Tetris Battle 2.0 DEMO

B07901090 劉紀霆
B08901169 梁正
B08901123 何景盛

Github Link:
https://github.com/Gting6/NTUEE_EMBED_LAB/tree/main/Final

Outline

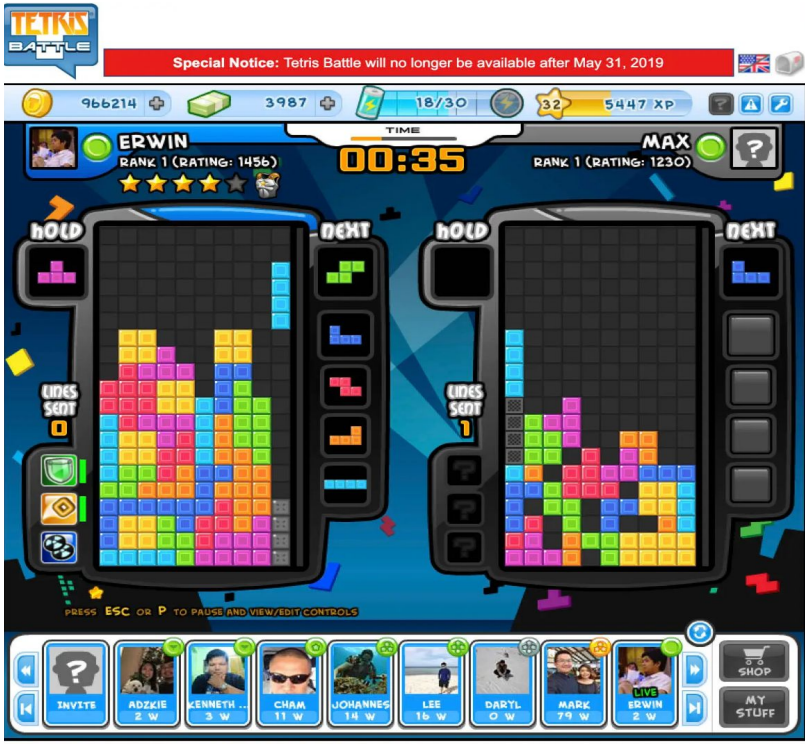
- Introduction
- Demo
- Technical Details
  - Tetris Game: Socket Programming + Multi-thread
  - LED Display: Mutex, FIFO
  - Joystick: STM32 Optimization

Motivation

This well-known little game, Tetris Battle on Facebook brought lots of joys in our childhood

However, this game was out of service on March31, 2020

We decide to make it alive again!
(and we successed!)

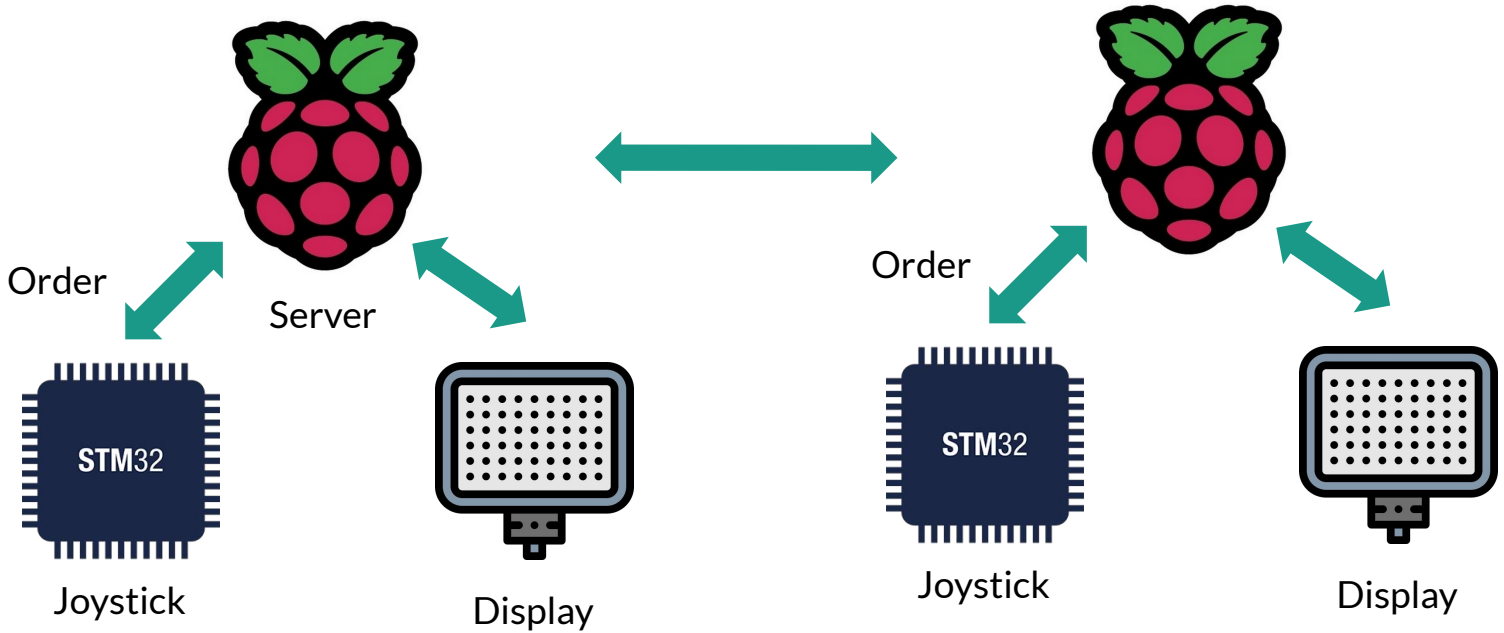


Expected Result

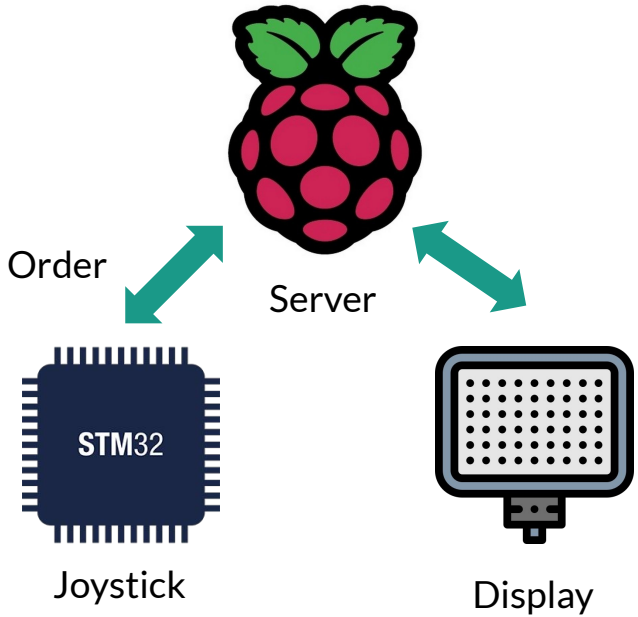
1. 1 Player Tetris
2. 2 Player Tetris Battle
3. Tetris Battle on LED screen

Use STM32 as joysticks, RPI to run the program, Screen or LED Board to display

Expected Result

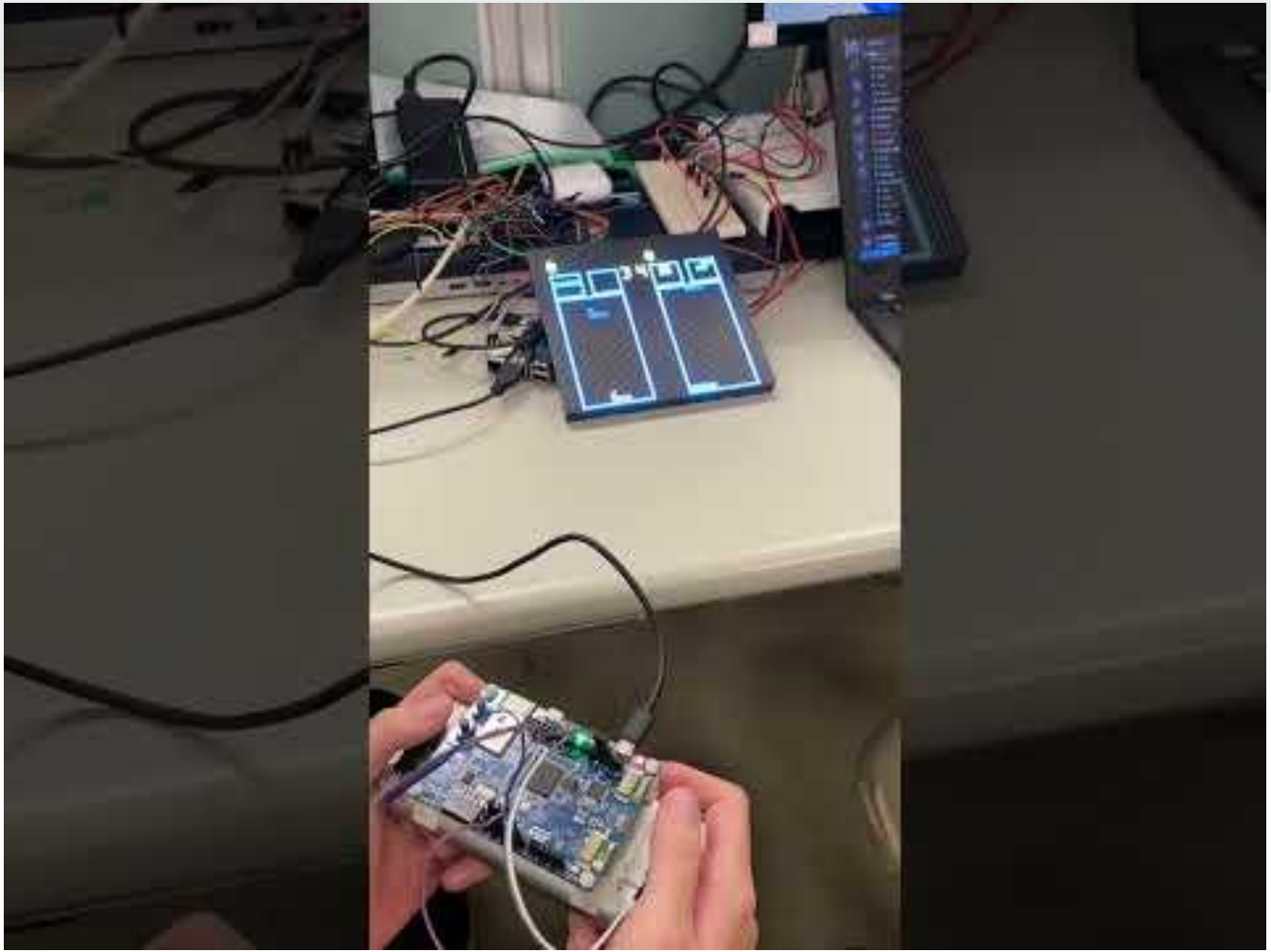


Expected Result



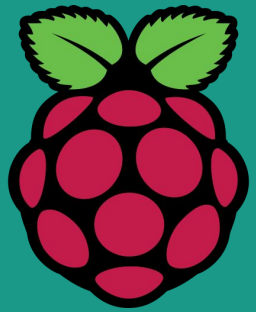
Just for the simplicity of demo
(can be replaced by RPI + STM + LED)

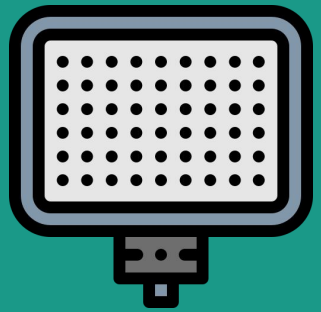
Demo



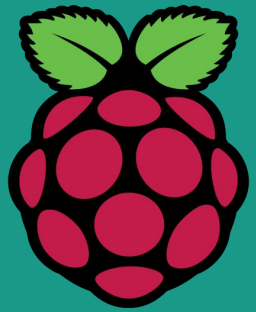
https://youtu.be/rcX75-qD13I

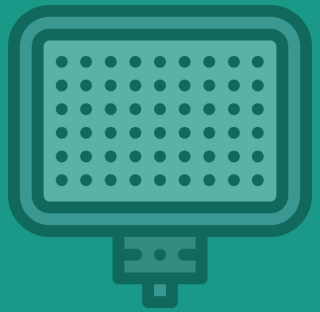
Take a closer look

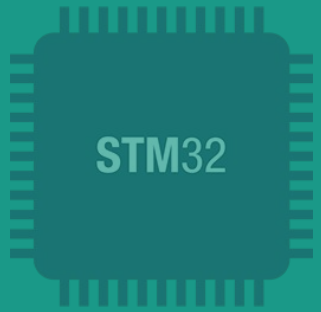




Take a closer look





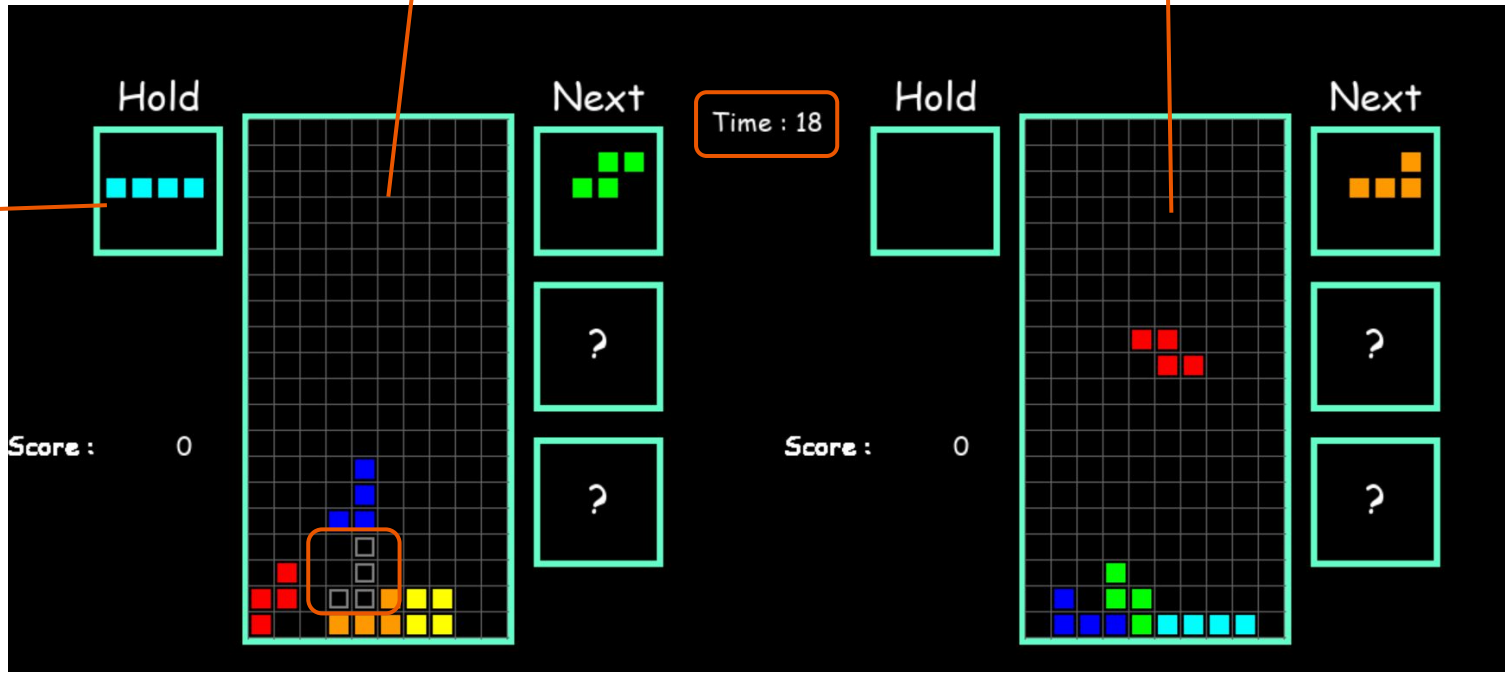


RPI: Tetris Client + STM Listner

10 x 20 Matrix

Get Opponent's Info from server

4 x 4 Matrix



RPI: Tetris Client + STM Listner

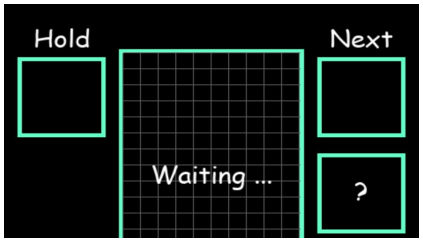
Use 3 Thread + Socket programming

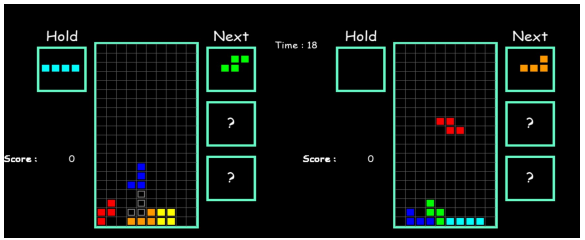
1. Listen to STM32 , push into queue
2. Main logic of tetris game (Retrieve info from queue).
   Send info and retrieve the status
   a. Rerender the scene
   b. Write to the FIFO for LED (Use ctype library to transfer data from python to C)
3. Countdown the time
   a. Once receive state change from server

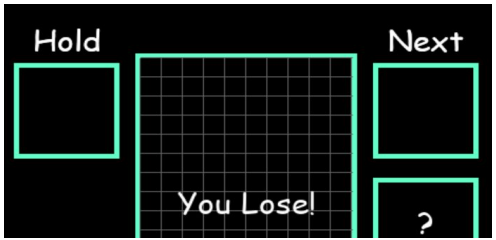
RPI: Tetris Server - Exchange status

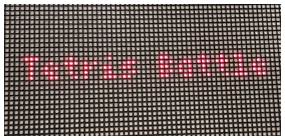
- Status // Main screen
- Game // Game status, only received from server
- Shift0
- Next0
- Combo
- Point

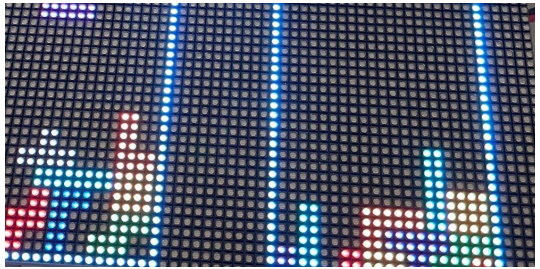
RPI: Tetris Server - Control State of the game



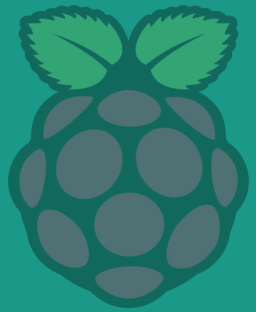


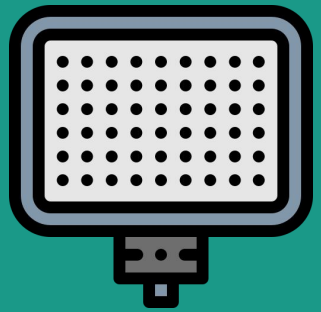


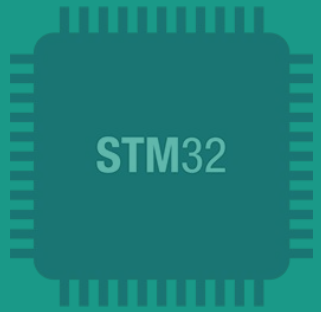




Take a closer look



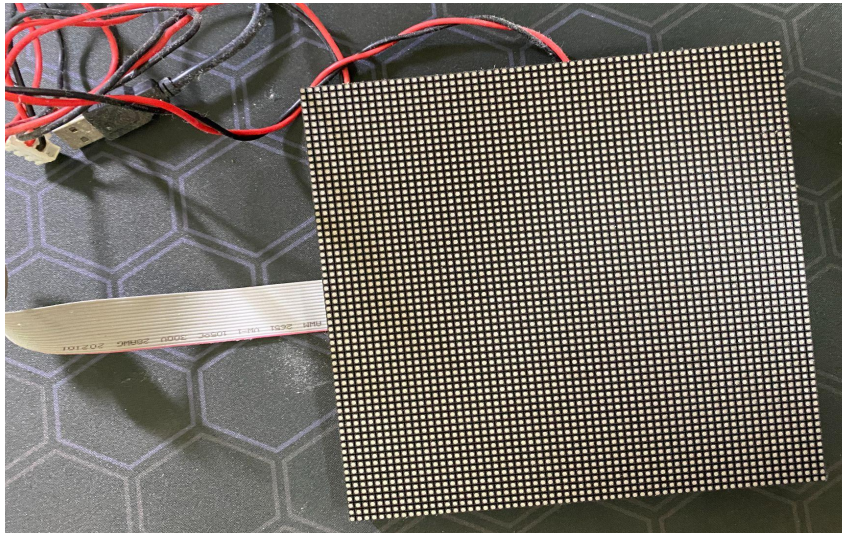


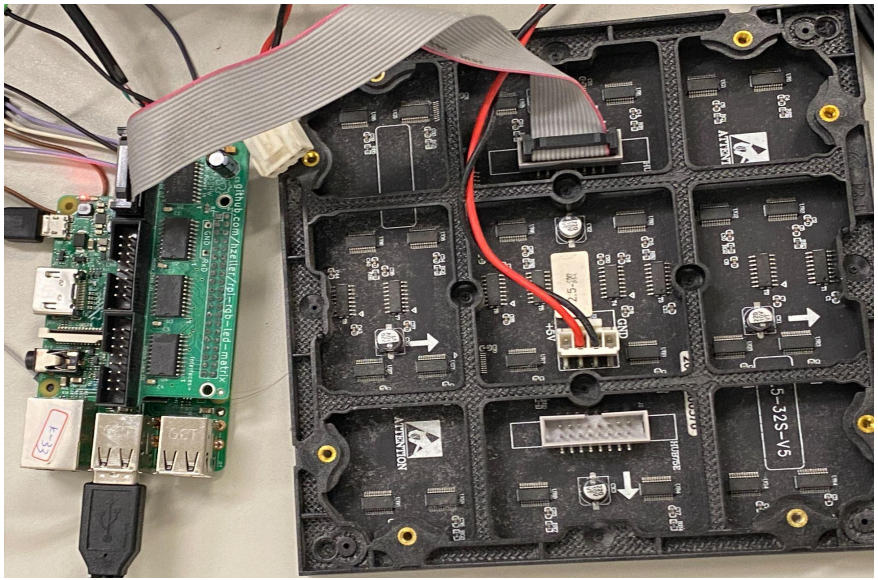


LED Display

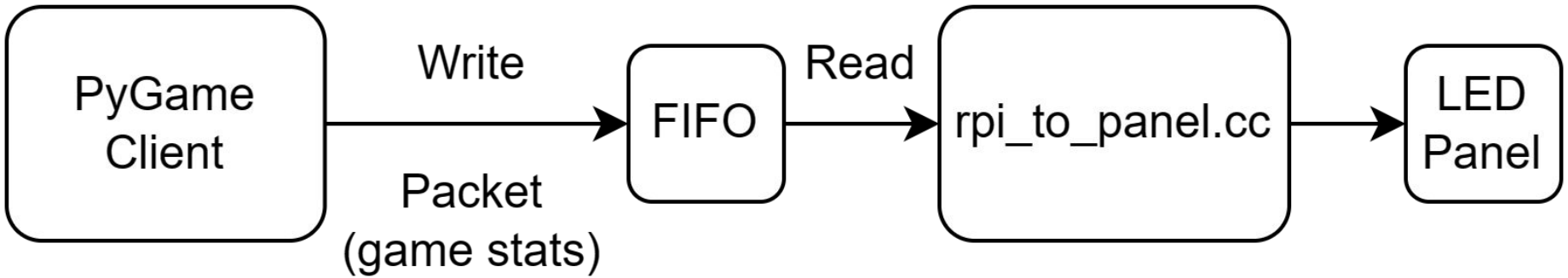
- 64 x 64 pixels RGB panel
- reference: https://github.com/hzeller/rpi-rgb-led-matrix

LED Display (hardware)

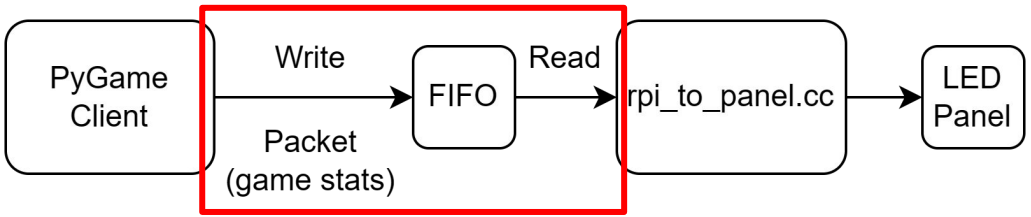




LED Display

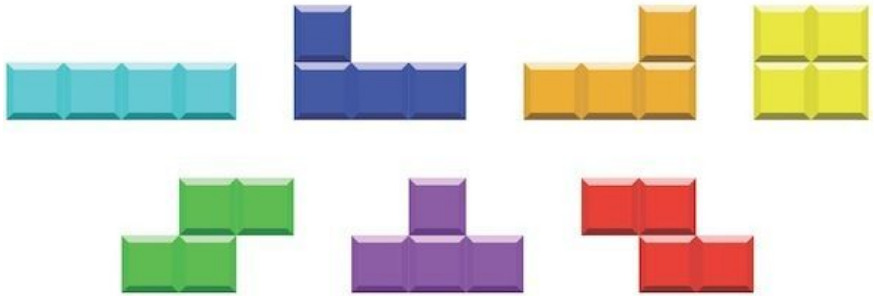

PyGame Client
Write
Packet (game stats)
FIFO
Read
rpi_to_panel.cc
LED Panel

LED Display

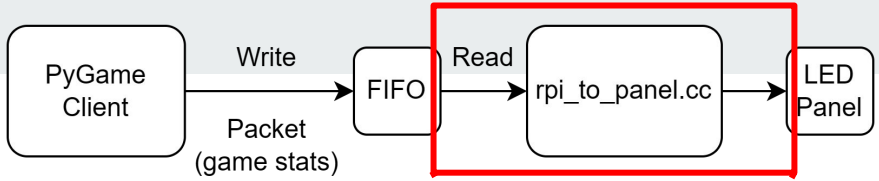



```
typedef struct Stats {
    int player0_main[10][20];
    int player0_shift[4][4];
    int player0_next[4][4];
    int player0_pts;
    int player0_cbs;
    int player1_main[10][20];
    int player1_shift[4][4];
    int player1_next[4][4];
    int player1_pts;
    int player1_cbs;
    int game;
    int time;
} Stats;
```

LED Display

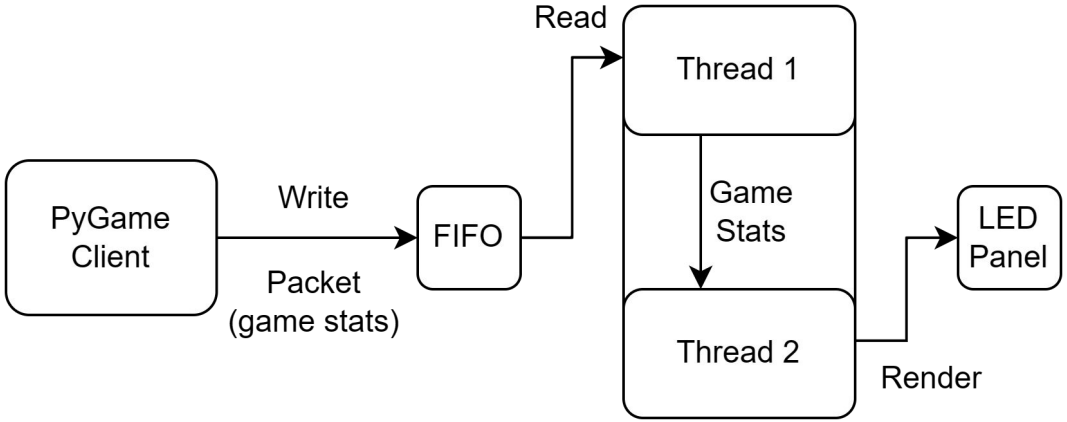


```
rgb_matrix::Color bg(100, 255, 200);        // bright blue background
rgb_matrix::Color score(255, 0, 255);       // bright yellow
rgb_matrix::Color combo(255, 0, 102);       // combo
rgb_matrix::Color text(255, 0, 0);          // text

rgb_matrix::Color EMPTY(0, 0, 0);           // void black
rgb_matrix::Color SHADOW(128, 128, 128);    // gray
rgb_matrix::Color I(23, 238, 222);          // light blue
rgb_matrix::Color J(3, 174, 65);            // deep blue
rgb_matrix::Color L(255, 28, 151);          // orange
rgb_matrix::Color O(255, 0, 213);           // yellow
rgb_matrix::Color S(114, 59, 203);          // green
rgb_matrix::Color T(118, 175, 55);          // purple
rgb_matrix::Color Z(255, 19, 50);           // red

void RGB(rgb_matrix::Color& color) {···

unordered_map<int, rgb_matrix::Color> id2color = {
    {0, EMPTY},
    {1, I},
    {2, L},
    {3, J},
    {4, O},
    {5, S},
    {6, Z},
    {7, T},
    {8, SHADOW}
};
```



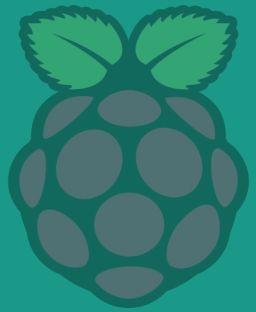
LED Display

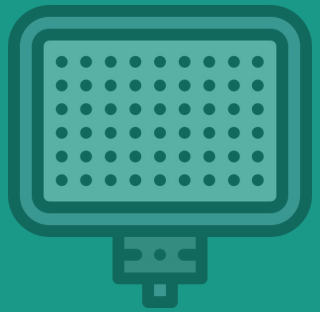
rpi_to_panel.cc




```cpp
int ready;

fd_set readset;
timeval timeout;
FD_ZERO(&readset);
FD_SET(fifo_fd, &readset);
timeout.tv_sec = 1;
timeout.tv_usec = 0;

while (!interrupt_received) {
    cerr << "thread loop" << endl;
    ready = select(fifo_fd+1, &readset, NULL, NULL, &timeout);
    if (ready == 1) {
        cerr << "ready" << endl;
        read(fifo_fd, &packet, sizeof(Stats));
        /* TODO: write to stats */
        int ret = pthread_mutex_trylock(&stats_lock);
        if (ret == 0) {
            stats = packet;
            pthread_mutex_unlock(&stats_lock);
        }
    } else if (ready == -1) {
        cerr << "select error" << endl;
        exit(1);
    }
}
```

Take a closer look



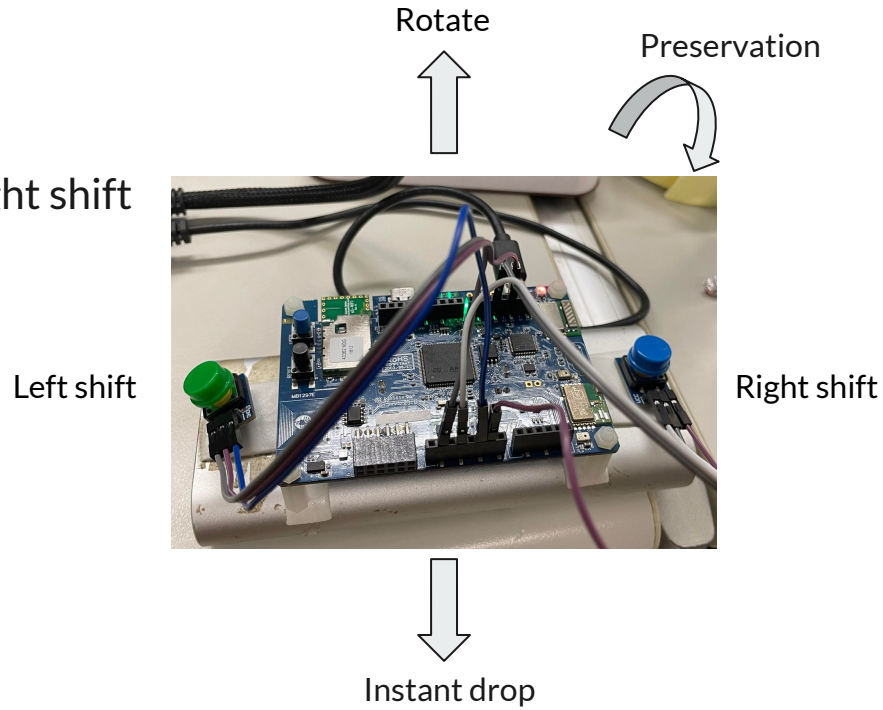


STM Controller

Use two external **button** to control left and right shift

Right rotate to preservation the tetris

Shake to rotate and instant drop

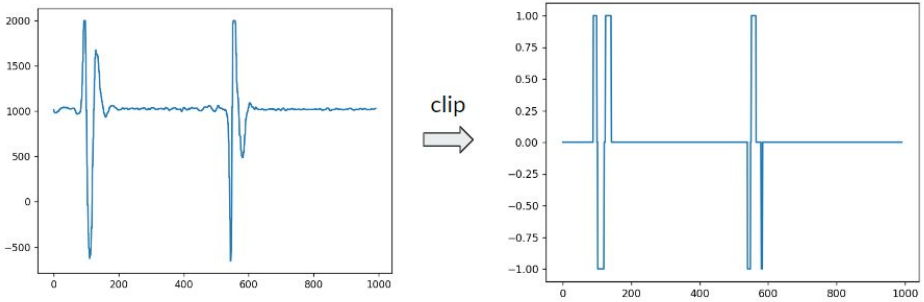


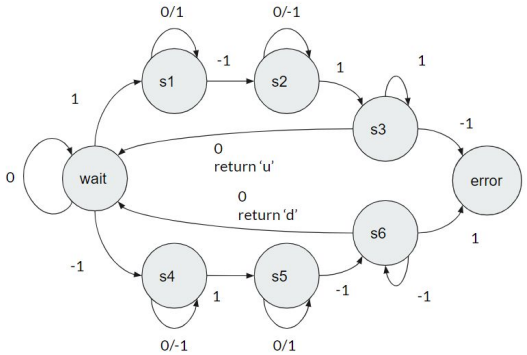
STM Controller (button)

Use mbed **InterruptIn** api to implement left and right button.

- void fall(Callback< void()> func)
- void rise(Callback< void()> func)

STM Controller





old method

accelerometer

gyroscope

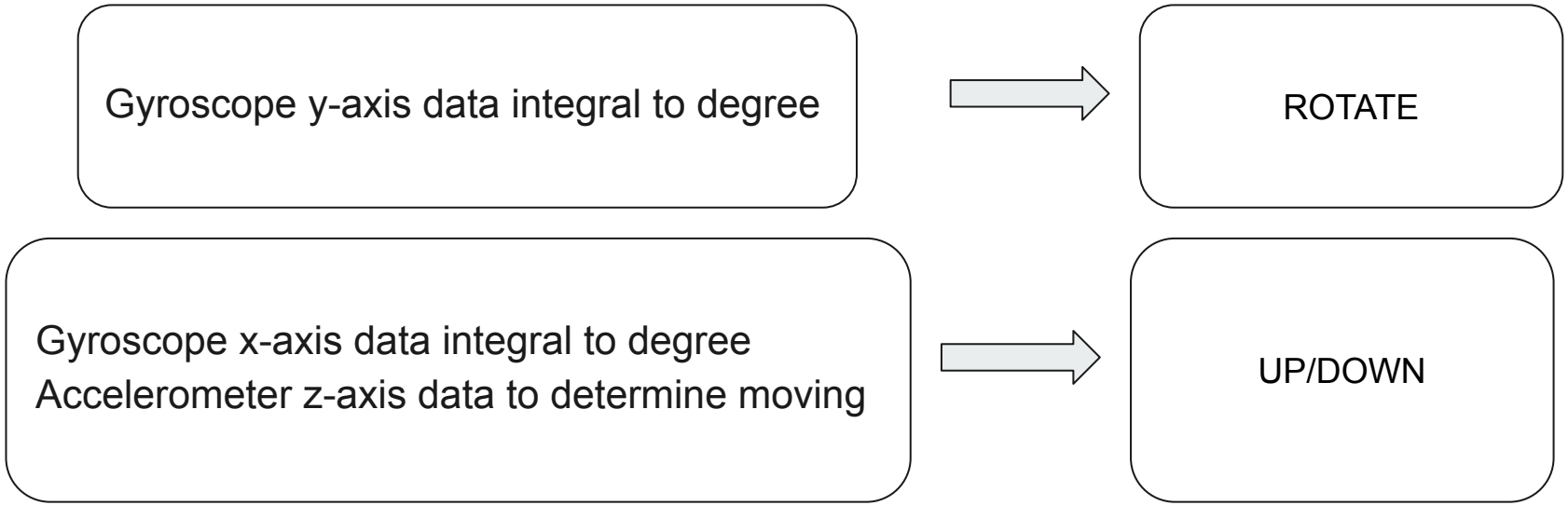
new method

STM Controller (rotate & shake) (old method)

Only accelerometer data to judge up and down by using fsm.

It’s easy to misjudge up and down with right rotate.

STM Controller (rotate & shake) (new method)



Reference

1. https://github.com/hzeller/rpi-rgb-led-matrix
2. Tetris Battle | Tetris
3. Socket Programming in Python (Guide) – Real Python
4. threading — Thread-based parallelism — Python 3.11.1 documentation
5. ctypes — A foreign function library for Python — Python 3.11.1 documentation
6. https://github.com/NTUEE-ESLab/2020-Pikachu-volleyball?fbclid=IwAR2OjNHmujj6J_A953Bmx-v4b5ptkjRtIIE-pMu8KrmLBQ7Pl4QlVhcfSbs
7. Linux Tutorial: POSIX Threads (cmu.edu)

Thanks!