ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÀI TẬP LỚN
MÔN HỆ NHÚNG

Đề tài: XÂY DỰNG TRÒ CHƠI 2048 TRÊN KIT PHÁT TRIỂN STM32F429I-DISC1

Giảng viên: Ngô Lam Trung

Lớp: 157539

Nhóm sinh viên thực hiện: Nhóm 6

STT	Họ và tên	MSSV
1	Nguyễn Xuân Nhân	20225217
2	Nguyễn Tùng Dương	20225300
3	Hoàng Đức Khải	20225341

Hà Nội, tháng 6 năm 2025

Nội dung

I. Giới thiệu đề tài

1. Mô tả

- 2048 là một trò chơi giải đố dạng bảng (puzzle game) được phát triển bởi Gabriele Cirulli vào năm 2014. Trò chơi nhanh chóng trở nên phổ biến nhờ vào luật chơi đơn giản nhưng đầy tính chiến lược và gây nghiện. Mục tiêu của người chơi là kết hợp các ô số để tạo ra ô có giá trị 2048.
- Trò chơi sẽ được xây dựng trên kit STM32F429I-DISC1 kết hợp với thư viện đồ hoạ TouchGFX. Bộ 32F429IDISCOVERY tận dụng khả năng của các vi điều khiển hiệu suất cao STM32F429, cho phép người dùng dễ dàng phát triển các ứng dụng phong phú với giao diện người dùng đồ họa tiên tiến.

2. Yêu cầu

2.1. Yêu cầu chức năng

- Trò chơi diễn ra trên một bảng vuông có kích thước mặc định 4x4
- Mỗi lượt, người chơi sử dụng joystick chỉnh theo 4 hướng: trái, phải, lên, xuống.

Khi này, sẽ có âm thanh từ buzzer báo hiệu.

- Tất cả các ô sẽ di chuyển theo hướng được chọn. Khi hai ô có cùng giá trị chạm nhau, chúng sẽ hợp nhất thành một ô có giá trị gấp đôi.
- Sau mỗi lượt đi, một ô mới (có giá trị 2 hoặc 4) sẽ xuất hiện ngẫu nhiên tại một vị trí trống.

Trò chơi kết thúc khi:

- Người chơi tạo được ô 2048 (thắng), lúc này sẽ có tùy chọn chơi tiếp hoặc chơi màn mới.
- Không còn ô trống và không thể thực hiện bất kỳ bước hợp nhất nào (thua).

2.2. Yêu cầu phi chức năng

- Giao diện thân thiện, dễ sử dụng, phản hồi nhanh
- Mã nguồn có khả năng mở rộng, dễ bảo trì

II. Thiết kế

1. Phân chia chức năng

1.1. Chức năng phần cứng

- Vi điều khiển STM32F429ZIT6: Vi điều khiển đóng vai trò trung tâm trong việc xử lý logic cốt lõi của trò chơi, chịu trách nhiệm chính cho việc thực thi toàn bộ mã chương trình – từ các phép tính logic cho đến việc điều phối các thao tác vào/ra (I/O). Ngoài ra, nó cũng đảm nhận việc lưu trữ chương trình, dữ liệu tĩnh và động, cũng như trạng thái hiện tại của game thông qua bộ nhớ Flash và RAM. Bên cạnh đó, vi điều khiển còn đảm bảo khả năng giao tiếp với các thiết bị ngoại vi thông qua các chuẩn như GPIO, ADC, TIM, SPI và I2S.
- Màn hình LCD: Hiển thị đồ hoạ, nhận dữ liệu hình ảnh (pixel, màu sắc) từ vi điều khiển và chuyển đổi thành tín hiệu quang học để hiển thị bàn chơi, điểm số và các thông báo.
- Module Joystick: Tạo các tín hiệu điện áp analog trong các biến trở tương ứng với vị trí trục X, Y.
- Module Buzzer: Phát âm thanh mỗi khi người chơi điều khiển joystick.

1.2. Chức năng phần mềm

Phần mềm đóng vai trò xử lý tất cả các logic nghiệp vụ và quản lý tương tác với phần cứng. Các module phần mềm chính bao gồm:

2. Thiết kế phần cứng

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:

2.1. 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 180MHz
- 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. 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.

2.3. 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.

2.4. 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.

2.5. 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.

2.6. 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.

2.7. Các thành phần ngoại vi khác

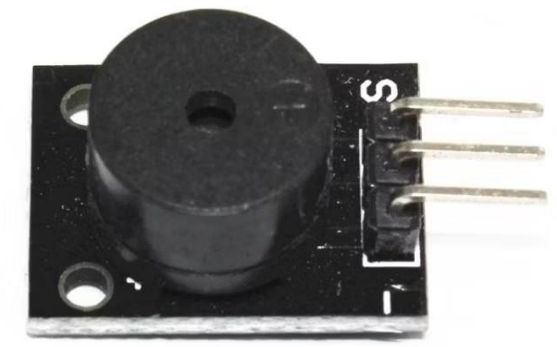
- Joystick: thiết bị điều khiển đầu vào (input device), thường dùng để điều khiển chuyển động theo 2 chiều.
- Buzzer: Phát âm thanh trong khi chơi trò chơi.

Kit phát triển STM32F429I-DISC1

Joystick

Các chân của module Joystick:

- GND nối đất
- VCC nối với nguồn 3,3V
- VRx nối với PC3 (hadc1)
- VRy nối với PA5 (hadc2)



Passive Buzzer

Module Passive Buzzer có 3 chân:

- Chân '-' là GDN nối đất
- Chân ở giữa, kí hiệu '+' là VCC nối nguồn 3.3V
- Chân 'S' là Signal, nối với PA7 để truyền tín hiệu điều khiển âm thanh

3. Thiết kế phần mềm

3.1. Xử lý Input từ Joystick:

- Trong file main.c, khởi tạo một hàng đợi 'myQueue01Handle'. Hàm 'StartDefaultTask()' liên tục polling tín hiệu từ module Joystick, chuyển đổi tín hiệu hành các ký tự 'W, S, A, D' tương ứng với các hành động vuốt lên, vuốt xuống, vuốt sang trái, vuốt sang phải. Các ký tự này sau đó được chuyển vào hàng đợi myQueue01Handle.

```c
/* USER CODE BEGIN PV */
uint8_t isRevD = 0; /* Applicable only for STM32F429I DISCOVERY REVD and above */

osMessageQueueId_t myQueue01Handle;
const osMessageQueueAttr_t myQueue01_attributes = {
        .name = ".myQueue01"
};
/* USER CODE END PV */
```

```c
char getDirection(uint16_t x, uint16_t y)
{
    // Vùng trung tâm nằm trong khoảng 1700–2300 với 3V
    bool cX = (x > 1700 && x < 2300);
    bool cY = (y > 1700 && y < 2300);

    // X là di chuyển theo chiều dọc
    if (!cX) {
        if (x < 1000) return 'W'; // di chuyển lên
        if (x > 3000) return 'S'; //di chuyển xuống
    }
    // Y là di chuyển theo chiều ngang
    if (!cY) {
        if (y < 1000) return 'D'; // di chuyển sang phải
        if (y > 3000) return 'A'; // di chuyển sang trái
    }

    return 0;
}
/* USER CODE END 4 */
```

```c
void StartDefaultTask(void *argument)
{
  /* USER CODE BEGIN 5 */
    char last_move = 0;
  /* Infinite loop */
  for(;;)
  {
      // Đọc ADC
    HAL_ADC_Start(&hadc1);
    HAL_ADC_Start(&hadc2);
    HAL_ADC_PollForConversion(&hadc1, 10);
    HAL_ADC_PollForConversion(&hadc2, 10);
    uint16_t URX = HAL_ADC_GetValue(&hadc1);
    uint16_t URY = HAL_ADC_GetValue(&hadc2);

    char move = getDirection(URX, URY);

    if (move != 0 && move != last_move)
    {
        last_move = move;
        osMessageQueuePut(myQueue01Handle, &move, 0, 0);
    }

    // Nếu Joystick đã về giữa, reset lại vị trí center
    if (move == 0)
    {
        last_move = 0;
    }

    osDelay(100); // tránh gửi liên tục
  }
  /* USER CODE END 5 */
}
```

- Trong file Screen2View.cpp, hàm tickEvent lấy dữ liệu từ myQueue01Handle, thực hiện các hành động tương ứng (slideUp(), slideDown(), slideRight(), slideLeft() và cập nhật lên màn hình (updateBoard()).

```
382 void Screen2View::tickEvent()
383 {
384     uint8_t res;
385     if (osMessageQueueGetCount(myQueue01Handle) > 0)
386     {
387         osMessageQueueGet(myQueue01Handle, &res, NULL, osWaitForever);
388 
389         bool boardChanged = false;
390 
391         switch (res)
392         {
393             case 'W': boardChanged = slideUp();  break;
394             case 'S': boardChanged = slideDown(); break;
395             case 'A': boardChanged = slideLeft(); break;
396             case 'D': boardChanged = slideRight(); break;
397         }
398 
399         if (boardChanged)
400         {
401             playSound(300, 30); // chỉ âm di chuyển
402             addNewTile();
403             updateBoard();
404             updateScore();
405         }
406 
407         if (isGameOver())
408         {
409             container_game_over.setVisible(true);
410             container_game_over.invalidate();
411             updateBestScore();
412             playGameOverSound();
413         }
414     }
415 }
```

3.2. Logic Game:

Khởi tạo các trạng thái hiển thị và logic trò chơi:

- int gameBoard[4][4]: Bảng dữ liệu chính của trò chơi, biểu hiện trạng thái hiện tại của lưới 4x4
- Int score: Biến hiển thị điểm cho lần chơi hiện tại của người chơi
- Int bestScore: Biến lưu điểm cao nhất của người chơi vào bộ nhớ

```
protected:
    int gameBoard[4][4];
    int score;
    int bestScore;
```

- Hàm sinh ngẫu nhiên giá trị: Hàm addNewTile có nhiệm vụ thêm 1 ô mới (2 hoặc 4) vào 1 ô trống ngẫu nhiên trong lưới 4x4. Đầu tiên, hàm sẽ kiểm tra số lượng ô trống, nếu như không còn ô nào thì thoát. Sau khi đã xác định được các ô trống, chọn 1 vị trí ngẫu nhiên thông qua HAL_RNG_GenerateRandomNumber(). Khi đến đúng vị trí đó, tiến hành sinh giá trị 2 hoặc 4 với tỉ lệ 90% là 2, 10% là 4

```
void Screen2View::addNewTile()
{
    int emptyCount = 0;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (gameBoard[i][j] == 0)
            {
                emptyCount++;
            }
        }
    }

    if (emptyCount == 0) return;

    // Sử dụng RNG phần cứng để chọn vị trí ngẫu nhiên
#ifdef SIMULATOR
    int newTilePos = std::rand() % emptyCount; // Không cần randomValue trong simulator
#else
    uint32_t randomValue;
    if (HAL_RNG_GenerateRandomNumber(&hrng, &randomValue) != HAL_OK)
    {
        randomValue = 0; // Xử lý lỗi
    }
    int newTilePos = randomValue % emptyCount;
#endif

    int currentEmpty = -1;

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (gameBoard[i][j] == 0)
            {
                currentEmpty++;
                if (currentEmpty == newTilePos)
```

```
                if (currentEmpty == newTilePos)
                {
#ifdef SIMULATOR
                    int r = std::rand() % 10;
#else
                    uint32_t randomValue;
                    if (HAL_RNG_GenerateRandomNumber(&hrng, &randomValue) != HAL_OK)
                    {
                        randomValue = 0; // Xử lý lỗi
                    }
                    int r = randomValue % 10;
#endif
                    gameBoard[i][j] = (r < 9) ? 2 : 4;
                    return;
                }
            }
        }
    }
}
```

- Các hàm slideLeft(), slideRight(), slideUp(), slideDown() thay đổi mảng gameBoard[4][4]. Hàm updateBoard() sẽ cập nhật lại giao diện và hiển thị cho người chơi. TextAreaWithOneWildCard* textAreas[4][4] là 1 mảng các con trỏ ánh xạ đến các thành phần văn bản (TextArea) trên giao diện. Box* boxes[4][4] là mảng ánh xạ đến nền của ô vuông, dùng để thay đổi màu nền của ô theo giá trị (2, 4, 8,...). Unicode::UnicodeChar* textBuffers[4][4] là các buffer chứa chuỗi văn bản cần hiển thị trong mỗi TextArea, sử dụng Unicode::snprintf() để chuyển số thành chuỗi hiển thị. Const uint16_t textSizes[4][4] chứa kích thước tối đa cho mỗi textBuffer, đảm bảo snprintf() không ghi quá bộ đệm

```
void Screen2View::updateBoard()
{
    TextAreaWithOneWildcard* textAreas[4][4] = {
        {&text_1_1, &text_1_2, &text_1_3, &text_1_4},
        {&text_2_1, &text_2_2, &text_2_3, &text_2_4},
        {&text_3_1, &text_3_2, &text_3_3, &text_3_4},
        {&text_4_1, &text_4_2, &text_4_3, &text_4_4}
    };
    Box* boxes[4][4] = {
        {&box_1_1, &box_1_2, &box_1_3, &box_1_4},
        {&box_2_1, &box_2_2, &box_2_3, &box_2_4},
        {&box_3_1, &box_3_2, &box_3_3, &box_3_4},
        {&box_4_1, &box_4_2, &box_4_3, &box_4_4}
    };
    Unicode::UnicodeChar* textBuffers[4][4] = {
        {text_1_1Buffer, text_1_2Buffer, text_1_3Buffer, text_1_4Buffer},
        {text_2_1Buffer, text_2_2Buffer, text_2_3Buffer, text_2_4Buffer},
        {text_3_1Buffer, text_3_2Buffer, text_3_3Buffer, text_3_4Buffer},
        {text_4_1Buffer, text_4_2Buffer, text_4_3Buffer, text_4_4Buffer}
    };
    const uint16_t textSizes[4][4] = {
        {TEXT_1_1_SIZE, TEXT_1_2_SIZE, TEXT_1_3_SIZE, TEXT_1_4_SIZE},
        {TEXT_2_1_SIZE, TEXT_2_2_SIZE, TEXT_2_3_SIZE, TEXT_2_4_SIZE},
        {TEXT_3_1_SIZE, TEXT_3_2_SIZE, TEXT_3_3_SIZE, TEXT_3_4_SIZE},
        {TEXT_4_1_SIZE, TEXT_4_2_SIZE, TEXT_4_3_SIZE, TEXT_4_4_SIZE}
    };

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            TextAreaWithOneWildcard& textArea = *textAreas[i][j];
            Box* box = boxes[i][j];
            Unicode::snprintf(textBuffers[i][j], textSizes[i][j], gameBoard[i][j] == 0 ? "" : "%d", gameBoard[i][j]);

            switch (gameBoard[i][j])
            {
                case 0: box->setColor(touchgfx::Color::getColorFromRGB(205, 193, 181)); break;
                case 2: box->setColor(touchgfx::Color::getColorFromRGB(238, 228, 218)); break;
                case 4: box->setColor(touchgfx::Color::getColorFromRGB(237, 224, 200)); break;
                case 8: box->setColor(touchgfx::Color::getColorFromRGB(242, 177, 121)); break;
                case 16: box->setColor(touchgfx::Color::getColorFromRGB(245, 149, 99)); break;
                case 32: box->setColor(touchgfx::Color::getColorFromRGB(246, 124, 95)); break;
                case 64: box->setColor(touchgfx::Color::getColorFromRGB(246, 94, 59)); break;
                case 128: box->setColor(touchgfx::Color::getColorFromRGB(237, 207, 114)); break;
                case 256: box->setColor(touchgfx::Color::getColorFromRGB(237, 204, 97)); break;
                case 512: box->setColor(touchgfx::Color::getColorFromRGB(237, 200, 80)); break;
                case 1024: box->setColor(touchgfx::Color::getColorFromRGB(237, 197, 63)); break;
                case 2048: box->setColor(touchgfx::Color::getColorFromRGB(237, 194, 46)); break;
                default: box->setColor(touchgfx::Color::getColorFromRGB(205, 193, 181)); break;
            }

            textArea.invalidate();
            box->invalidate();
        }
    }
}
```

- Xử lý Game Over: Sử dụng hàm isGameOver() để kiểm tra xem liệu người chơi còn nước đi hợp lệ nào không, thông qua 2 điều kiện: Còn ô trống hay không, còn có thể gộp 2 ô kề nhau hay không (Trên-dưới và trái-phải).

```
bool Screen2View::isGameOver()
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (gameBoard[i][j] == 0) return false;
        }
    }

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (gameBoard[i][j] == gameBoard[i][j + 1]) return false;
        }
    }

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (gameBoard[i][j] == gameBoard[i + 1][j]) return false;
        }
    }

    return true;
}
```

- Hàm Screen2View::tickEvent() là hàm xử lý chính được gọi mỗi chu kỳ (tick) trong game để kiểm tra và xử lý tín hiệu từ hàm hàng đợi điều khiển (Message Queue) - Tức là thao tác từ joystick của người chơi, cập nhật giao diện và kiểm tra điều kiện thua game.

```
382 void Screen2View::tickEvent()
383 {
384     uint8_t res;
385     if (osMessageQueueGetCount(myQueue01Handle) > 0)
386     {
387         osMessageQueueGet(myQueue01Handle, &res, NULL, osWaitForever);
388 
389         bool boardChanged = false;
390 
391         switch (res)
392         {
393             case 'W': boardChanged = slideUp();  break;
394             case 'S': boardChanged = slideDown(); break;
395             case 'A': boardChanged = slideLeft(); break;
396             case 'D': boardChanged = slideRight(); break;
397         }
398 
399         if (boardChanged)
400         {
401             playSound(300, 30); // chỉ âm di chuyển
402             addNewTile();
403             updateBoard();
404             updateScore();
405         }
406 
407         if (isGameOver())
408         {
409             container_game_over.setVisible(true);
410             container_game_over.invalidate();
411             updateBestScore();
412             playGameOverSound();
413         }
414     }
415 }
```

3.3. Xử lý âm thanh:

- Đối với module Buzzer thụ động, thiết lập Timer PWM để phát âm thanh. Trong Timer, TIM3, chọn Channel 2 với chế độ PWM Generation CH2. Chân Signal của Buzzer nối với PA7 của kit.
- Trong file Screen2View, âm thanh được điều khiển phát ra bởi hàm playSound() với hai tham số là tần số và khoảng thời gian phát. Trong hàm tickEvent(), mỗi lần người chơi thực hiện một nước đi hợp lệ, sẽ gọi đến hàm playSound phát ra âm thanh. Nếu người chơi thất bại, không còn nước đi nào nữa một popup thông báo ‘Game Over’ hiện ra, đồng thời hàm playGameOverSound() điều khiển phát một đoạn nhạc ngắn.

```
362 void Screen2View::playSound(uint32_t freq, uint32_t durationMs)
363 {
364     uint32_t period = 1000000 / freq; // Timer chạy ở 1MHz
365     __HAL_TIM_SET_AUTORELOAD(&htim3, period - 1);
366     __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, period / 2); // 50% duty
367
368     HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);
369     HAL_Delay(durationMs);
370     HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_2);
371 }
```

```
373 void Screen2View::playGameOverSound()
374 {
375     const uint32_t node[] = {880, 660, 440, 330, 220}; // A5-E5-A4-E4-A3
376     for (int i = 0; i < sizeof(node)/sizeof(node[0]); i++) {
377         playSound(node[i], 120);
378         HAL_Delay(50);
379     }
380 }
```

3.4. Lưu điểm cao:

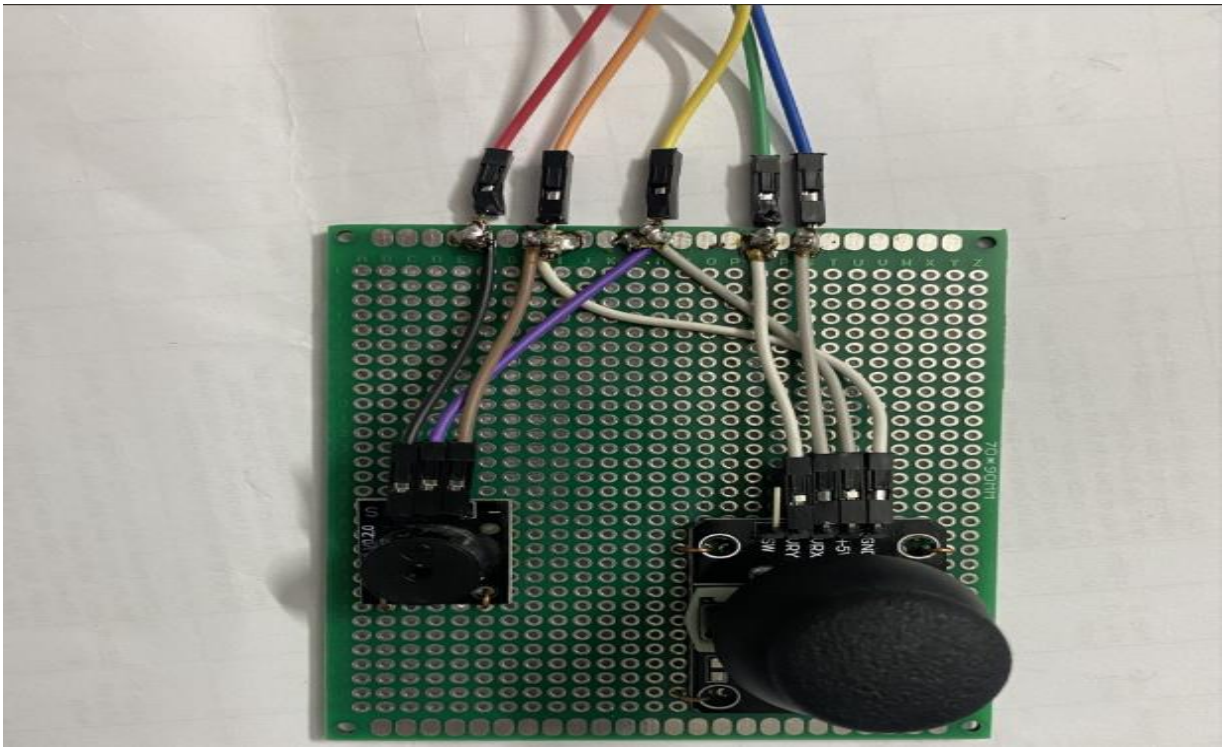
- Tính năng này đảm bảo giữ lại điểm cao nhất (bestScore) ngay cả khi người chơi tắt nguồn hoặc nhấn nút reset thiết bị. Nhờ đó, khi được cấp nguồn trở lại, game vẫn hiển thị đúng điểm cao nhất đã đạt được trước đó.
- Điểm cao nhất được lưu tại sector 11 (`0x080F0000`) trong bộ nhớ Flash của STM32F429ZIT6. Đây là một vùng nhớ không bị xóa khi tắt nguồn, thích hợp để lưu dữ liệu vĩnh viễn.
- Hàm saveBestScoreToFlash() lưu điểm cao nhất mà người chơi đạt được vào Flash. Hàm loadBestScoreFronFlash() đọc giá trị điểm cao nhất từ địa chỉ đã được chỉ định. Nếu chưa có điểm được lưu, trả về bestScore = 0, ngược lại lấy giá trị đó và hiển thị lên màn hình.

```
433 // Hàm lưu bestScore vào Flash
434 #ifndef SIMULATOR
435 void Screen2View::saveBestScoreToFlash()
436 {
437     HAL_FLASH_Unlock(); // Mở khóa Flash
438     FLASH_EraseInitTypeDef eraseInit;
439     uint32_t sectorError;
440 
441     eraseInit.TypeErase = FLASH_TYPEERASE_SECTORS;
442     eraseInit.Sector = FLASH_SECTOR_11; // Sector 11 (0x080F0000)
443     eraseInit.NbSectors = 1;
444     eraseInit.VoltageRange = FLASH_VOLTAGE_RANGE_3;
445 
446     if (HAL_FLASHEx_Erase(&eraseInit, &sectorError) != HAL_OK)
447     {
448         Error_Handler();
449         return;
450     }
451 
452     if (HAL_FLASH_Program(FLASH_TYPEPROGRAM_WORD, FLASH_ADDR, bestScore) != HAL_OK)
453     {
454         Error_Handler();
455         return;
456     }
457 
458     HAL_FLASH_Lock(); // Khóa Flash lại
459 }
460 #endif
```

```
462 // Hàm đọc bestScore từ Flash
463 #ifndef SIMULATOR
464 void Screen2View::loadBestScoreFromFlash()
465 {
466     uint32_t value = (*(__IO uint32_t*)FLASH_ADDR);
467 
468         if (value != 0xFFFFFFFF)
469         {
470             bestScore = value;
471         }
472         else
473         {
474             bestScore = 0;
475         }
476 }
477 #endif
```

4. Kiến trúc hệ thống

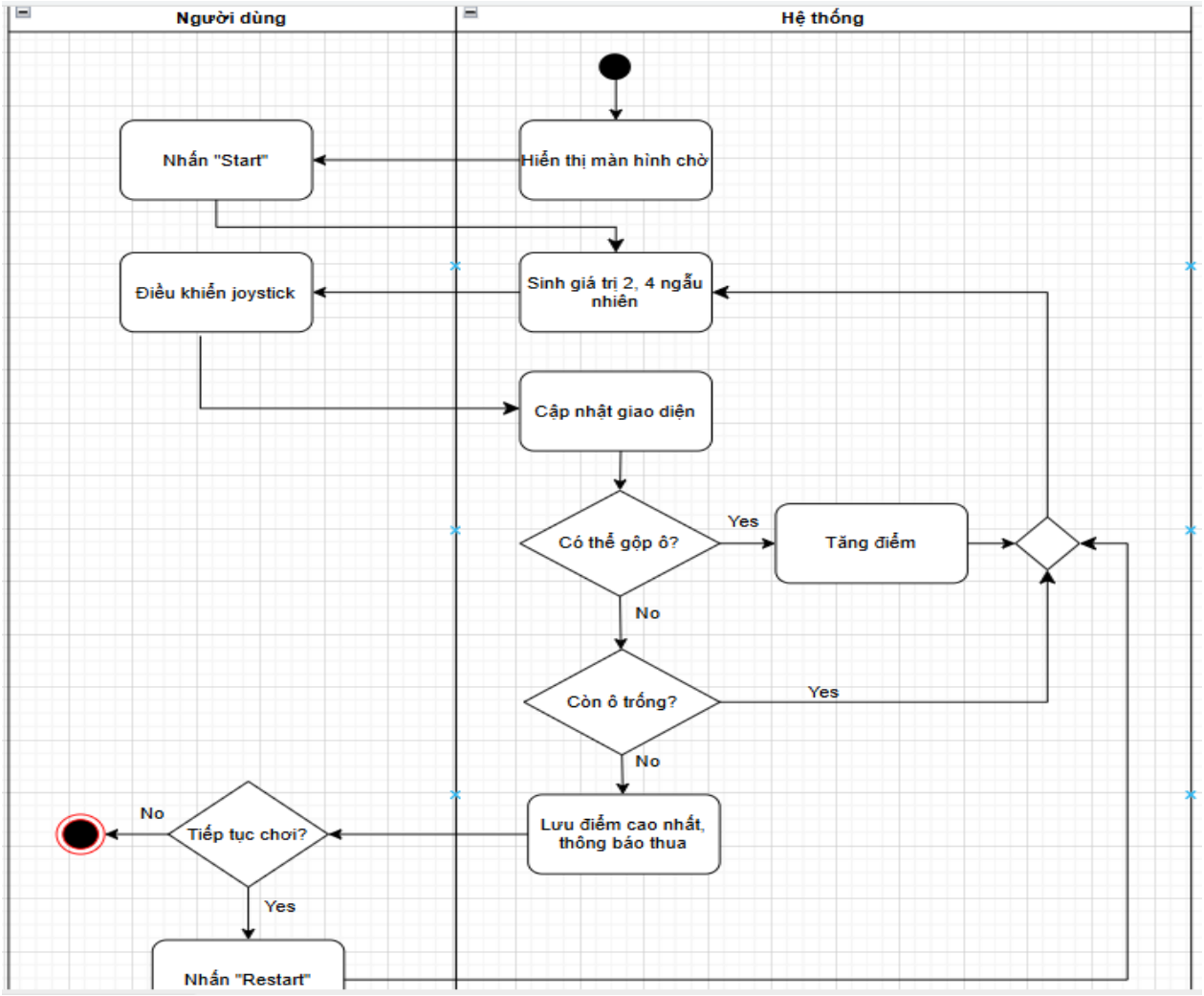
4.1. Mạch nút bấm



Mạch nút bấm

- Mạch thao tác được trang bị 1 joystick giúp người chơi điều khiển thao tác lên, xuống, trái, phải. Ngoài ra, 1 buzzer được tích hợp vào để phát âm thanh phản hồi khi người dùng thao tác nhấn nút, nhằm tăng tính trải nghiệm khi chơi.

4.2. Biểu đồ luồng



Hình 1: Biểu đồ luồng

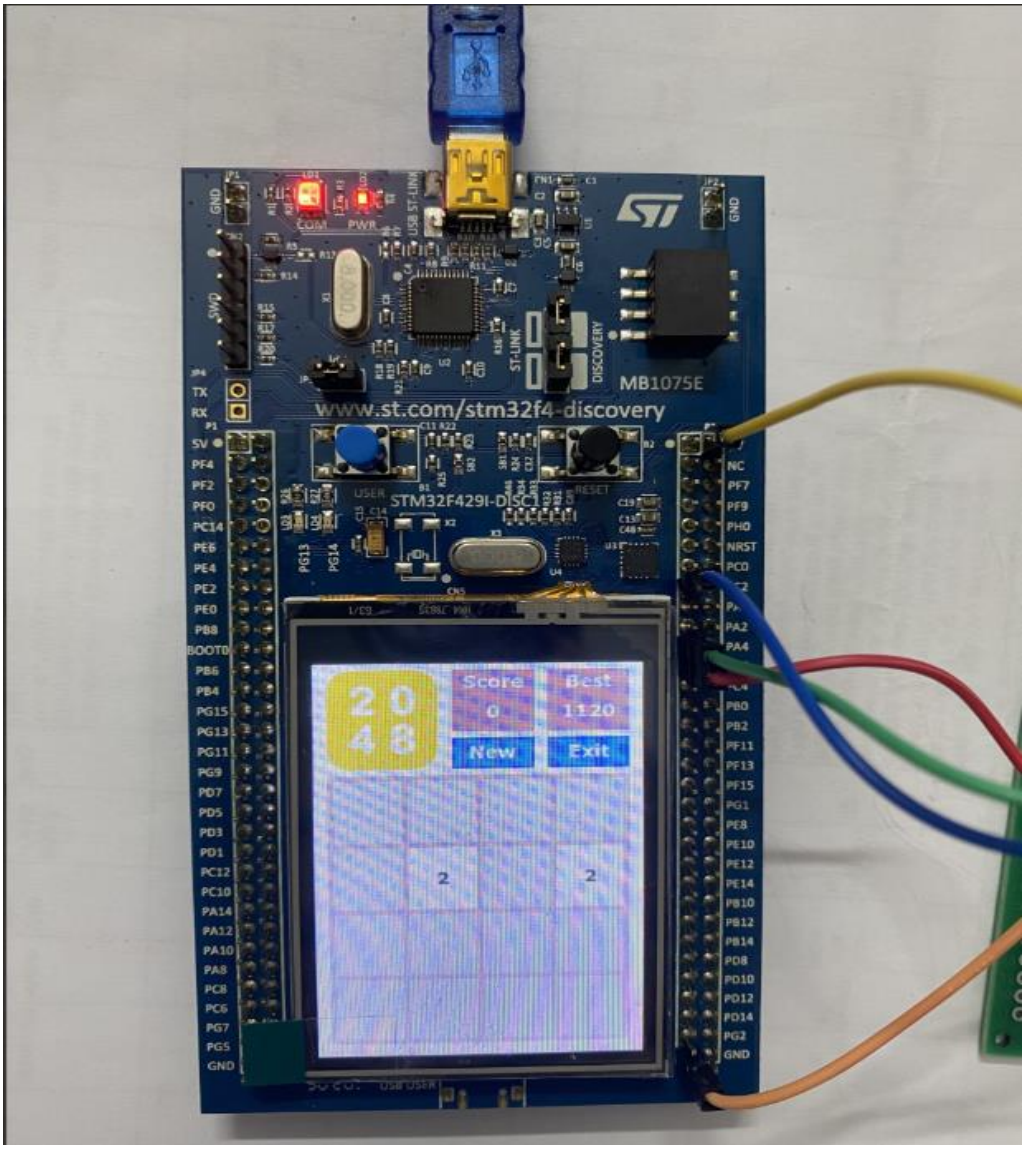
III. Cài đặt/xây dựng hệ thống

1. **Link github:**

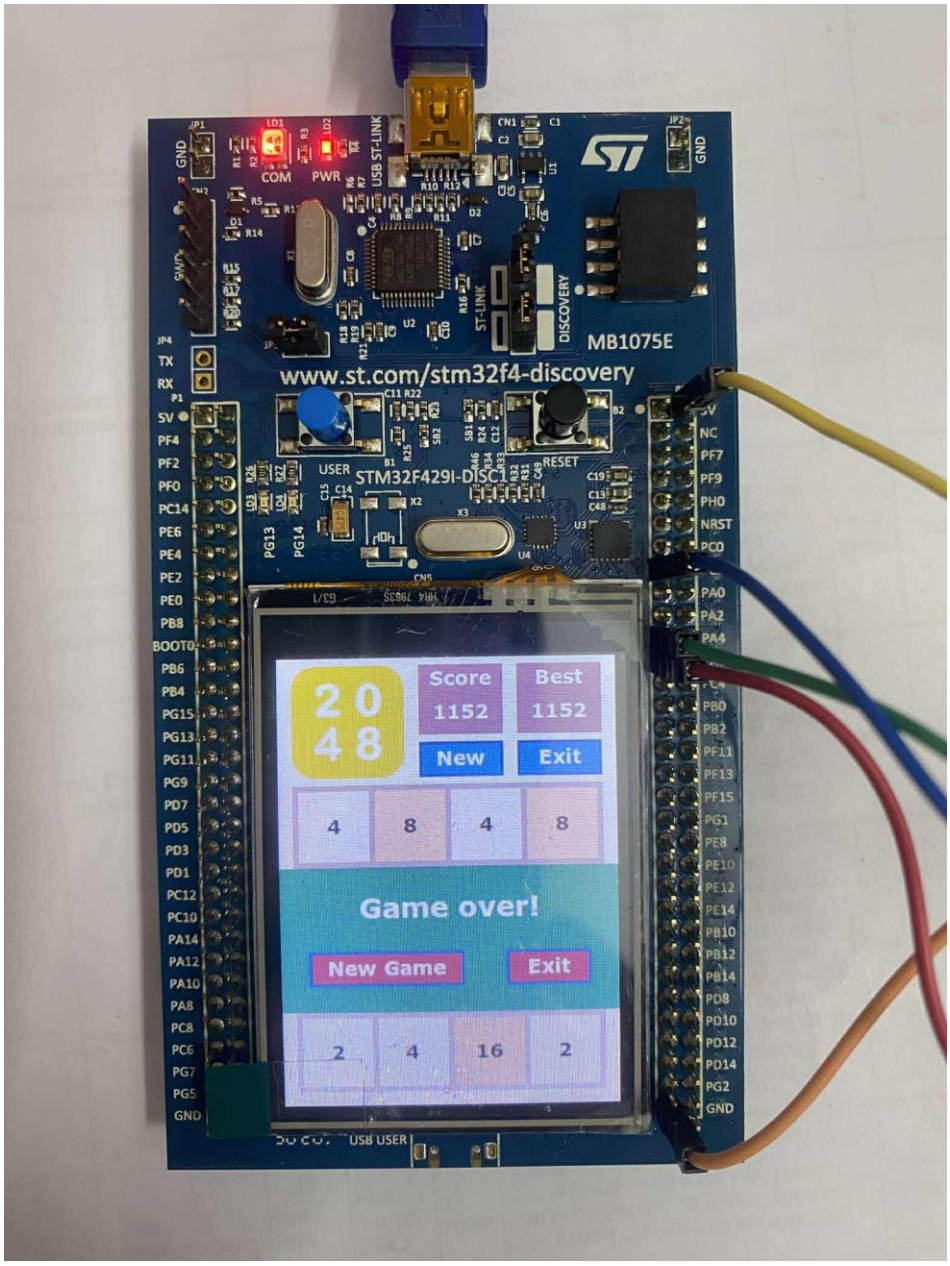
[xnhannn/2048-project]

2. **Hình ảnh & Video demo**

Giao diện khởi động game



Giao diện chính

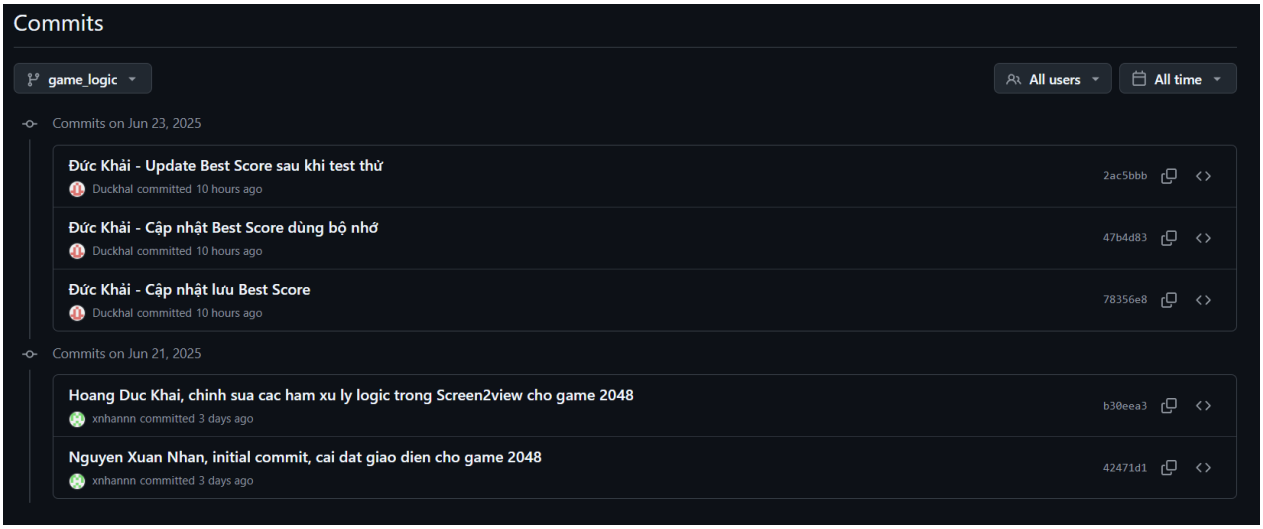


Giao diện Game Over

3. Đóng góp của các thành viên

Thành viên	Nhiệm vụ	Đánh giá
Nguyễn Xuân Nhân	Viết báo cáo Thiết kế giao diện trong TouchGFX Xử lý logic game + lưu điểm Xử lý kết nối joystick, buzzer	Hoàn thành
Hoàng Đức Khải	Viết báo cáo Thiết kế giao diện trong TouchGFX Xử lý logic game + lưu điểm	Hoàn thành
Nguyễn Tùng Dương	Viết báo cáo Thiết kế giao diện Phụ trách phần cứng	Hoàn thành

4. Lịch sử commit của từng thành viên



IV. Đánh giá

1.1. Đánh giá chức năng

- Sinh các giá trị ngẫu nhiên 2, 4: Đáp ứng tốt
- Người chơi điều khiển bằng Joystick để chơi game: Đáp ứng tốt
- Hai ô có cùng giá trị thì gộp vào nhau: Đáp ứng tốt
- Tăng điểm mỗi khi có 2 ô được gộp: Đáp ứng tốt
- Lưu lại điểm cao nhất và hiển thị điểm của người chơi: Đáp ứng tốt

1.2. Đánh giá phi chức năng

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

1.3. Giao diện và trải nghiệm người dùng

- Giao diện dễ nhìn: Tốt
- Các số có các màu tương ứng với các giá trị khác nhau để phân biệt: Tốt
- Điều khiển dễ dàng: Tốt