_square += 1
            if current_square > 8:
                current_square = 0
            time.sleep(0.3)


        if pb2.value() == 0:
            if board[current_square] == 0:

                board[current_square] = current_player


                if current_player == 1:
                    current_player = 2
                else:
                    current_player = 1
```

```
                for w in wins:
                    if (board[w[0]] != 0 and
                        board[w[0]] == board[w[1]] == board[w[2]]):
                        game_over = True


                if 0 not in board:
                    game_over = True

            time.sleep(0.3)

    else:

        buzzer.value(1)
        time.sleep(0.4)
        buzzer.value(0)


        colors = [(255,0,0),(0,255,0),(0,0,255),(255,255,0),(255,0,255),(0,255,255)]
```

```
            for c in colors:
                for i in range(9):
                    raju[i] = c
                raju.write()
                time.sleep(0.3)

            break
```

CIRCUIT DIAGRAM:



PROCESS JOURNEY & REFLECTION

When we began this project, the ideation stage took longer than we expected because we were trying to balance creativity with practicality. We wanted something unique, but also realistic and achievable using the concepts we had learned in class. After a lot of discussion, we chose Tic-Tac-Toe because it is a universally familiar game, and that familiarity allowed us to focus more on how we could reinterpret it through code and hardware. The interesting challenge was that a traditional Tic-Tac-Toe board is linear and grid-based, while ours was built on a circular NeoPixel ring. This meant we had to rethink how the positions mapped logically and visually, and figuring out that “math” and indexing was an important part of the process.

Midway through, we faced significant hardware issues. Our NeoPixel wasn’t working, and after a lot of confusion, we discovered that the power supply module was outputting nearly double the required voltage — which unfortunately fried our first NeoPixel. We had to borrow another one and be much more cautious during the second attempt, even improvising by using the ESP32 as a power source. Debugging the code also required patience, especially for small but impactful mistakes like assigning the wrong GPIO pin.

After user testing, we made iterative improvements, such as numbering the LEDs for better strategic planning and adjusting finger placement to make the interaction smoother. Overall, the experience was stressful at times but deeply rewarding, as we were able to apply classroom concepts like lists and conditional logic in a tangible, hands-on way.

WORK DIVISION

Samruddhi- code, ppt, video, decorative elements

Kavya- circuitry, decorative elements, code