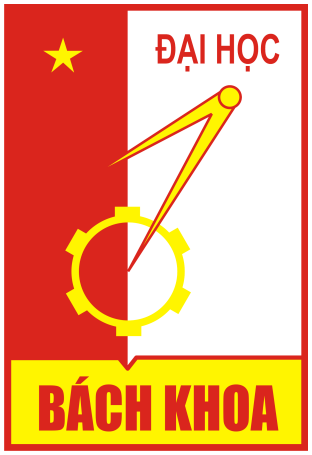
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



EMBEDDED SYSTEM - CAPSTONE PROJECT

XÂY DỰNG GAME MÔ PHỎNG TỪ SPACE INVADERS

Instructor: Ph.D. Ngô Lam Trung

Students: Nguyễn Lan Nhi - 20225991
Bùi Hà My - 20225987

Ha Noi, June 2025

Contents

1 Giới thiệu đề tài

1.1 Mô tả sản phẩm

"Chạy đâu cho thoát" là một trò chơi bắn súng được phát triển trên vi điều khiển STM32F429ZIT6, lấy cảm hứng từ trò chơi kinh điển Space Invaders. Trong game, người chơi điều khiển một nhân vật "Phú ông" ở phần đáy màn hình và có nhiệm vụ "tiêu diệt" hai đứa con - "Cậu cả" và "Cậu út" đang tiến dần xuống bằng cách bắn "đá". Nếu để "địch" chạm đến vạch chia phía dưới màn hình hoặc tiêu diệt hết mạng của "Phú ông", người chơi sẽ thua.

Trò chơi còn có những tính năng như âm thanh sống động, mức độ khó tăng dần và hệ thống điểm cao (high score). Đặc biệt, dự án khai thác các tính năng phần cứng của STM32 như bộ sinh số ngẫu nhiên (hardware RNG), FreeRTOS để quản lý đa luồng, và ngắt (interrupts) để phản hồi nhanh với nút bấm.

1.2 Yêu cầu bài toán

1.2.1 Yêu cầu chức năng

Sản phẩm có những yêu cầu chức năng như sau:

1. **Điều khiển trò chơi**: Sử dụng hai nút bấm vật lý (button) được ghép nối ở chân PG2 và PG3 để điều khiển "Phú ông" di chuyển sang trái và sang phải ở phía dưới màn hình. Ngoài ra, một số màn hình còn được trang bị các nút điều hướng sang các màn hình liên quan.

2. **Giao diện hiển thị trò chơi**: Trò chơi bao gồm 4 màn hình - màn hình Trang chủ (Home), màn hình Giới thiệu (Intro), màn hình Trò chơi chính (Screen1), màn hình Thua cuộc (GameOver) và màn hình Chiến thắng (Victory). Màn hình Trò chơi chính hiển thị đồ họa đơn giản của địch - "Cậu cả" và "Cậu út" xen kẽ, người chơi - "Phú ông", đạn của địch và người chơi - "Ngôi sao" và "Hòn đá", rào chắn - "Đống rơm", điểm số hiện tại, điểm số cao nhất và level. Tất cả các màn hình đều được hiển thị trên LCD ILI9341 (giao tiếp qua SPI) được ghép nối sẵn trên STM32F429ZIT6. Giao diện cập nhật liên tục vị trí và trạng thái của các đối tượng trong trò chơi.

3. **Cơ chế hoạt động của địch**: Các hàng địch di chuyển theo chiều zigzag dần xuống phía dưới màn hình. Sau một khoảng thời gian, sử dụng **sinh số ngẫu nhiên phần cứng (hardware RNG)** để chọn một kẻ địch còn sống và phát đạn từ vị trí đó.Tốc độ di chuyển và tần suất bắn đạn sẽ tăng dần theo level.

4. **Xử lý va chạm**:
   - Đạn người chơi trúng địch: loại bỏ địch, tăng điểm.
   - Đạn (của cả hai phía) va vào rào chắn: trừ một mạng rào chắn.
   - Địch va vào người chơi: trừ một mạng người chơi.
   - Địch chạm vạch chia: người chơi thua, trò chơi kết thúc.

5. **Âm thanh**: Buzzer phát âm thanh khi có va chạm

6. **Điểm số và cấp độ**: Ở trên cùng màn hình hiển thị điểm cao nhất, cấp độ level (tự động tăng mỗi khi có thêm 5 điểm) và điểm số hiện tại của người chơi.
7. **Hệ thống hoạt động đa luồng**: Sử dụng FreeRTOS để quản lý đa nhiệm gồm:
    - Task GUI để xử lý các tác vụ liên quan đến GUI, logic game.
    - Task Default chịu trách nhiệm đọc trạng thái nút bấm và truyền dữ liệu điều khiển qua Queue1 đến task xử lý logic game để điều khiển di chuyển của người chơi.
8. **Tương tác người dùng**: Hiển thị màn hình "Chiến thắng" hoặc "Thất bại", cho phép người chơi restart bằng nút bấm trên màn hình.

1.2.2 Yêu cầu phi chức năng

Trò chơi có những yêu cầu phi chức năng như sau

- **Hiển thị mượt mà:** Giao diện người dùng được xây dựng bằng TouchGFX chạy trên kit STM32F429I-DISC1 sử dụng vi điều khiển STM32F429ZIT6 cho phép hệ thống hiển thị ở tốc độ cao (khoảng 60 FPS), đảm bảo trải nghiệm giao diện người dùng mượt mà và ổn định.
- **Phản hồi điều khiển tức thời:** Các nút điều khiển PG2 (LEFT) và PG3 (RIGHT) được đọc thông qua GPIO kết hợp FreeRTOS task, cho phép hệ thống phản hồi ngay lập tức khi người chơi bấm nút.
- **Tính ổn định cao:** Trong quá trình thử nghiệm thực tế, hệ thống không ghi nhận tình trạng treo, khởi động lại, hoặc lỗi logic trong game. FreeRTOS đảm bảo quản lý tác vụ ổn định và tránh xung đột tài nguyên.
- **Cấu trúc phần mềm rõ ràng, dễ bảo trì:** Mã nguồn được chia module theo chức năng: giao diện (Screen1View), điều khiển (main.c),... giúp việc phát triển và mở rộng về sau trở nên thuận lợi.
- **Khả năng mở rộng:** Thiết kế hiện tại cho phép dễ dàng bổ sung các tính năng nâng cao như lưu điểm cao vào Flash, chia level phức tạp hơn hoặc phát âm thanh đa dạng hơn qua buzzer. Các chức năng này đã được chuẩn bị về mặt phần mềm hoặc phần cứng.

2 Thiết kế hệ thống

2.1 Phân chia chức năng

Chức năng	Phần cứng thực hiện	Phần mềm thực hiện
Điều khiển LEFT/RIGHT	Hai nút nhấn ngoài nối PG2 (LEFT) và PG3 (RIGHT), cấu hình GPIO input	Task `StartDefaultTask` trong FreeRTOS thực hiện polling trạng thái nút, sau đó gửi tín hiệu điều khiển vào queue để xử lý di chuyển nhân vật.
Nút RESET	Nút tích hợp B1 trên board, nối vào RESET_N	Tự động reset vi điều khiển khi nhấn, không xử lý trong phần mềm trò chơi.
Hiển thị giao diện	LCD TFT 2.4", độ phân giải 240x320, kết nối qua RGB	Giao diện xây dựng bằng TouchGFX, hoạt động trong thread `GUI_Task`.
Sinh số ngẫu nhiên	Bộ RNG phần cứng tích hợp trong STM32F429ZIT6	Dùng `HAL_RNG_GenerateRandomNumber()` để chọn ngẫu nhiên enemy còn sống bắn đạn, giúp tăng tính bất ngờ cho trò chơi.
Lưu điểm cao nhất	Không có phần cứng chuyên biệt	Biến `highScore` chỉ tồn tại trong RAM và được cập nhật khi điểm mới vượt qua điểm cũ. Dữ liệu này sẽ mất khi reset hệ thống.
Game logic và level	Không có phần cứng chuyên biệt	Logic điều khiển gameplay, xử lý va chạm, tính điểm và tăng độ khó theo thời gian được xử lý trong `defaultTask`. Tốc độ di chuyển của enemy tăng dần, đóng vai trò như "level".
Phát âm thanh	Active Buzzer (Còi chip) nối PA5, cấu hình GPIO Output Push-pull	Phát hiện sự kiện va chạm để phát âm thanh trong thời gian ngắn

Table 1: Phân chia chức năng phần cứng – phần mềm

2.2 Thiết kế phần cứng

2.2.1 Các module và linh kiện

2.2.1.a Phần cứng của Kit

- **Vi điều khiển (MCU):** STM32F429ZIT6
  - ARM Cortex-M4, 180 MHz
  - 2MB Flash, 256KB RAM
  - Gói chân LQFP144
- **Màn hình LCD:** TFT 2.4 inch QVGA (240x320), 262K màu, điều khiển bởi ILI9341
- **Bộ nhớ ngoài:** 64 Mbit SDRAM (8MB), bus 16-bit
- **Cảm biến:** Gyroscope I3G4250D (3 trục, giao tiếp SPI)
- **Giao tiếp USB:** USB OTG Micro-AB, hỗ trợ Host/Device

- **Mạch nạp/debug:** ST-LINK/V2-B tích hợp, hỗ trợ SWD, Virtual COM port (VCP), và Mass Storage
- **LEDs:** Tổng cộng 6 LED
  - LD1 (COM), LD2 (PWR), LD3-LD4 (User), LD5-LD6 (USB OTG)
- **Nút nhấn:** hai nút nhấn User và Reset
- **Nguồn cấp:** Qua cổng Mini-USB cấp nguồn 3V hoặc 5V

Figure 1: KIT STM32F429I-DISC vi điều khiển STM32F429ZIT6

2.2.1.b Mạch nút bấm

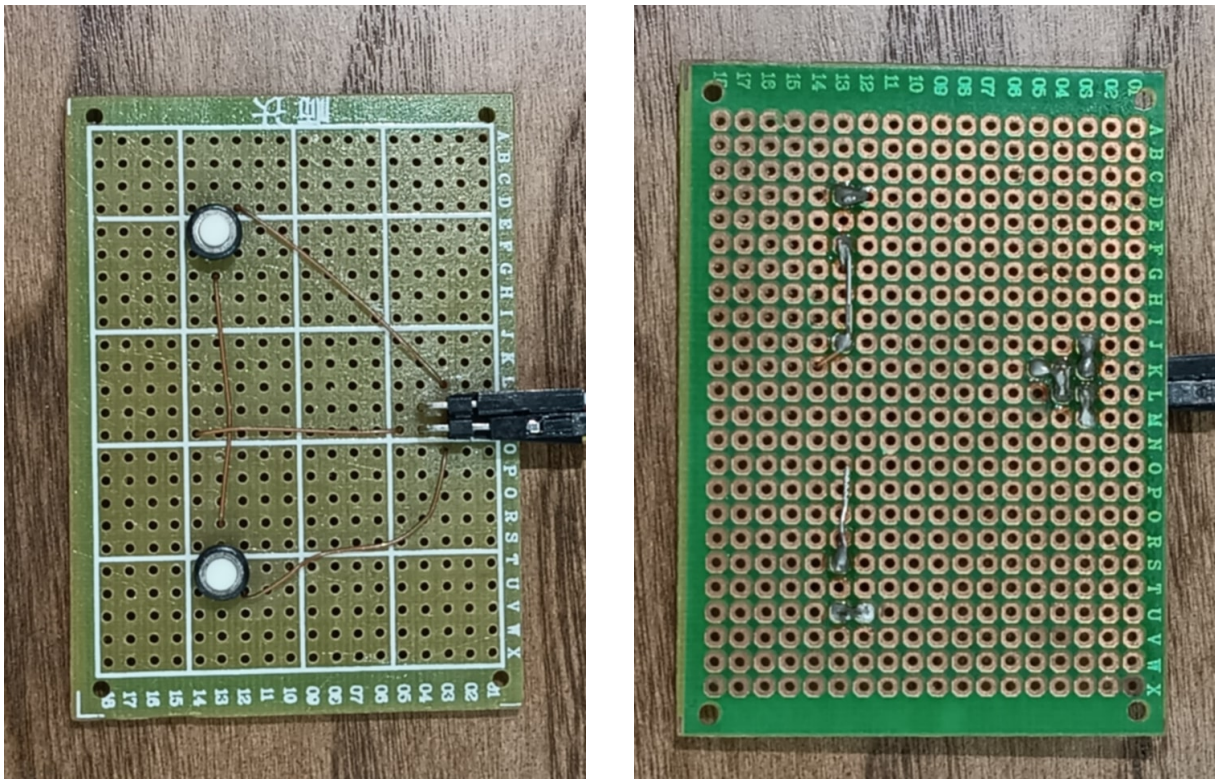


Figure 2: Sơ đồ mạch nút bấm

2.2.1.c Mạch buzzer âm thanh

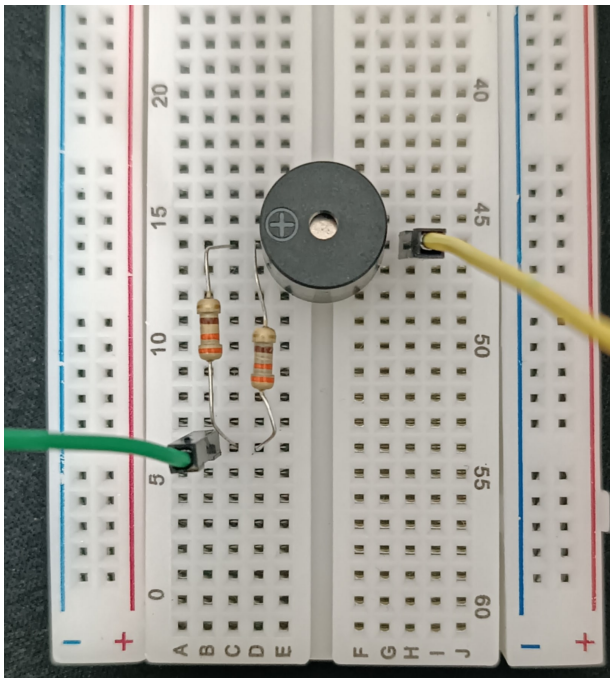


Figure 3: Mạch Active Buzzer âm thanh

2.2.2 Sơ đồ ghép nối logic

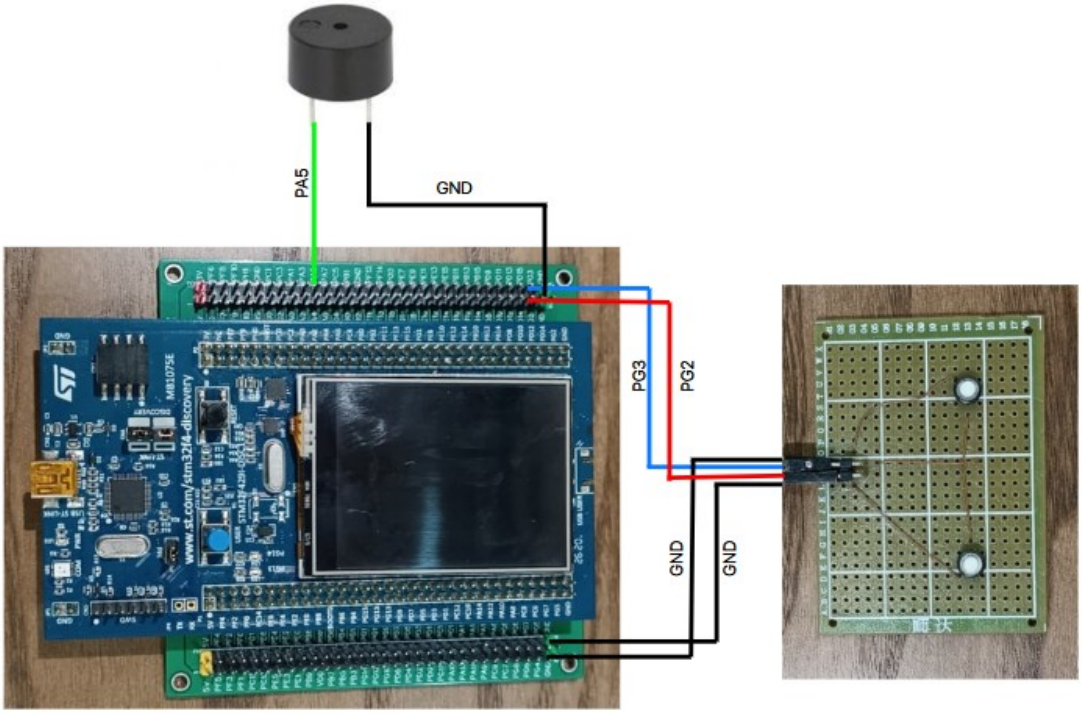


Figure 4: Sơ đồ ghép nối logic

2.2.3 Địa chỉ và tốc độ trao đổi dữ liệu giữa các thành phần

Thành phần	Giao tiếp	Tốc độ / Cấu hình	Địa chỉ / Chi tiết
LCD TFT	LTDC + DMA2D	30–60 fps, cập nhật liên tục	Không yêu cầu địa chỉ
Bộ RNG	Nội bộ	Truy xuất phần cứng, độ trễ $< 1\ \mu s$	Không yêu cầu địa chỉ
GPIO (nút/LED)	GPIO port G/A	Nút đọc theo polling, LED điều khiển trực tiếp	Cấu hình chân trong chương trình

Table 2: Địa chỉ và tốc độ trao đổi dữ liệu giữa các thành phần

2.3 Thiết kế phần mềm

2.3.1 Các module phần mềm theo chức năng

Module	Tệp liên quan	Chức năng chính
Khởi tạo phần cứng	`main.c`	- Cấu hình `GPIO`, `SPI`, `I2C`, `RNG`, `LTDC`, `SDRAM`, `Timer (TIM3)` - Thiết lập đồng hồ hệ thống
Khởi tạo TouchGFX	`main.c`	- Gọi `MX_TouchGFX_Init()` và tạo `GUI_Task` để khởi động giao diện
Quản lý RTOS	`main.c`	- Tạo thread `defaultTask` để xử lý nút nhấn - Tạo queue `myQueue1` dùng chung với GUI
Xử lý nút nhấn vật lý	`main.c` (`defaultTask`)	- Đọc GPIO PG2 (trái), PG3 (phải) - Gửi tín hiệu tương ứng (cmd=1 or cmd=2) vào `myQueue1Handle`
Giao diện người dùng	`Screen1View.cpp`	- Giao diện chính của trò chơi - Hiển thị điểm số, mạng, level, nhân vật và enemy, v.v
Xử lý logic trò chơi	`Screen1View.cpp`	- Cập nhật vị trí enemy - Quản lý trạng thái `enemy`, `tulong`, `straw`
Xử lý chuyển động	`Screen1View.cpp`	- Điều khiển `tulong` dựa vào hàng đợi RTOS - Enemy di chuyển zigzag - Đạn `star`/`stone` bay
Quản lý bắn & va chạm	`Screen1View.cpp`	- Hệ thống bắn enemy và `tulong` - Kiểm tra va chạm đạn với `enemy`, `tulong`, `straw`
Điểm số & level	`Screen1View.cpp`	- Tính điểm mỗi lần bắn trúng `enemy`, tránh được đạn - Tăng level sau mỗi 5 điểm - Tăng độ khó theo level
Xử lý kết thúc game	`Screen1View.cpp`	- Phát hiện `enemy` chạm line hoặc `tulong` hết máu/ `tulong` tiêu diệt hết enemies và vẫn còn máu - Chuyển sang màn hình Victory hoặc GameOver
Xử lý âm thanh	`Screen1View.cpp`	- Âm thanh khi đạn va chạm với đối tượng trong game (`tulong`, enemies, `straw`)
Random phần cứng	`Screen1View.cpp`	- Dùng RNG của STM32 (`HAL_RNG_GenerateRandomNumber`) để chọn enemy và phát đạn từ vị trí đó

Table 3: Phân chia các module phần mềm theo chức năng trong dự án game

2.3.2 Sơ đồ khối hệ thống

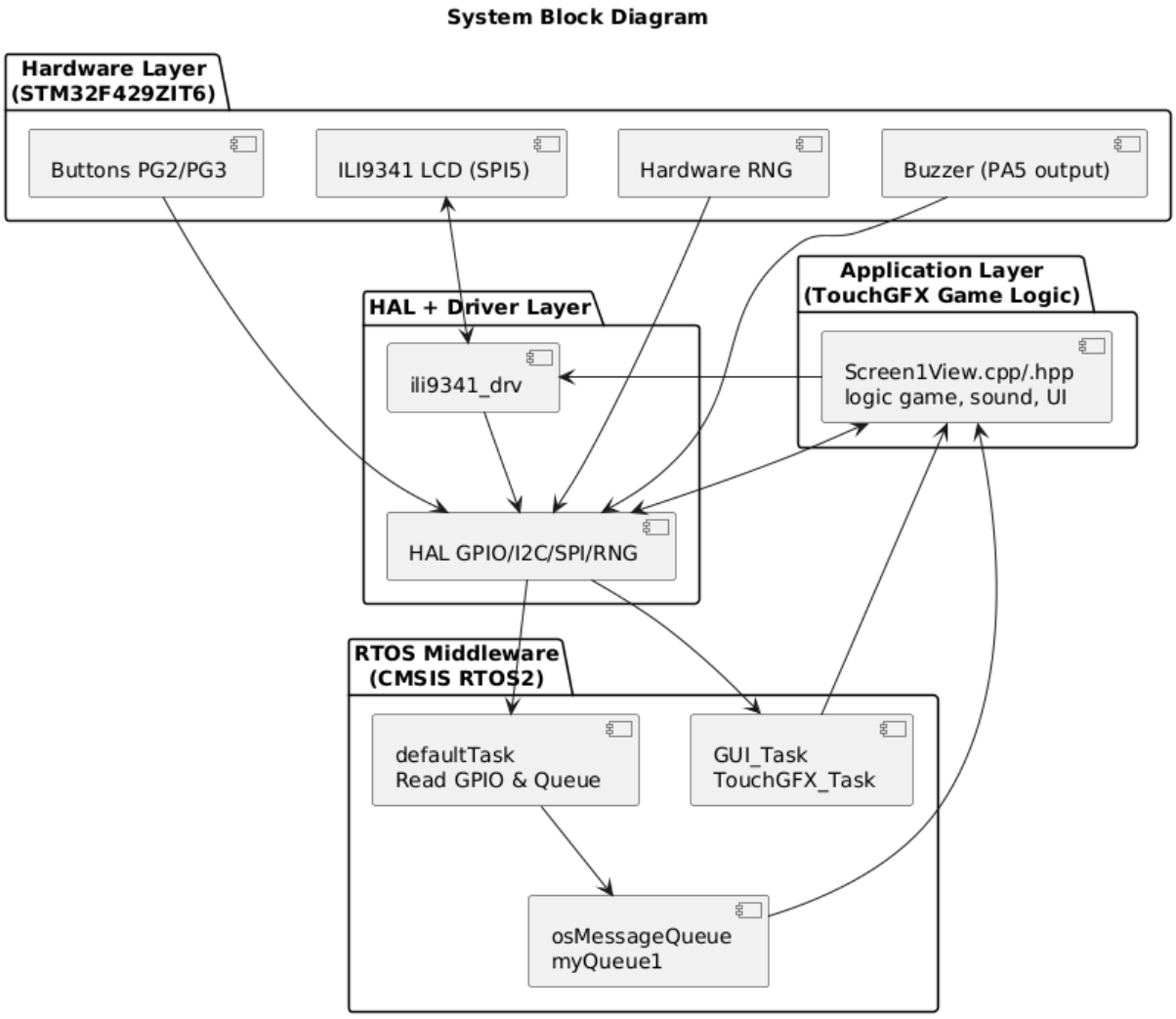


Figure 5: Sơ đồ khối hệ thống

2.3.3 Biểu đồ luồng hoạt động

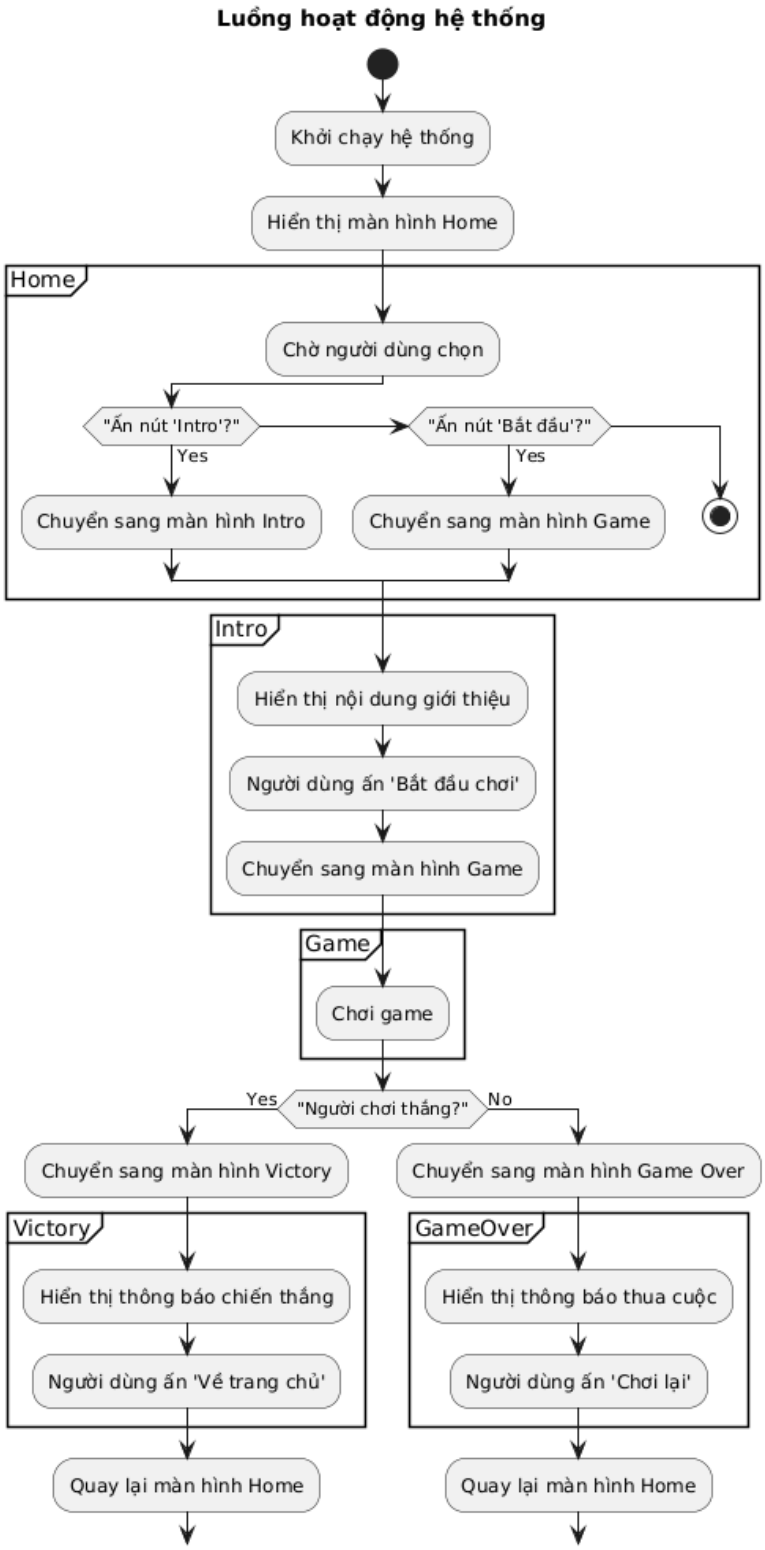


Figure 6: Biểu đồ luồng hoạt động của game

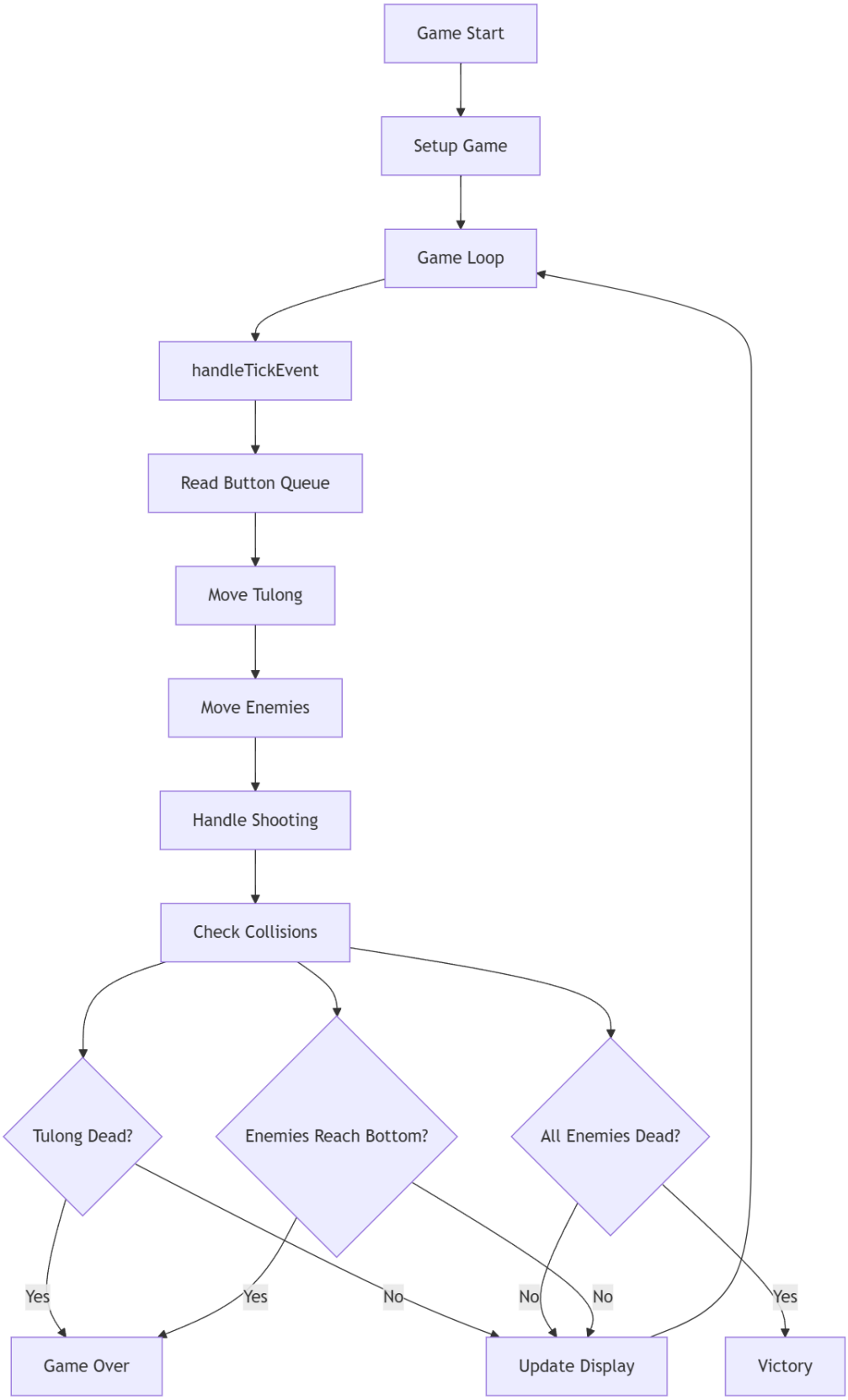


Figure 7: Biểu đồ luồng hoạt động của logic game

3 Cài đặt & Xây dựng hệ thống

3.1 Github

Repository của project:**NowhereToRun**

3.2 Mô tả các module phần mềm chính

3.2.1 Cấu hình các task

```
const osThreadAttr_t defaultTask_attributes = {
  .name = "defaultTask",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* Definitions for GUI_Task */
osThreadId_t GUI_TaskHandle;
const osThreadAttr_t GUI_Task_attributes = {
  .name = "GUI_Task",
  .stack_size = 8192 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
```

Figure 8: Cấu hình các task

Cấu hình task `defaultTask()` để xử lý nhận dữ liệu điều khiển từ nút bấm, cấu hình task `GUI_Task()` để xử lý các task GUI, logic game.

3.2.2 Nhận dữ liệu từ tác nhân người chơi

```
  /*Configure GPIO pins : PG2 PG3 */
  GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
```

Figure 9: Cấu hình GPIO nhận tín hiệu điều khiển từ button

```
/* Definitions for myQueue1 */
osMessageQueueId_t myQueue1Handle;
const osMessageQueueAttr_t myQueue1_attributes = {
  .name = "myQueue1"
};
```

Figure 10: Cấu hình Queue1 nhận và truyền dữ liệu nút bấm sang logic game

```c
void StartDefaultTask(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
      for (;;) {
        if (HAL_GPIO_ReadPin(GPIOG, GPIO_PIN_2) == GPIO_PIN_RESET) {
          uint8_t cmd = 1; // 1 = TRÁI
          HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_13);
          osMessageQueuePut(myQueue1Handle, &cmd, 0, 0);
        }

        if (HAL_GPIO_ReadPin(GPIOG, GPIO_PIN_3) == GPIO_PIN_RESET) {
          uint8_t cmd = 2; // 2 = PHẢI
          HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_13);
          osMessageQueuePut(myQueue1Handle, &cmd, 0, 0);
        }

        osDelay(100);
      }
  /* USER CODE END 5 */
}
```

Figure 11: DefaultTask nhận dữ liệu từ nút bấm rồi truyền vào Queue1

Liên tục polling trạng thái nút bấm, nếu có tín hiệu, sẽ truyền dữ liệu điều khiển tương ứng vào `Queue1` để hàm logic game trong `Screen1View.cpp` xử lý điều khiển nhân vật di chuyển.

```
void Screen1View::handleTulongMovement()
{
    uint8_t cmd;

    osStatus_t status = osMessageQueueGet(myQueue1Handle, &cmd, NULL, 0);

    if (status == osOK)
    {
        if (!tulongAlive) return; // Don't move if tulong is dead

        int currentX = tulong.getX();
        int newX = currentX;

        switch(cmd)
        {
            case 1: // LEFT button pressed
                newX = currentX - 10;
                // Check left boundary
                if (newX < TULONG_MIN_X)
                {
                    newX = TULONG_MIN_X;
                }
                break;

            case 2: // RIGHT button pressed
                newX = currentX + 10;
                // Check right boundary
                if (newX > TULONG_MAX_X)
                {
                    newX = TULONG_MAX_X;
                }
                break;

            default:
                return;
        }

        // Update tulong position
        if (newX != currentX)
        {
            tulong.setX(newX);
            tulongX = newX; // Update stored position
            invalidate();
        }
    }
    else if (status == osErrorTimeout)
    {
        return;
    }
    else
    {
        return;
    }
}
```

Figure 12: Hàm nhận tín hiệu từ Queue1 và xử lý di chuyển nhân vật

3.2.3 Khởi tạo khi vào trò chơi

```
Screen1View::Screen1View()
: moveRight(true), zigzagStep(2), dropStep(10), tickCount(0), speedMultiplier(1),
isDimming(false), dimTickCount(0),
starActive(false), starX(0), starY(0), starSpeed(STAR_SPEED), shootCooldown(0),
stoneActive(false), stoneX(0), stoneY(0), stoneSpeed(STONE_SPEED), tulongShootCooldown(0),
tulongHealth(MAX_HEALTH), tulongAlive(true), tulongX(0)
{
    tickCount = 0;
    isDimming = false;
    dimTickCount = 0;

    // Khởi tạo hệ thống bắn của enemies
    starActive = false;
    starX = 0;
    starY = 0;
    shootCooldown = 0;

    // Khởi tạo hệ thống bắn của tulong
    stoneActive = false;
    stoneX = 0;
    stoneY = 0;
    tulongShootCooldown = 0;

    // Khởi tạo tulong movement
    tulongX = 0;

    for (int i = 0; i < 3; i++) {
        strawHealth[i] = 6;
        strawAlive[i] = true;
    }
    tulongHealth = MAX_HEALTH;
    tulongAlive = true;

    // Khởi tạo Hardware RNG
    initHardwareRNG();
}
```

Figure 13: Construction khởi tạo

`Screen1View::Screen1View()` Khởi tạo tất cả biến và trạng thái ban đầu của game.

- Thiết lập hướng di chuyển ban đầu của enemies (moveRight = true)
- Khởi tạo tốc độ di chuyển và bắn đạn
- Thiết lập hệ thống sức khỏe cho các nhân vật
- Khởi tạo hardware random number generator.

```
void Screen1View::setupScreen()
{
    Screen1ViewBase::setupScreen();

    //Scoring
    currentScore = 0;

    // Gán buffer trước, rồi snprintf
    score.setWildcard(scoreBuffer);
    Unicode::snprintf(scoreBuffer, 10, "%d", currentScore);
    score.invalidate();

    textArea1.setWildcard(highScoreBuffer); // <- highscore
    Unicode::snprintf(highScoreBuffer, 10, "%d", highScore);
    textArea1.invalidate();

    currentLevel = 1;
    enemyShootCount = 1; // level 1 chỉ bắn 1 viên
    speedMultiplier = 1;

    // Hiển thị text level
    Unicode::snprintf(levelBuffer, 10, "%d", currentLevel);
    level.setWildcard(levelBuffer);
    level.invalidate();


    // Gán enemy xen kẽ sb và c7
    enemies[0] = &sb1; enemies[1] = &c71; enemies[2] = &sb2;
    enemies[3] = &c72; enemies[4] = &sb3; enemies[5] = &c73;
    enemies[6] = &sb4; enemies[7] = &c74; enemies[8] = &sb5;
    enemies[9] = &c75; enemies[10] = &sb6; enemies[11] = &c76;
    enemies[12] = &sb7; enemies[13] = &c77; enemies[14] = &sb8;
    enemies[15] = &c78; enemies[16] = &sb9; enemies[17] = &c79;

    // Vị trí bắt đầu và khoảng cách
    int startX = 20;
    int startY = 20;
    int spacingX = 35;
    int spacingY = 40;

    // Khởi tạo vị trí ban đầu và trạng thái sống cho từng enemy
    for (int row = 0; row < NUM_ROWS; ++row)
    {
        for (int col = 0; col < NUM_COLS; ++col)
        {
            int idx = row * NUM_COLS + col;
            enemyX[idx] = startX + col * spacingX;
            enemyY[idx] = startY + row * spacingY;
            enemies[idx]->setXY(enemyX[idx], enemyY[idx]);
            enemies[idx]->setVisible(true);
            enemyAlive[idx] = true;
        }
    }
```

Figure 14: Setup màn hình chính của game

`setupScreen()` thiết lập giao diện và vị trí ban đầu của tất cả đối tượng.

- Thiết lập điểm số ban đầu (score = 0)
- Sắp xếp enemies thành hàng ngang và dọc
- Đặt vị trí ban đầu cho tulong (nhân vật chính)
- Hiển thị các barrier (straw1, straw2, straw3)
- Ẩn các viên đạn ban đầu

3.2.4 Các hàm thực hiện logic game

1. **Screen1View::handleTickEvent()**: Vòng lập chính của game, xử lý movement, shooting, collision detection và game state updates mỗi 10 ticks.
2. **Screen1View::handleTulongMovement()**: Xử lý di chuyển nhân vật tulong dựa trên message queue từ RTOS, kiểm tra boundaries và cập nhật vị trí.
3. **Screen1View::handleEnemyShooting()**: Quản lý hệ thống bắn của enemies, tạo đạn star từ một enemies ngẫu nhiên còn sống với cooldown và tốc độ theo level.
4. **Screen1View::handleTulongShooting()**: Quản lý hệ thống bắn của tulong, tạo đạn stone hướng về enemy gần nhất với cooldown riêng.
5. **Screen1View::updateStarMovement()**: Cập nhật chuyển động đạn star (enemies) di chuyển xuống dưới và kiểm tra ra khỏi màn hình.
6. **Screen1View::updateStoneMovement()**: Cập nhật chuyển động đạn stone (tulong) di chuyển lên trên và kiểm tra ra khỏi màn hình.
7. **Screen1View::checkCollision()**: Kiểm tra va chạm giữa hai collision box sử dụng thuật toán AABB (Axis-Aligned Bounding Box).
8. **Screen1View::getImageCollisionBox()**: Tạo collision box từ touchgfx::Image object với tọa độ và kích thước.
9. **Screen1View::handleCollisions()**: Xử lý va chạm đạn star với barriers (straw) và tulong, gây damage và ẩn đạn khi trúng.
10. **Screen1View::handleStoneCollisions()**: Xử lý va chạm đạn stone với enemies và barriers, phá hủy enemy hoặc gây damage cho straw.
11. **Screen1View::damageStraw()**: Giảm health của barrier straw, ẩn straw khi health = 0 và cập nhật health display.
12. **Screen1View::damageTulong()**: Giảm health của tulong, trigger game over khi health = 0 và ẩn các đạn đang bay.
13. **Screen1View::destroyEnemy()**: Phá hủy enemy (1 hit kill), tăng score và kiểm tra victory condition khi tất cả enemies chết.
14. **Screen1View::updateHealthDisplay()**: Cập nhật hiển thị thanh máu của tulong dựa trên current health value.
15. **Screen1View::getRandomAliveEnemy()**: Chọn ngẫu nhiên một enemy còn sống từ danh sách enemies để thực hiện bắn.
16. **Screen1View::getClosestEnemy()**: Tìm enemy gần nhất với tulong (theo trục X) để tulong tăng khả năng nhắm bắn chính xác.
17. **Screen1View::initHardwareRNG()**: Khởi tạo Hardware Random Number Generator của STM32 để tạo số ngẫu nhiên thật.

18. **Screen1View::getHardwareRandom()**: Tạo số ngẫu nhiên sử dụng hardware RNG, fallback về tick-based random nếu lỗi.
19. **Screen1View::getTulongX()**: Getter function trả về tọa độ X hiện tại của tulong.
20. **Screen1View::isTulongAtLeftBoundary()**: Kiểm tra xem tulong có đang ở boundary trái của màn hình không.
21. **Screen1View::isTulongAtRightBoundary()**: Kiểm tra xem tulong có đang ở boundary phải của màn hình không.
22. **Screen1View::addScore()**: Cộng điểm, cập nhật high score, tính toán level progression và tăng difficulty theo level.
23. **Screen1View::triggerGameOver()**: Kết thúc game, ẩn tất cả đạn và chuyển sang GameOver screen.
24. **Screen1View::triggerVictory()**: Kết thúc game với victory, ẩn tất cả đạn và chuyển sang Victory screen.
25. **Screen1View::buzzerOn()**: Bật buzzer bằng cách set GPIO PA5 lên HIGH để phát âm thanh.
26. **Screen1View::buzzerOff()**: Tắt buzzer bằng cách reset GPIO PA5 xuống LOW để ngừng phát âm thanh.
27. **Screen1View::playSoundEffect()**: Phát hiệu ứng âm thanh bằng cách bật buzzer trong 100ms rồi tắt (beep sound).

3.3 Đóng góp của từng thành viên

Nhiệm vụ	Người thực hiện
Tìm hiểu và lên ý tưởng trò chơi	Nguyễn Lan Nhi Bùi Hà My
Xử lý tín hiệu nút bấm	Nguyễn Lan Nhi
Tìm hiểu thu thập hình ảnh đồ họa	Nguyễn Lan Nhi Bùi Hà My
Thiết kế đồ họa bằng TouchGFX	Bùi Hà My
Phát triển phần mềm điều khiển	Nguyễn Lan Nhi
Xử lý logic và hiển thị trò chơi	Nguyễn Lan Nhi Bùi Hà My
Phát âm thanh	Nguyễn Lan Nhi Bùi Hà My
Viết báo cáo	Nguyễn Lan Nhi Bùi Hà My

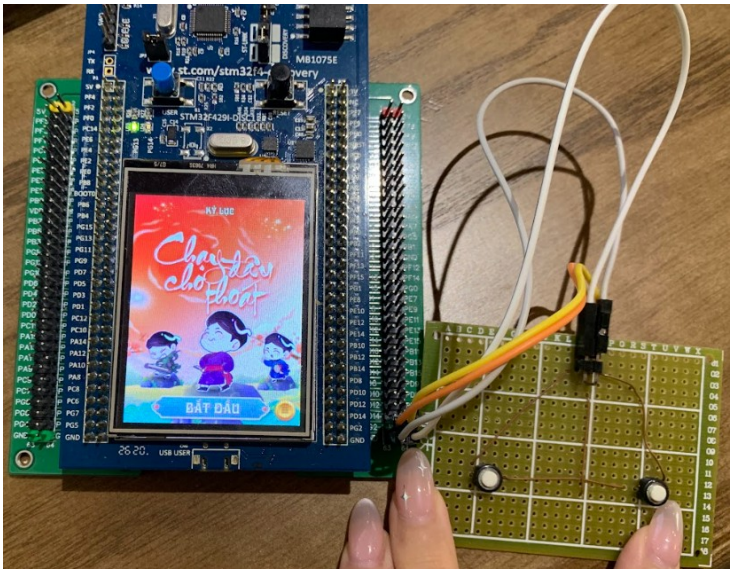
Table 4: Phân công công việc giữa các thành viên

3.4 Kết quả

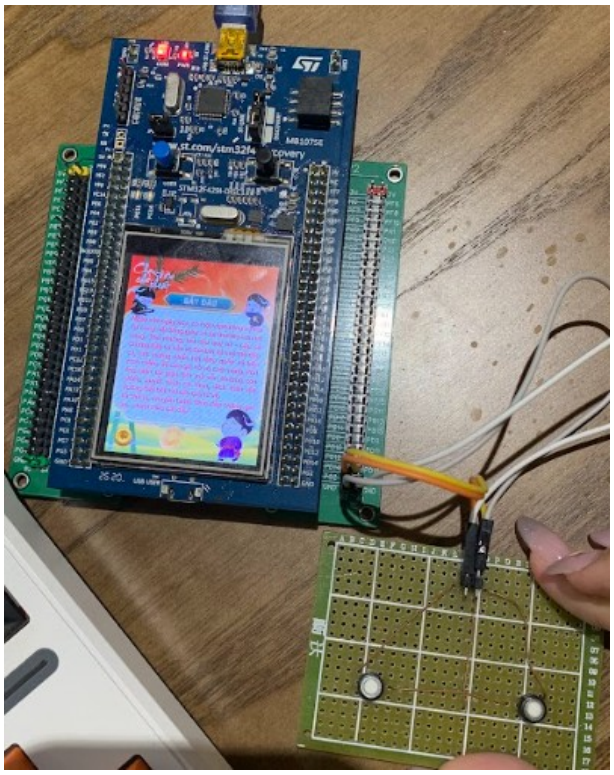
3.4.1 Video demo

https://youtube.com/shorts/v_ZoxEzecjQ

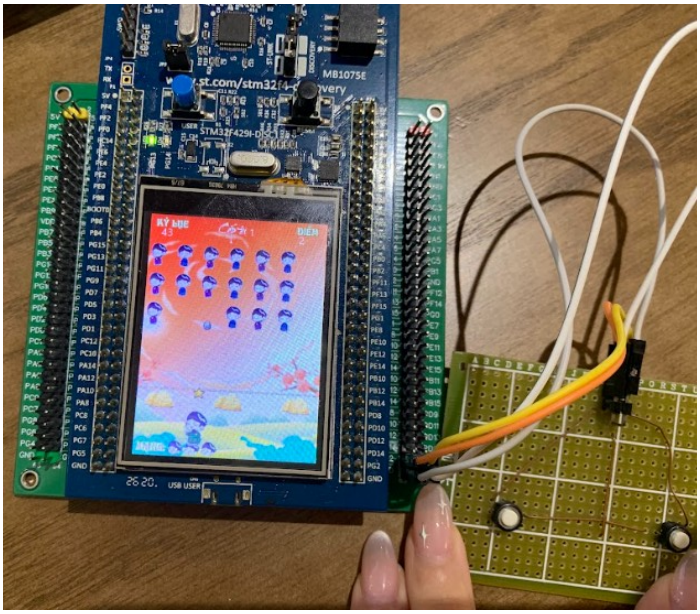
3.4.2 Giao diện khởi động



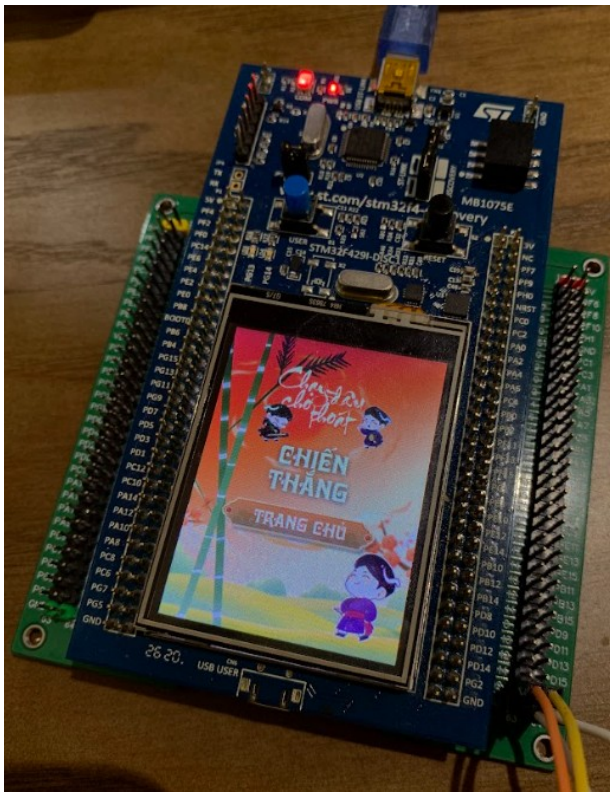
3.4.3 Giao diện giới thiệu



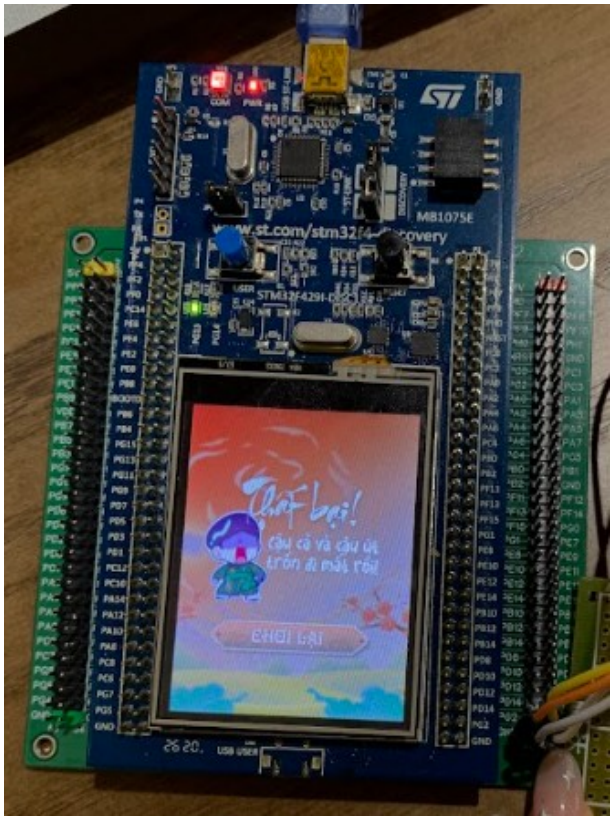
3.4.4 Giao diện khi chơi



3.4.5 Giao diện chiến thắng



3.4.6 Giao diện thất bại



3.5 Đánh giá

Hệ thống trò chơi mô phỏng từ Space Invader với tựa đề *"Chạy đâu cho thoát"* đã được triển khai thành công trên nền tảng phần cứng STM32F429I-DISC1. Tất cả các yêu cầu chức năng và phi chức năng đặt ra trong giai đoạn thiết kế đều đã được hiện thực hóa đầy đủ.

Trong quá trình vận hành thực tế, hệ thống cho thấy tính ổn định cao, không ghi nhận hiện tượng treo, trễ phản hồi hoặc lỗi logic. Việc phát âm thanh và hiển thị hình ảnh được duy trì liên tục ở tốc độ khung hình cao, trong khi các thao tác điều khiển có độ trễ rất thấp, đảm bảo tính tương tác thời gian thực giữa người dùng và hệ thống.

Bên cạnh các yêu cầu kỹ thuật, hệ thống còn được đánh giá cao về trải nghiệm người dùng. Giao diện đồ họa được thiết kế trực quan, mới lạ và có tính thẩm mỹ cao. Đặc biệt, trò chơi xây dựng một cốt truyện rõ ràng, lấy bối cảnh quen thuộc của làng quê vùng đồng bằng Bắc Bộ Việt Nam, từ đó góp phần tăng tính hấp dẫn và mức độ nhập vai cho người chơi.

Tổng thể, hệ thống đáp ứng đầy đủ các tiêu chí về chức năng, hiệu năng, độ ổn định và tính mở rộng, đồng thời cung cấp một trải nghiệm người dùng hoàn chỉnh, phù hợp với mục tiêu của đồ án môn học hệ thống nhúng.