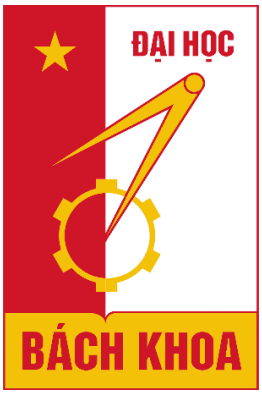
ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO ĐỒ ÁN MÔN HỌC

Đề tài: Xây dựng game Tetris

sử dụng KIT STM32F429ZIT6

Lớp	:	157539
Học phần	:	Hệ nhúng
Mã học phần	:	IT4210
Giảng viên hướng dẫn	:	TS. Ngô Lam Trung

Danh sách thành viên nhóm 15:

Họ và tên	Mã số sinh viên	Email
Nguyễn Tuấn Đạt	20225278	dat.nt225278@sis.hust.edu.vn
Đinh Quốc Đạt	20225275	dat.dq225275@sis.hust.edu.vn
Trần Hữu Huy	20210444	huy.th210444@sis.hust.edu.vn

Hà Nội, tháng 6 năm 2025

Mục lục

LỜI NÓI ĐẦU **3**
CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI **5**
1.1. Mô tả sản phẩm **5**
1.2. Mục tiêu của project **5**
1.3. Các yêu cầu đối với project **5**
a. Yêu cầu chức năng **5**
b. Yêu cầu phi chức năng **6**
CHƯƠNG 2: THIẾT KẾ PHẦN CỨNG **7**
2.1. Kiến trúc hệ thống **7**
2.2. Tổng quan về Kit STM32F429ZIT6 ..... **8**
2.2.1. Phần cứng của Kit 9
2.2.2. Tính năng, giao thức kết nối của Kit 10
2.2.3. Vị trí các thành phần trên Kit và kích thước 12
2.3. Mạch nút bấm **13**
CHƯƠNG 3: THIẾT KẾ PHẦN MỀM VÀ TRIỂN KHAI CÀI ĐẶT **13**
3.1. Sơ đồ khối hệ thống **13**
3.2. Luồng hệ thống **14**
3.3. Module phần mềm tương ứng các tính năng **14**
CHƯƠNG 4: ĐÁNH GIÁ TỔNG QUAN **23**
1. Giao diện người dùng ..... Error! Bookmark not defined.
2. Đánh giá hệ thống **25**
PHỤ LỤC **26**
1. Phân công nhiệm vụ **26**
2. Tài liệu tham khảo **26**
3. Mã nguồn (source code) và các resource của đồ án **26**

LỜI NÓI ĐẦU

Công nghệ thông tin ngày càng phát triển và có vai trò hết sức quan trọng không thể thiếu trong cuộc sống hiện đại, nó đã thay đổi mọi khía cạnh của cuộc sống, từ việc giao tiếp, giải trí, đến học tập và công việc. Chúng ta sống trong một thế giới với hàng tỷ thiết bị kết nối với Internet, tạo nên mạng lưới thông tin phức tạp. Nhưng không chỉ có những thiết bị lớn như máy tính cá nhân và điện thoại thông minh, công nghệ thông tin còn hiện diện ẩn mình trong những hệ thống nhỏ gọn và tích hợp, được gọi là hệ thống nhúng.

Hệ thống nhúng là những thiết bị và hệ thống tích hợp các vi xử lý, cảm biến, và giao tiếp mạng. Điển hình cho các ứng dụng của hệ thống nhúng là các thiết bị di động, các thiết bị y tế thông minh, đồ gia dụng kết nối Internet (Internet of Things - IoT) và nhiều ứng dụng khác.

Sự phát triển của công nghệ đã thúc đẩy sự tiến bộ của hệ nhúng. Các bộ vi xử lý ngày càng mạnh mẽ và nhỏ gọn, công nghệ bộ nhớ và lưu trữ phát triển nhanh chóng, đồng thời với đó là xu hướng phần mềm mã nguồn mở và các công cụ phát triển hệ thống nhúng. Tất cả những tiến bộ này đã tạo điều kiện thuận lợi cho việc phát triển các ứng dụng hệ thống nhúng phức tạp và đa dạng.

Với sự hấp dẫn của lĩnh vực và những thách thức còn đang ở phía trước, với niềm đam mê, mong muốn được học hỏi các công nghệ, tiếp xúc với các bài toán của Hệ nhúng, nhóm chúng em đã quyết định lựa chọn đề tài "**Xây dựng game Tetris sử dụng KIT STM32F429ZIT6** " cho Đồ án môn học của mình.

Đồ án môn học của nhóm chúng em bao gồm 4 nội dung chính:

1. Tổng quan đề tài
2. Hệ thống phần cứng
3. Xây dựng chương trình và triển khai cài đặt
4. Đánh giá tổng quan

Mặc dù đã cố gắng hoàn thiện sản phẩm nhưng không thể tránh khỏi những thiếu hụt về kiến thức và sai sót trong kiểm thử. Chúng em rất mong nhận được những nhận xét

thẳng thắn, chi tiết đến từ thầy để tiếp tục hoàn thiện hơn nữa. Cuối cùng, nhóm chúng em xin được gửi lời cảm ơn đến thầy **TS. Ngô Lam Trung** đã hướng dẫn chúng em trong suốt quá trình hoàn thiện Đồ án môn học. Nhóm chúng em xin chân thành cảm ơn thầy.

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

1.1. Mô tả sản phẩm

- Sản phẩm của đề tài là một trò chơi **Tetris** được xây dựng và triển khai trực tiếp trên **board vi điều khiển STM32F429ZIT6**, một nền tảng mạnh mẽ thuộc dòng STM32F4 của STMicroelectronics, có tích hợp sẵn màn hình và hỗ trợ thư viện đồ họa **TouchGFX**. Trò chơi mô phỏng lại cơ chế hoạt động của Tetris cổ điển – một trò chơi xếp hình nổi tiếng, quen thuộc với nhiều thế hệ người dùng, với giao diện trực quan, xử lý logic chính xác và khả năng tương tác trực tiếp thông qua phần cứng.

1.2. Mục tiêu của project

- Sản phẩm được thiết kế với mục tiêu vừa học tập vừa giải trí, đồng thời giúp người thực hiện tiếp cận sâu hơn với các kỹ thuật lập trình nhúng thực tế như: xử lý sự kiện từ nút nhấn, vẽ đồ họa thời gian thực, điều khiển âm thanh và tổ chức luồng xử lý chương trình.

1.3. Các yêu cầu đối với project

a. Yêu cầu chức năng

- **Khởi tạo và hiển thị lưới chơi** (matrix Tetris 11 cột x 20 hàng) trên màn hình khi bắt đầu trò chơi.
- **Tạo khối Tetris (Tetromino)** ngẫu nhiên với đúng hình dạng (I, O, T, L, J, S, Z) và màu sắc khác nhau.
- **Cho phép điều khiển khối bằng 4 nút nhấn vật lý**:
  - Nút sang trái: dịch chuyển khối sang trái một ô.
  - Nút sang phải: dịch chuyển khối sang phải một ô.
  - Nút xoay: xoay khối theo chiều kim đồng hồ.
  - Nút rơi nhanh: đẩy khối rơi nhanh xuống đáy.
- **Kiểm tra va chạm**: ngăn khối đi ra ngoài ranh giới hoặc chồng lên khối đã cố định.
- **Xử lý khi khối chạm đáy**:
  - Cố định vị trí khối.
  - Kiểm tra và xóa hàng nếu hàng đã đầy.
  - Cập nhật điểm số.
- **Tạo khối mới** sau mỗi lần cố định xong khối cũ.
- **Hiển thị điểm số** trên màn hình và cập nhật điểm số.
- **Phát âm thanh từ buzzer** mỗi khi người chơi thực hiện thao tác điều khiển.
- **Kết thúc trò chơi** khi không còn không gian cho khối mới (game over).

b. Yêu cầu phi chức năng

- **Độ ổn định và phản hồi nhanh**: Trò chơi phải xử lý thao tác người dùng nhanh chóng, không bị treo hoặc đơ.
- **Hiển thị mượt mà, không bị giật hình**: Các khối cần được hiển thị chính xác vị trí, không bị chớp nháy khi rơi hoặc xoay.
- **Âm thanh rõ ràng, không bị rè** khi buzzer phát ra âm thanh.
- **Giao diện thân thiện**: Màu sắc các khối dễ phân biệt, điểm số dễ quan sát, bố cục rõ ràng.
- **Thiết kế phần cứng chắc chắn**: Các nút nhấn được gắn chắc, dây kết nối an toàn.

CHƯƠNG 2: THIẾT KẾ PHẦN CỨNG

2.1. Kiến trúc hệ thống

- Hệ thống gồm 5 thành phần chính:
  1. Kit STM32F429I-DISC1
  2. Mini USB-cable
  3. 4 nút bấm
  4. Dây nối
  5. Buzzer
- Sơ đồ kết nối các chân nút bấm với GPIO của kit

STM32F429ZIT6	Ngoại vi
PC11	Button sang trái
PC12	Button sang phải
PA14	Button xoay khối
PA15	Button tăng tốc khối
PC10	Buzzer

2.2. Tổng quan về Kit STM32F429ZIT6

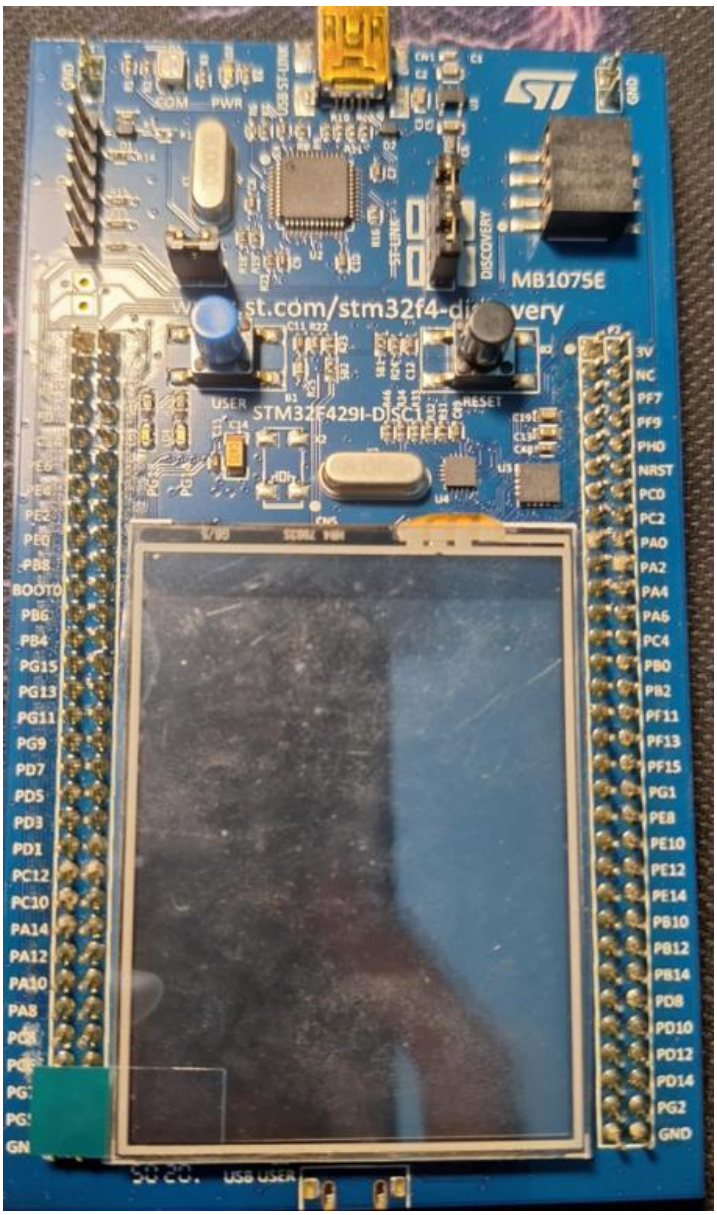


Figure 1: Mặt trước của kit stm32f429zit6

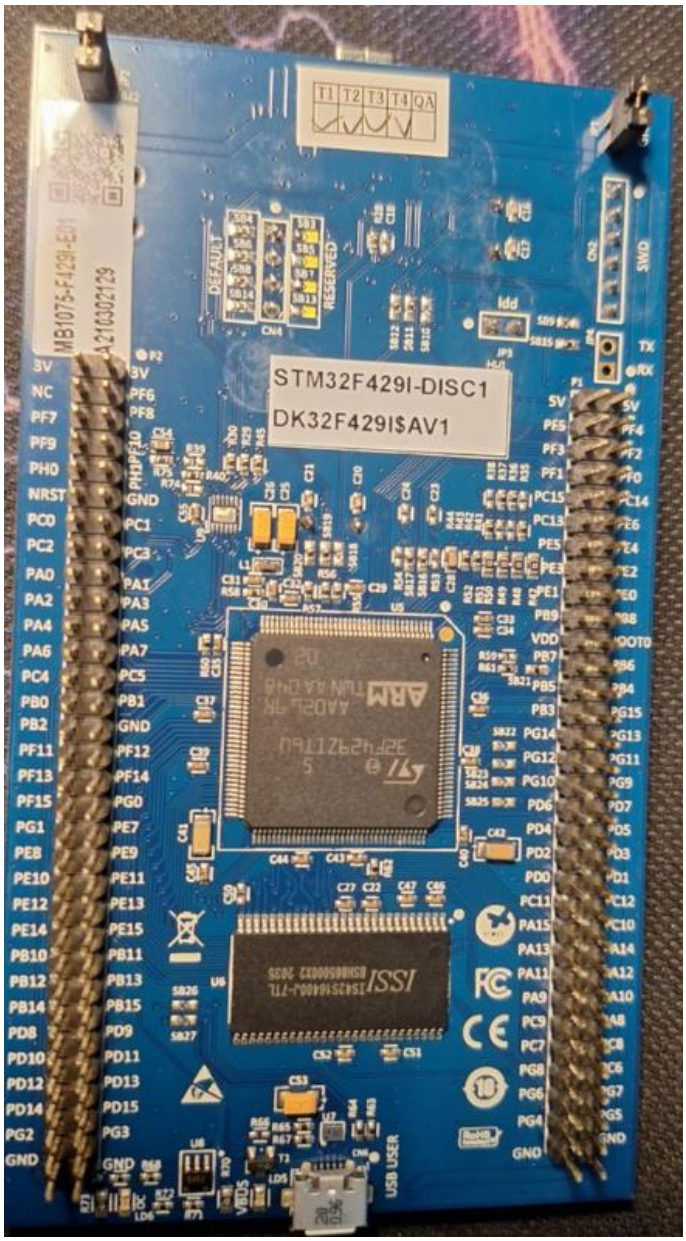


Figure 2: Mặt sau của kit stm32f429zit6

2.2.1. Phần cứng của Kit

Bộ kit phát triển sử dụng vi điều khiển **STM32F429ZIT6** là một nền tảng phần cứng mạnh, tích hợp đầy đủ các thành phần cần thiết để xây dựng và thử nghiệm các ứng dụng nhúng trong thực tế. Kit được thiết kế tối ưu hóa cho mục tiêu học tập, nghiên cứu, cũng như phát triển sản phẩm nguyên mẫu (prototype). Các thành phần phần cứng chính bao gồm:

2.2.1.1. Vi điều khiển STM32F429ZIT6

- Vi xử lý: ARM Cortex-M4 32-bit với bộ tính dấu phẩy động (FPU), hoạt động ở tần số lên đến 180 MHz.
- Bộ nhớ: 2 MB Flash và 256 KB SRAM.

- Tích hợp: bộ tạo mã kiểm tra CRC, đồng hồ thời gian thực (RTC), watchdog timer, bộ định thời (TIM), bộ điều khiển ngắt (NVIC), và các kênh DMA.

2.2.1.2 Hệ thống cấp nguồn

- Hỗ trợ nhiều phương thức cấp nguồn: qua cổng USB, thông qua mạch nạp ST-LINK tích hợp, hoặc bằng nguồn ngoài thông qua jack 5V.
- Trên kit có tích hợp bộ ổn áp giúp chuyển đổi điện áp 5V xuống 3.3V phục vụ cho vi điều khiển và các linh kiện ngoại vi.

2.2.1.3. Giao tiếp ngoại vi

- Hỗ trợ đầy đủ các chuẩn giao tiếp: UART, USART, SPI, I2C, CAN, ADC, DAC, PWM.
- Tích hợp cổng USB OTG FS hỗ trợ hoạt động ở cả chế độ thiết bị (Device) và máy chủ (Host).
- Hỗ trợ giao tiếp thẻ nhớ MicroSD thông qua giao diện SDIO.

2.2.1.4. Bộ nhớ mở rộng

- Có khả năng kết nối với bộ nhớ ngoài như **SDRAM** hoặc **SRAM** thông qua bộ điều khiển giao tiếp bộ nhớ FMC (Flexible Memory Controller).
- Cho phép lưu trữ dữ liệu dung lượng lớn hoặc phục vụ cho xử lý đồ họa khi hiển thị.

2.2.1.5. Khả năng hiển thị

- Tích hợp sẵn màn hình cảm ứng TFT LCD kích thước từ 2.4 đến 2.8 inch.
- Có hỗ trợ bộ điều khiển LTDC (LCD-TFT Display Controller) và DMA2D (Chrom-ART Accelerator) để tăng tốc xử lý đồ họa.

2.2.1.6. Mạch nạp và gỡ lỗi

- Trên kit được tích hợp mạch ST-LINK/V2-1 cho phép nạp chương trình và gỡ lỗi (debug) thông qua cổng USB mà không cần thiết bị ngoài.

2.2.1.7 Các thành phần ngoại vi khác

- Đèn LED: hỗ trợ kiểm tra trạng thái hoặc hiển thị tín hiệu đầu ra.
- Nút nhấn (User Button): dùng để kiểm tra chức năng ngắt ngoài (external interrupt).
- Ngoài ra, một số phiên bản kit còn tích hợp thêm cảm biến gia tốc, micro thu âm, hoặc các cổng mở rộng để kết nối thêm các module chức năng khác.

2.2.2. Tính năng, giao thức kết nối của Kit

Bộ kit phát triển STM32F429ZIT6 được tích hợp nhiều tính năng nổi bật và hỗ trợ đa dạng các giao thức kết nối, cho phép người dùng dễ dàng xây dựng, thử nghiệm các ứng dụng trong lĩnh

vực điều khiển nhúng, giao tiếp thiết bị ngoại vi, xử lý tín hiệu số, và các ứng dụng giao diện đồ họa.

2.2.2.1. Các tính năng nổi bật

- **Xử lý hiệu năng cao**: Vi điều khiển ARM Cortex-M4 hoạt động ở tần số 180 MHz, tích hợp FPU (bộ tính dấu phẩy động) giúp xử lý nhanh các phép toán số thực, phù hợp với các ứng dụng tính toán nhanh như xử lý ảnh, âm thanh, và điều khiển động cơ.
- **Hỗ trợ hiển thị đồ họa**: Tích hợp bộ điều khiển LCD-TFT (LTDC) và DMA2D (Chrom-ART) giúp hỗ trợ hiển thị màu 16-bit hoặc 24-bit, phục vụ tốt cho các giao diện người dùng (GUI).
- **Khả năng mở rộng bộ nhớ**: Hỗ trợ kết nối với SDRAM ngoài thông qua FMC, phù hợp cho các ứng dụng cần vùng nhớ lớn như TouchGFX, xử lý hình ảnh.
- **Hệ thống ngắt và DMA**: Cho phép quản lý hiệu quả các luồng dữ liệu với độ trễ thấp, giảm tải CPU trong khi xử lý tín hiệu hoặc truyền nhận dữ liệu.

2.2.1.2. Các giao thức kết nối được hỗ trợ

a. Giao tiếp nối tiếp (Serial Communication)

- **USART/UART**: Giao tiếp phổ biến dùng để truyền/nhận dữ liệu với máy tính hoặc module Bluetooth, GPS,...
- **SPI (Serial Peripheral Interface)**: Kết nối với các cảm biến, bộ nhớ Flash, màn hình OLED, hoặc module RF.
- **I2C (Inter-Integrated Circuit)**: Giao tiếp với cảm biến nhiệt độ, EEPROM, RTC, màn hình OLED,...

b. Giao tiếp ngoại vi tốc độ cao

- **USB OTG (On-The-Go)**: Hỗ trợ chế độ Host hoặc Device, dùng để kết nối với USB flash, chuột, bàn phím hoặc làm thiết bị nhận dữ liệu từ máy tính.
- **SDIO (Secure Digital Input Output)**: Giao tiếp với thẻ nhớ MicroSD tốc độ cao, phục vụ lưu trữ dữ liệu lớn.

c. Giao tiếp thời gian thực và công nghiệp

- **CAN (Controller Area Network)**: Phù hợp cho ứng dụng giao tiếp công nghiệp, ô tô, truyền thông tin đa điểm tốc độ cao.
- **ADC (Analog to Digital Converter)**: Chuyển đổi tín hiệu analog từ cảm biến sang tín hiệu số để xử lý.
- **DAC (Digital to Analog Converter)**: Phát tín hiệu analog từ dữ liệu số, sử dụng trong điều khiển điện áp, tạo sóng.
- **PWM (Pulse Width Modulation)**: Điều khiển độ sáng LED, động cơ DC/servo, và tín hiệu xung.

d. Giao tiếp mở rộng

- **FMC (Flexible Memory Controller)**: Kết nối với SDRAM, SRAM, NOR/NAND Flash.
- **JTAG/SWD**: Dùng để nạp chương trình và gỡ lỗi (debug).

2.2.2.3. Kết nối người dùng

- **LED, nút nhấn**: Dùng để test nhanh phản hồi điều khiển, tạo ngắt,...
- **Màn hình cảm ứng TFT**: Tương tác với người dùng thông qua giao diện đồ họa trực quan.
- **Cổng USB mini/micro**: Dùng để cấp nguồn, nạp chương trình, và giao tiếp với máy tính.

2.2.3. Vị trí các thành phần trên Kit và kích thước

2.2.3.1. Sơ đồ khối của Kit

Kit STM32F429ZIT6 được thiết kế với cấu trúc phần cứng dạng mô-đun, tập trung vào khả năng tích hợp hiển thị, xử lý tín hiệu và giao tiếp ngoại vi. Sơ đồ khối tổng quát của Kit bao gồm các khối chức năng chính như sau:

- **Bộ vi điều khiển STM32F429ZIT6**: trung tâm xử lý, điều phối toàn bộ hoạt động của hệ thống.
- **Khối hiển thị TFT LCD**: phục vụ giao diện người dùng, tương tác cảm ứng.
- **Khối nạp và debug ST-LINK/V2-1**: hỗ trợ nạp chương trình và gỡ lỗi qua cổng USB.
- **Khối cấp nguồn**: nhận nguồn từ USB hoặc nguồn ngoài, có bộ chuyển đổi 5V ↔ 3.3V.
- **Khối giao tiếp ngoại vi**: bao gồm UART, SPI, I2C, USB OTG, SDIO, CAN.
- **Khối bộ nhớ mở rộng**: gồm SDRAM ngoài thông qua FMC.
- **Cảm biến và thiết bị ngoại vi**: như LED, nút nhấn, khe cắm thẻ nhớ, v.v

2.2.3.2. Kích thước của Kit

- **Chiều dài:** khoảng **95 mm**
- **Chiều rộng:** khoảng **65 mm**
- **Độ dày:** khoảng **15 mm** (tính cả màn hình và các chân header)

=> Kích thước này đủ nhỏ gọn để dễ dàng thao tác trên bàn làm việc, đồng thời vẫn đủ diện tích để tích hợp đầy đủ các thành phần chức năng.

2.3. Mạch nút bấm

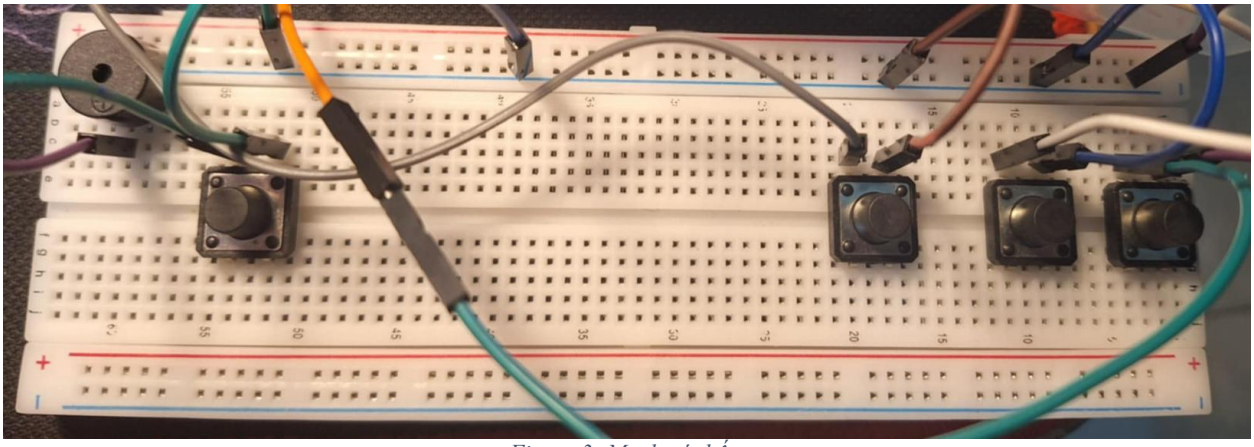


Figure 3: Mạch nút bấm

- Mạch được trang bị 4 nút nhấn, tương ứng với các thao tác điều khiển trong trò chơi gồm: di chuyển khối sang trái, sang phải, xoay khối và tăng tốc độ rơi. Ngoài ra, một buzzer được tích hợp để phát âm thanh phản hồi mỗi khi người dùng thao tác nhấn nút, nhằm tăng tính tương tác và trải nghiệm người dùng.

CHƯƠNG 3: THIẾT KẾ PHẦN MỀM VÀ TRIỂN KHAI CÀI ĐẶT

3.1. Sơ đồ khối hệ thống

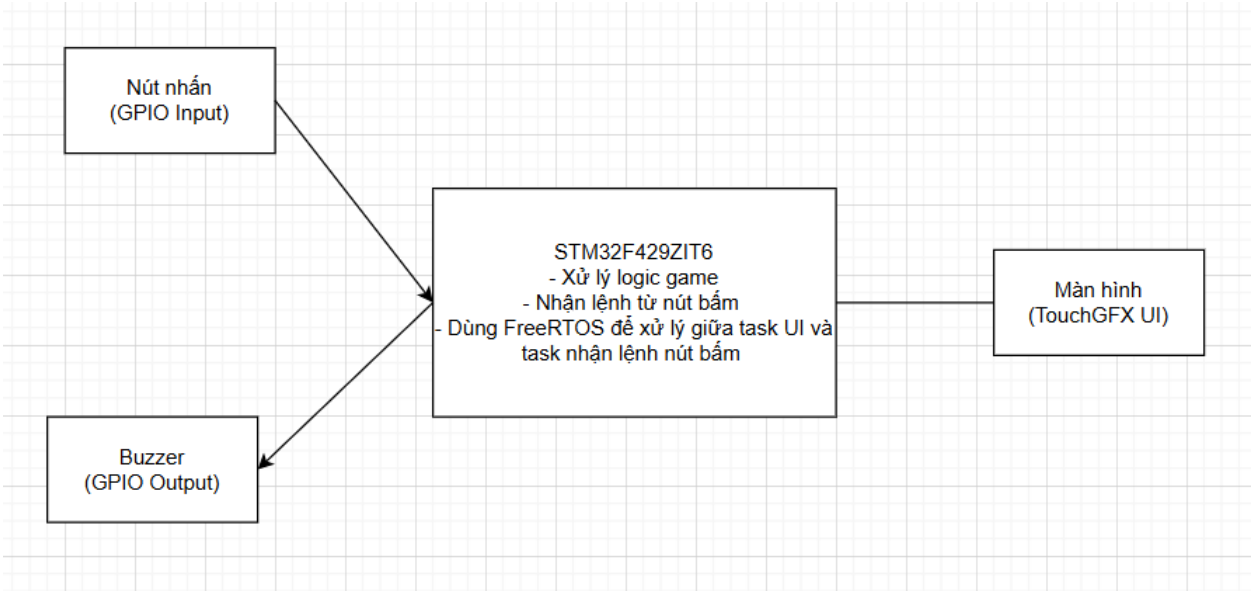


Figure 4: Sơ đồ khối của hệ thống

3.2. Luồng hệ thống

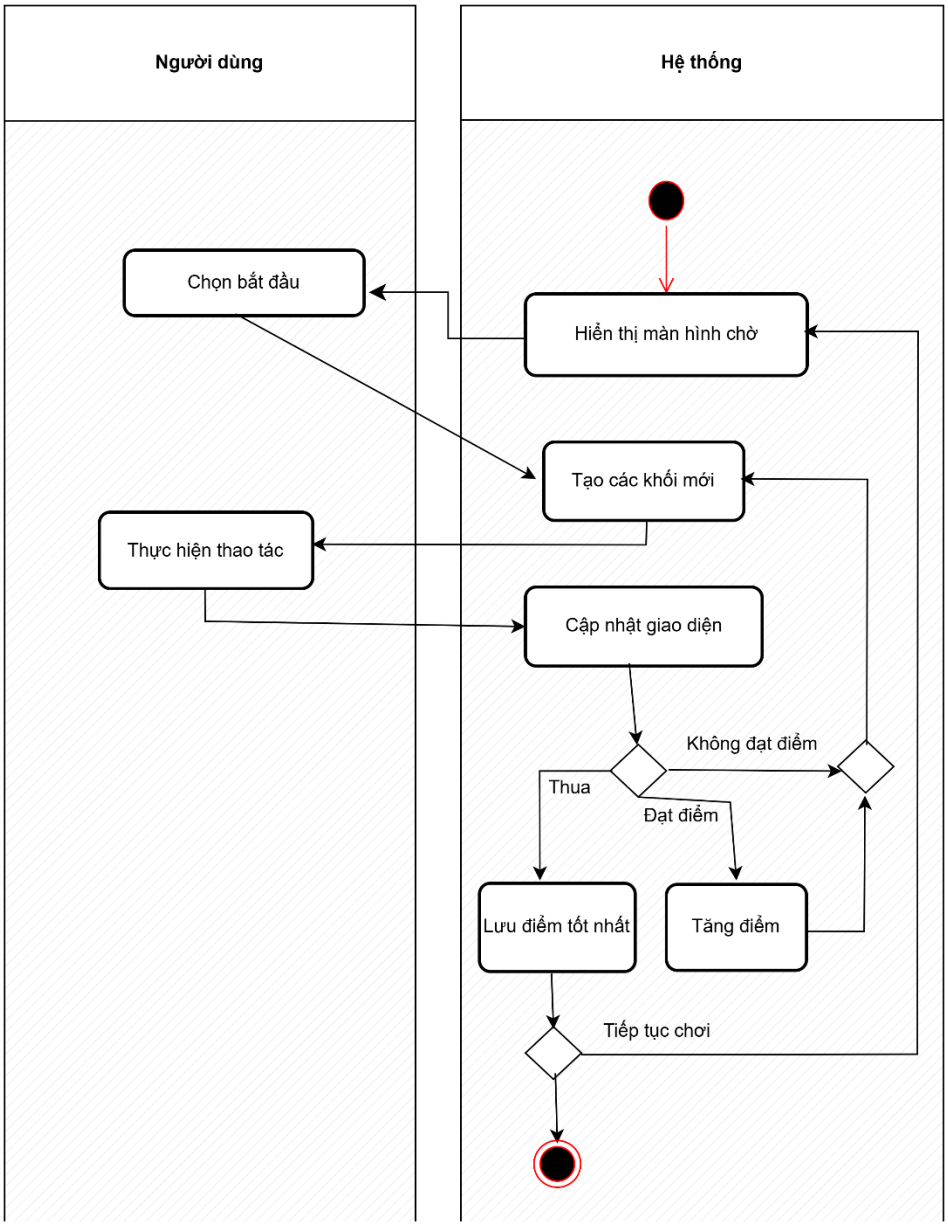


Figure 5: Sơ đồ activity luồng hoạt động của hệ thống

3.3. Module phần mềm tương ứng các tính năng

- StartDefaultTask có nhiệm vụ xử lý nhận tín hiệu từ các nút bấm, sau đó kiểm tra nút bấm gì và gửi ký tự tương ứng vô queue. Mỗi khi có sự kiện bấm nút sẽ ghi ra GPIOC10(Buzzer) tín hiệu mức cao để phát tiếng pip cho buzzer.

```c
void StartDefaultTask(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
    static uint8_t prevStateL = 1;
    static uint8_t prevStateR = 1;
    static uint8_t prevStateT = 1;
    static uint8_t prevStateS = 1;
    for(;;){
        uint8_t currStateL = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_11);
        uint8_t currStateR = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_12);
        uint8_t currStateT = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_14);
        uint8_t currStateS = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_15);
        // Nút L
        if (prevStateL == 1 && currStateL == 0) {
            uint8_t data = 'L';
            osStatus_t status = osMessageQueuePut(myQueue01Handle, &data, 0, 10);
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_SET);
            osDelay(100); // 100ms kêu "pip"
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_RESET);
        }
        // Nút R
        if (prevStateR == 1 && currStateR == 0) {
            uint8_t data = 'R';
            osStatus_t status = osMessageQueuePut(myQueue01Handle, &data, 0, 10);
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_SET);
            osDelay(100); // 100ms kêu "pip"
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_RESET);
        }
        // Nut T de Rotate
```

Figure 6: StartDefaultTask()

```c
            osStatus_t status = osMessageQueuePut(myQueue01Handle, &data, 0, 10);
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_SET);
            osDelay(100); // 100ms kêu "pip"
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_RESET);
        }
        // Nut T de Rotate
        if (prevStateT == 1 && currStateT == 0) {
            uint8_t data = 'T';
            osStatus_t status = osMessageQueuePut(myQueue01Handle, &data, 0, 10);
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_SET);
            osDelay(100); // 100ms kêu "pip"
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_RESET);
        }
        // Nut S de tang toc cho block
        if (prevStateS == 1 && currStateS == 0) {
            uint8_t data = 'S';
            osStatus_t status = osMessageQueuePut(myQueue01Handle, &data, 0, 10);
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_SET);
            osDelay(100); // 100ms kêu "pip"
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_RESET);
        }
        prevStateL = currStateL;
        prevStateR = currStateR;
        prevStateT = currStateT;
        prevStateS = currStateS;
        osDelay(20);
    }
  /* USER CODE END 5 */
}
```

Figure 7: StartDefaultTask() (tiếp)

- Hàm vẽ khối với tham số là loại khối muốn vẽ.

```
void Screen2View::drawPiece(int type)
{
    if (type < 0 || type > 6) return;

    for (int i = 0; i < 4; i++) {
        int dx = TETROMINO[type][i][0];
        int dy = TETROMINO[type][i][1];

        int x = pieceX + dx;
        int y = pieceY + dy;

//        currentBlockPos[i][0] = x;
//        currentBlockPos[i][1] = y;     //thua

        if (x >= 0 && x < 20 && y >= 0 && y < 11) {
            gridBox[x][y]->setColor(touchgfx::Color::getColorFromRGB(
                colors[type][0], colors[type][1], colors[type][2]
            ));
            gridBox[x][y]->invalidate();
        }
    }
}
```

Figure 8: drawPiece()

- Hàm xóa khối để cập nhật khi vẽ, tạo hiệu ứng khối rơi xuống dưới.

```
void Screen2View::clearPiece(int type)
{
    if (type < 0 || type > 6) return;

    for (int i = 0; i < 4; i++) {
        int dx = TETROMINO[type][i][0];
        int dy = TETROMINO[type][i][1];

        int x = pieceX + dx;
        int y = pieceY + dy;

        if (x >= 0 && x < 20 && y >= 0 && y < 11) {
            // Trả về màu nền hoặc màu trống
            gridBox[x][y]->setColor(grid[x][y].getColor());
            gridBox[x][y]->invalidate();
        }
    }
}
```

Figure 9: clearPiece()

- Hàm check xem khối có va chạm đất hoặc chạm khối khác hay không.

```
bool Screen2View::checkCollision(int type, int nextX, int nextY) {
    for (int i = 0; i < 4; i++) {
        int dx = TETROMINO[type][i][0];
        int dy = TETROMINO[type][i][1];
        int x = nextX + dx;
        int y = nextY + dy;

        if (x >= 20) return true; // cham đáy
        if (x >= 0 && grid[x][y].isOccupied()) return true; // cham block khác
    }
    return false;
}
```

Figure 10: checkCollision()

- Khi check chạm đáy hoặc chạm khối khác, thì đính khối hiện tại xuống.

```
void Screen2View::attachBlock(int type) {
    if (type < 0 || type > 6) return;

    for (int i = 0; i < 4; i++) {
        int dx = TETROMINO[type][i][0];
        int dy = TETROMINO[type][i][1];

        int x = pieceX + dx;
        int y = pieceY + dy;

        currentBlockPos[i][0] = x;
        currentBlockPos[i][1] = y;

        if (x >= 0 && x < 20 && y >= 0 && y < 11) {
            gridBox[x][y]->setColor(touchgfx::Color::getColorFromRGB(
                colors[type][0], colors[type][1], colors[type][2]
            ));
            gridBox[x][y]->invalidate();
            grid[x][y].setOccupied(true);
        }
    }
}
```

Figure 11: attachBlock()

- Hàm check xem khối có chạm vào tường bên trái hoặc chạm tường bên phải hay không.

```
bool Screen2View::checkBlockRight(int type,int y){
    int dymax = TETROMINO[type][0][1];
    for(int i=1;i<4;i++){
        if( TETROMINO[type][i][1] > dymax){
            dymax = TETROMINO[type][i][1];
        }
    }
    if(y + dymax >10)    return false;
    return true;
}

bool Screen2View::checkBlockLeft(int type,int y){
    int dymin = TETROMINO[type][0][1];
    for(int i=1;i<4;i++){
        if(TETROMINO[type][i][1] < dymin){
            dymin = TETROMINO[type][i][1];
        }
    }
    if(y + dymin < 0)    return false;
    return true;
}
```

Figure 12: checkBlockRight() và checkBlockLeft()

- Hàm dùng để xóa 1 hàng.

```
void Screen2View::clearLine(int row) {
    // Dồn tất cả hàng phía trên xuống 1 hàng
    for (int i = row; i > 0; i--) {
        for (int j = 0; j < 11; j++) {
            // Copy trạng thái của ô trên xuống ô hiện tại
            grid[i][j].setOccupied(grid[i - 1][j].isOccupied());
            // Copy màu
            gridBox[i][j]->setColor(gridBox[i - 1][j]->getColor());
        }
    }

    // Reset dòng đầu tiên (dòng 0) về rỗng
    for (int j = 0; j < 11; j++) {
        grid[0][j].setOccupied(false);
        gridBox[0][j]->setColor(touchgfx::Color::getColorFromRGB(50, 50, 50)); // Đen
    }

    // Cập nhật lại giao diện
    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 11; j++) {
            gridBox[i][j]->invalidate();
        }
    }
}
```

Figure 13: clearLine()

- Hàm check xem có hàng nào đầy hay không.

```
void Screen2View::checkFullLines() {
    for (int i = 0; i < 20; i++) { // Duyệt 20 hàng
        bool isFull = true;
        for (int j = 0; j < 11; j++) {
            if (!grid[i][j].isOccupied()) {
                isFull = false;
                break;
            }
        }

        if (isFull) {
            clearLine(i);
            currentScore++;
            Unicode::snprintf(scoreBuffer, sizeof(scoreBuffer), "%d", currentScore);
            score.setWildcard(scoreBuffer);
            score.invalidate(); // Vẽ lại TextArea với giá trị mới
        }
    }
}
```

Figure 14: checkFullLines()

- Hàm check xem có khối nào chạm đỉnh hay không.

```
int Screen2View::checkLose(){
    for(int i=0;i<11;i++){
        if(grid[0][i].isOccupied()){
            return 1;
        }
    }
    return 0;
}
```

Figure 15: checkLose()

- Hàm xoay khối, với tham số là loại khối cần xoay.

```
void Screen2View::rotatePiece(int type) {
    int pivotX = TETROMINO[type][1][0];
    int pivotY = TETROMINO[type][1][1];
    // Tính trước khối xoay mới
    for (int i = 0; i < 4; i++) {
        int dx = TETROMINO[type][i][0];
        int dy = TETROMINO[type][i][1];
        int xo = dx - pivotX;
        int yo = dy - pivotY;
        // Xoay 90 độ thuận chiều kim đồng hồ
        int xoNew = -yo;
        int yoNew = xo;

        rotatedPiece[i][0] = xoNew + pivotX;
        rotatedPiece[i][1] = yoNew + pivotY;
    }
    // Kiểm tra nếu xoay xong mà ra ngoài biên thì hủy
    int dymin = rotatedPiece[0][1], dymax = rotatedPiece[0][1];
    for (int i = 1; i < 4; i++) {
        if (rotatedPiece[i][1] < dymin) dymin = rotatedPiece[i][1];
        if (rotatedPiece[i][1] > dymax) dymax = rotatedPiece[i][1];
    }
    if (pieceY + dymin < 0 || pieceY + dymax > 10) {
        // Nếu vượt trái/phải thì không xoay
        return;
    }
    // Nếu không vượt biên, cập nhật vào khối hiện tại
    for (int i = 0; i < 4; i++) {
        TETROMINO[type][i][0] = rotatedPiece[i][0];
        TETROMINO[type][i][1] = rotatedPiece[i][1];
    }
}
```

Figure 16: rotatePiece()

- Hàm sử dụng random generator hardware để sinh khối ngẫu nhiên.

```
void Screen2View::spawnNewBlock() {
    uint32_t randNum;
    if (HAL_RNG_GenerateRandomNumber(&hrng, &randNum) == HAL_OK) {
        typeBlock = randNum % 7;
    } else {
        typeBlock = 0; // fallback nếu RNG lỗi
    }
}
```

Figure 17: spawnNewBlock()

- Hàm xử lý logic của game(1 chu kỳ của game) , đầu tiên sẽ lấy từ queue ra, kiểm tra xem lệnh từ nút bấm là gì, sau đó thực hiện theo lệnh tương ứng.

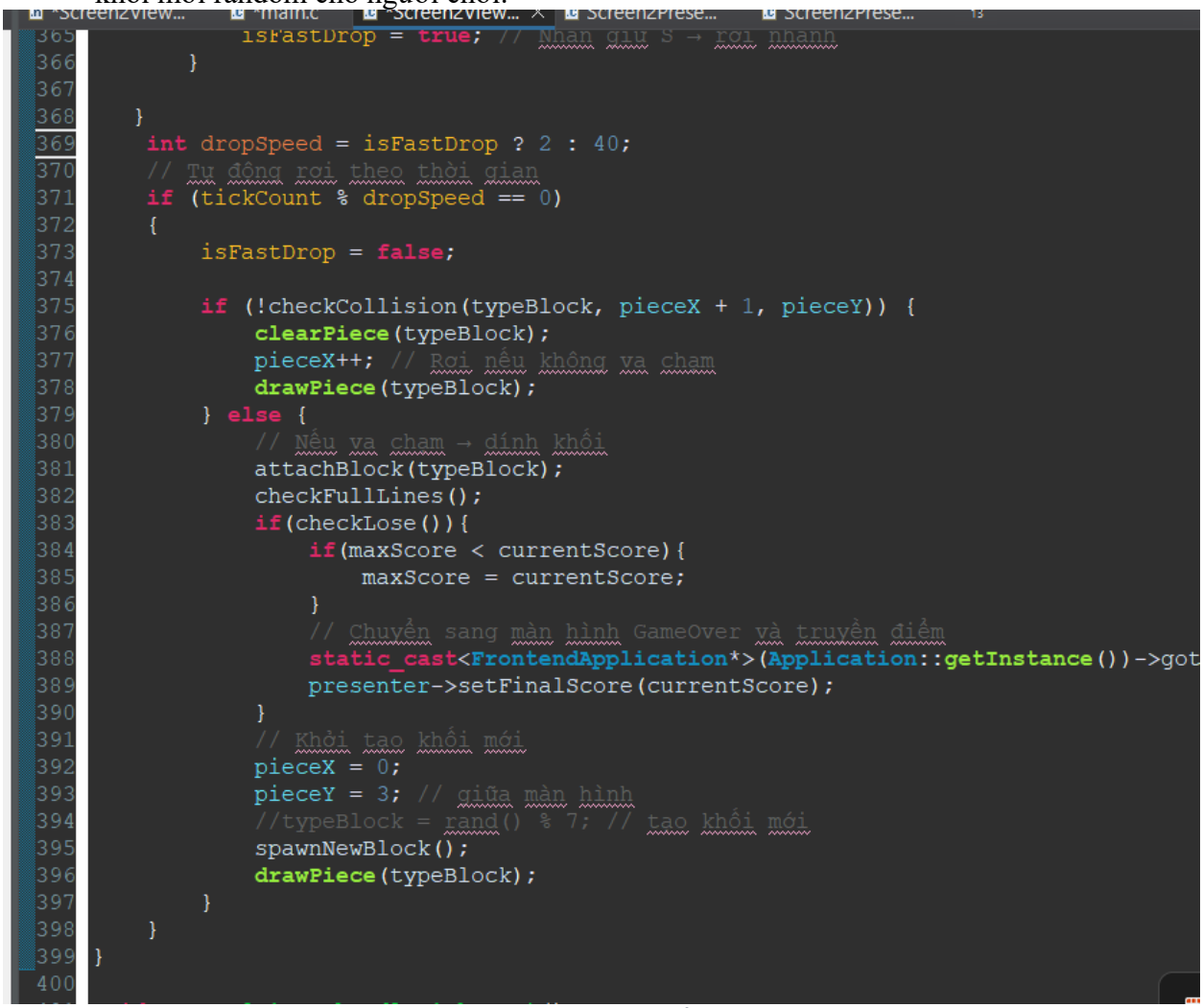
```
void Screen2View::tick()
{
    static int tickCount = 0;
    static bool isFastDrop = false;
    tickCount++;
    uint8_t res;
   int maxEvents = 5; // Giới hạn xử lý tối đa 5 sự kiện mỗi tick
   while (maxEvents-- > 0 && osMessageQueueGet(myQueue01Handle, &res, NULL, 0) ==
       if (res == 'R') {
           clearPiece(typeBlock);
           pieceY++;
           bool valid = checkBlockRight(typeBlock, pieceY);
           if (!valid) pieceY--;
           drawPiece(typeBlock);
       }
       else if (res == 'L') {
           clearPiece(typeBlock);
           pieceY--;
           bool valid = checkBlockLeft(typeBlock, pieceY);
           if (!valid) pieceY++;
           drawPiece(typeBlock);
       }
       else if (res == 'T') {
           clearPiece(typeBlock);
           rotatePiece(typeBlock);
           drawPiece(typeBlock);
       }
       else if (res == 'S') {
           isFastDrop = true; // Nhấn giữ S → rơi nhanh
       }

   }
    int dropSpeed = isFastDrop ? 2 : 40;
    // Tự động rơi theo thời gian
    if (tickCount % dropSpeed == 0)
    {
        isFastDrop = false;

```

Figure 18: tick()

- Tiếp đó sẽ check xem có va chạm với đáy hay khối khác hay không, nếu không thì tiếp tục cho khối rơi xuống dưới. Nếu có thì sẽ đính khối vào đáy (hoặc khối khác)., sau đó check xem có hàng nào full hay không (có thì sẽ xóa và tăng điểm cho người chơi), tiếp đến sẽ check xem khối nào chạm đỉnh hay không, rồi cập nhật lại điểm chơi cao nhất của người chơi, chuyển người chơi sang màn hình game over. Nếu mà không thua thì sẽ sinh khối mới random cho người chơi.



```
            isFastDrop = true; // Nhấn giữ S → rơi nhanh
        }

    }
    int dropSpeed = isFastDrop ? 2 : 40;
    // Tự động rơi theo thời gian
    if (tickCount % dropSpeed == 0)
    {
        isFastDrop = false;

        if (!checkCollision(typeBlock, pieceX + 1, pieceY)) {
            clearPiece(typeBlock);
            pieceX++; // Rơi nếu không va chạm
            drawPiece(typeBlock);
        } else {
            // Nếu va chạm → đính khối
            attachBlock(typeBlock);
            checkFullLines();
            if(checkLose()){
                if(maxScore < currentScore){
                    maxScore = currentScore;
                }
                // Chuyển sang màn hình GameOver và truyền điểm
                static_cast<FrontendApplication*>(Application::getInstance())->got
                presenter->setFinalScore(currentScore);
            }
            // Khởi tạo khối mới
            pieceX = 0;
            pieceY = 3; // giữa màn hình
            //typeBlock = rand() % 7; // tạo khối mới
            spawnNewBlock();
            drawPiece(typeBlock);
        }
    }
}
```

Figure 19: tick() (tiếp)

- Hàm này sẽ gọi liên tục hàm tick() để cập nhật giao diện trên màn hình.

```
void Screen2View::handleTickEvent()
{
    tick(); // Gọi tick() mỗi frame
}
```

Figure 20: handleTickEvent()

CHƯƠNG 4: ĐÁNH GIÁ TỔNG QUAN

1. Giao diện người dùng

Figure 21: Màn hình home của game

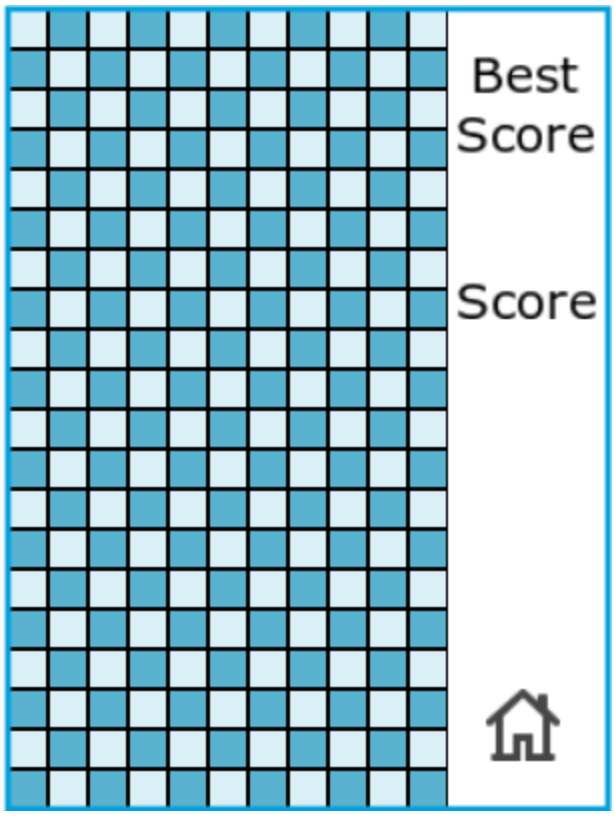


Figure 22: Màn hình chính khi chơi game

Figure 23: Màn hình game over

2. Đánh giá hệ thống

Trong khuôn khổ đồ án môn học, nhóm chúng em đã thiết kế được mô hình đạt được các yêu cầu đề ra:

a) Đánh giá chức năng

- Các khối (I, J, L, O, S, T, Z) sinh ra ngẫu nhiên(sử dụng random hardware generator) và rơi xuống từ trên: Đáp ứng tốt
- Người chơi điều khiển được khối rơi sang trái, phải, xoay và rơi nhanh: Đáp ứng tốt
- Khối dừng lại đúng khi chạm đáy hoặc khối khác: Đáp ứng tốt
- Tự động kiểm tra và xóa hàng đầy, tăng điểm: Đáp ứng tốt
- Tốc độ game tăng theo thời gian: Chưa đáp ứng được
- Kết thúc khi khối chạm đỉnh: Đáp ứng tốt
- Lưu điểm cao nhất và hiển thị điểm hiện tại của người chơi: Đáp ứng tốt

b) Đánh giá phi chức năng

- Game chạy mượt, phản hồi nhanh, không giật lag: Ổn định
- Không bị crash, lỗi logic, văng chương trình: Ổn định
- Dây cắm chắc chắn vào board: Chưa được ổn định

c) Giao diện và trải nghiệm người dùng

- Các ô vuông được chia đều, dễ nhìn: Ổn(Cần cải thiện màu của background)
- Màu sắc các khối khác nhau giúp người chơi dễ phân biệt: Tốt
- Hiển thị điểm số và điểm cao nhất của người chơi (có tạo cảm giác muốn chơi tiếp, chinh phục điểm cao): Tốt
- Nút điều khiển dễ dùng: Tốt

PHỤ LỤC

1. Phân công nhiệm vụ

Nhiệm vụ	Người thực hiện
Tìm hiểu và lên ý tưởng trò chơi	Đinh Quốc Đạt
Xử lý tín hiệu nút bấm	Nguyễn Tuấn Đạt
Tìm hiểu thu thập hình ảnh đồ họa	Đinh Quốc Đạt
Thiết kế đồ họa bằng TouchGFX	Đinh Quốc Đạt Nguyễn Tuấn Đạt
Xử lý logic và hiển thị trò chơi	Nguyễn Tuấn Đạt Trần Hữu Huy
Viết báo cáo thành phần	Đinh Quốc Đạt Nguyễn Tuấn Đạt Trần Hữu Huy
Tổng hợp báo cáo và in quyển đồ án	Nguyễn Tuấn Đạt

2. Tài liệu tham khảo

- Bài giảng học phần "Hệ nhúng" – IT4210 của TS. Ngô Lam Trung.

3. Mã nguồn (source code) và các resource của đồ án

- Mã nguồn chương trình: https://github.com/tuandat1110/Tetris.git
- IDE lập trình: STM32CubeIDE
- Công cụ vẽ sơ đồ, biểu đồ: Draw.io