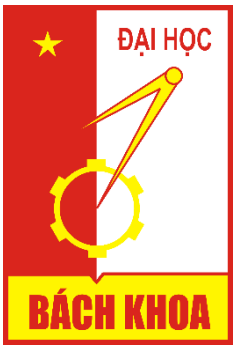
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION COMMUNICATION AND TECHNOLOGY



EMBEDDED SYSTEMS REPORT

Dinosaur Game using TouchGFX library on STM32F429

Lecturer: **PhD Ngo Lam Trung**

Name	**StudentID**
Trần Như Thái	20194835
Lê Triệu Sáng	20194829
Trần Chí Thành	20194845

Hanoi, 06/2025

TABLE OF CONTENTS

INTRODUCTION

In this project, we recreated the Dinosaur Game on STM32F429 using the TouchGFX library. The project is developed incrementally, from scaffolding graphics using the TouchGFX designer, to the implementation of game functions and the connection to external devices. As a consequence of the project, we have gained practical knowledge and skills in developing an embedded application with an ARM CPU.

I. Project Overview

1. Objective

The objective of the project is recreating the Dinosaur Game to run on the STM32F4 hardware.

2. Content

The project is divided into 3 main parts :

- Scaffolding graphics and generate source code : To design and create screens, we use the TouchGFX Designer software to place contents (sprites, texts) and define the screen transitions. Furthermore, we generate the source code for the STM32 application from the TouchGFX project.
- Developing the game logic : This game contains main functionalities:
    + The movement of the objects (dinosaur, enemies)
    + The enemies instantiation
    + Collision detection
    + Gameplay (Scoring, difficulty, game ending condition)

All of these functionalities are developed and tested incrementally by connecting to the STM32.

- External hardware connection: In addition to the User button B1 which is responsible for the screen transition, we utilize an external hardware component, which is a joystick. This joystick is responsible for controlling the movement of the Dinosaur.

3. Workload

Work	**Contributor**
Design UI	Trần Như Thái
Movement of the dinosaur	Lê Triệu Sáng
Collision detection	Trần Như Thái
Enemies instantiation	Trần Chí Thành
Scoring	Trần Chí Thành
External hardware connection	Lê Triệu Sáng

II. Project Design

1. Scaffolding game UI

After obtaining images for all the objects in the game, we began to put them together into the TouchGFX designer software to build the user interface for the game.

The application contains three screens :

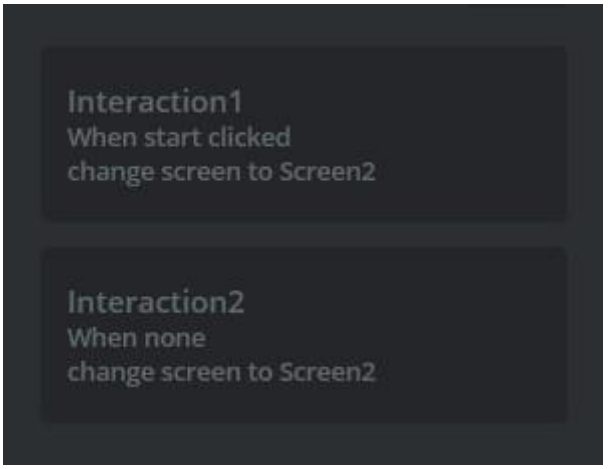
- Screen 1 is the start screen which only contains the start button for the user to click on to begin playing the game.

- Screen 2 is the gameplay screen which is composed of all the gameplay objects. When the user score surpasses the best score, the user score will disappear and the best score will now track user's.

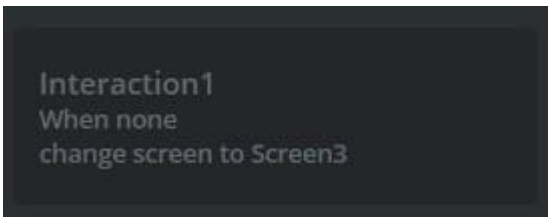
- Screen 3 is the end screen, which will display the gameover interface, the user score and a button for them to restart the game.

Besides putting together images and screens, we also define 3 screen transitional interactions :

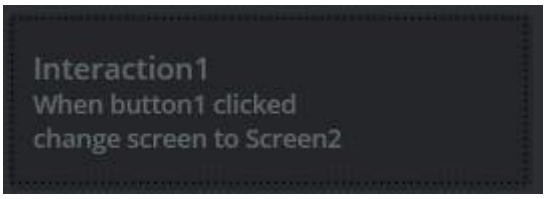
- The first interaction happens when the user click on the start button on the screen in the first screen or press the user button which will take them to the gameplay screen :



- The second interaction is a placeholder transition, which will be used in code to handle the game ending logic to transition into the game over screen :



- The last interaction is used to restart the game by transitioning into the gameplay screen from the gameover screen :



After defining all the screens, images, and transitions, we generate the application code for the project.

2. Game Logic

1) Movement

a) Dinosaur

- The Dinosaur is the main character of the game. In this project, we used two widgets to implement two different states of the Dinosaur: *dino* for the normal upright posture and *dinoc* for the crouched posture. Each of these widgets is an Animated Image Widget created using the TouchGFX Designer. Each widget contains two images, creating the appearance of movement for the character.
- At the beginning, the character is placed near the left side of the screen. The *dino* widget is used to represent the Dinosaur by default, with its visibility set to *True*, allowing it to run normally. The *dinoc* widget, on the other hand, has its visibility set to *False* while still being positioned at the same location as the *dino* widget. The character remains in a fixed horizontal position throughout the game, only changing in the vertical direction while jumping.
- There are two actions available: *jumping* and *crouching*, which are controlled via a joystick.

- *Jumping*: In this game, the Dinosaur jumps by moving the dino widget vertically upwards then downwards. We use a quadratic function to control the height of the dino in every tick during its jump.

  We use function $y = a * t^2 + v_0 * t + y_0$ for calculate the next position at next tick with the parameter below:

  a is acceleration

  v_0 is base speed when dino jump.

  y_0 is the current position

  t is time unit (t = tick/10) control the jump speed of dino

  When the user moves the joystick upward, the bool variable *isJumping* is set to true and the *dinoJump()* function is called. This function performs in every tick until the Dinosaur's height is smaller than *BASE_DINO_Y*. At that time, the Dinosaur landed, then we reset *isJumping* to false.

- *Crouching*: When the user moves the joystick downward in the y-direction until the value received from the peripheral is under 64, the Dinosaur widget switches to `dinoc` while staying in the same position.

b) Enemies

- In this game, enemies are obstacles that the player has to dodge. If the player collides with any enemy, the game stops. There are two types of obstacles: *cactus* and *bird*.
  - *Cactus*: This is an *Image* widget that displays a static image.
  - *Bird*: This is an *Animated Image* widget that supports flapping animation using two images, similar to the mechanism of the Dinosaur.
- Only one enemy spawns at a time, entering the screen from a random horizontal position on the right side and becoming visible as it moves into view. When the game starts, the enemy gradually moves from the right side to the left side of the screen at a random speed. For the bird obstacle, its vertical position is also randomized before entering the screen.

- All random operations are performed using a manual randomization function based on the system tick as shown below.

```
uint32_t Screen2View::randint(void) {
    static uint32_t seed = 0;

    if (seed == 0) {
        seed = SysTick->VAL;
    }
    seed = (seed * 1103515245 + 12345) & 0x7FFFFFFF;
    seed ^= SysTick->VAL;

    return seed;
}
```

- This function generates a pseudo-random number using a combination of linear congruential generator (LCG) and bitwise XOR with the current value of the system tick counter (`SysTick->VAL`). The steps are as follows:
    - **Initialization**: The seed is initially set to 0. If it remains 0, it is assigned the current value of the system tick counter.
    - **LCG Step**: The seed is updated using the LCG formula. This formula generates a new pseudo-random number based on the previous seed value.
    - **Bitwise XOR**: The updated seed is then XORed with the current value of the system tick counter to introduce additional randomness.
    - **Return Value**: The resulting seed value is returned as the pseudo-random number.

2) Collision detection

When the Dinosaur collides with an enemy, the game is over. Therefore, we need to handle the collision between the Dinosaur and the enemies. We treat the body of the objects as a box and apply the bounding box collision detection technique. The implementation of this collision detection is as follow:

- Each object participating in the collision detection will have its top, bottom, left and right value calculated.

- In the case of Dinosaur, we check whether the Dinosaur is in the normal form or the crouched form and calculate the bounding box values based on its state.

- In the case of the enemy, we need to check the current present enemy to calculate its bounding box values.

- We then perform a condition expression to check the combined conditions : Dinosaur’s right value >= enemy’s left value, Dinosaur’s left value <= enemy’s right value, Dinosaur’s bottom value >= enemy’s top value and Dinosaur’s top value <= enemy’s bottom value. If the condition expression evaluates to true, the collision has occurred, else there is no collision happening.

- Since the objects in this game are not actually a box, we need to compensate for it by making the Dinosaur’s box smaller than it actually is.

```
bool Screen2View::checkCollision(){
    int16_t dinoT, dinoR, dinoB, dinoL;
    if (dino.isVisible()){
        int deviation = BASE_DINO_Y - dino.getY(); // amount of deviation of the dino from the ground
        if (deviation > 8) deviation = 8; // max deviation is 8, which reaches dino's legs
        dinoL = dino.getX() + 10;
        dinoT = dino.getY() + 10;
        // simulate the skew of the dino's lower body using the deviation value
        dinoR = dino.getX() + dino.getWidth() - 10 - deviation;
        dinoB = dino.getY() + dino.getHeight() - 10;
    } else {
        dinoL = dinoc.getX();
        dinoT = dinoc.getY() + 10;
        dinoR = dinoc.getX() + dinoc.getWidth();
        dinoB = dinoc.getY() + dinoc.getHeight();
    }

    int16_t obsT, obsR, obsB, obsL;
    if (currentEnemy == 0){
        obsL = cacti1.getX();
        obsT = cacti1.getY();
        obsR = cacti1.getX() + cacti1.getWidth();
        obsB = cacti1.getY() + cacti1.getHeight();
    } else {
        obsL = bird.getX();
        obsT = bird.getY();
        obsR = bird.getX() + bird.getWidth();
        obsB = bird.getY() + bird.getHeight();
    }

    if (dinoR >= obsL && dinoL <= obsR && dinoB >= obsT && dinoT <= obsB){
        return true;
    }

    return false;
}
```

The collision detection procedure is included in the main loop. If the game detects a collision, it will navigate to the gameover screen, signifying game ending. This is achieved through the call to the predefined screen transition method generated by TouchGFX when we define a placeholder screen transition :

```
static_cast<FrontendApplication*>(Application::getInstance())->gotoScreen3ScreenSlideTransitionEast();
```

3) Scoring

We have two variables for storing scores : *score* and *highestScore*, which is stored in the model class. In the ScreenView, we use *counter* and *highestScore* to keep track of the user score.

For each five ticks, we increase *counter* by 1. When the game over occurs, we check whether the user score is *higher* than *highestScore*, if yes, assign value to model's *highestScore*.

Beside displaying the score in the gameplay screen, we also need to display it in the gameover screen. We read the high score from the ScreenView's *highestScore.*

3. External hardware connection

In this project, two main hardware components are used: the User button B1 and a joystick.

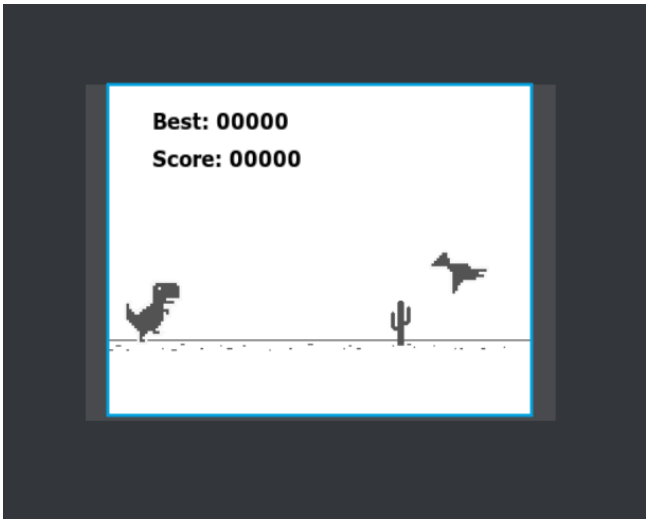
- The User button is the blue button attached to the board. It serves two purposes: it starts the game when the user first powers on the board and displays the initial screen, and it restarts the game when it is over, allowing the user to replay. This button's function is managed using a *buttonPressQueue* queue. In the *hardware_polling_task*, a value (by default, *1*) is pushed to the queue to inform the system that the button has been pressed, triggering the appropriate response.
- The joystick is an analog component used to control the Dinosaur character. It utilizes the *ADC1* channel in mode *IN13* and the *ADC2* channel in mode *IN5*, with each channel set to an 8-bit resolution. The detailed wiring connection for the joystick is shown below:

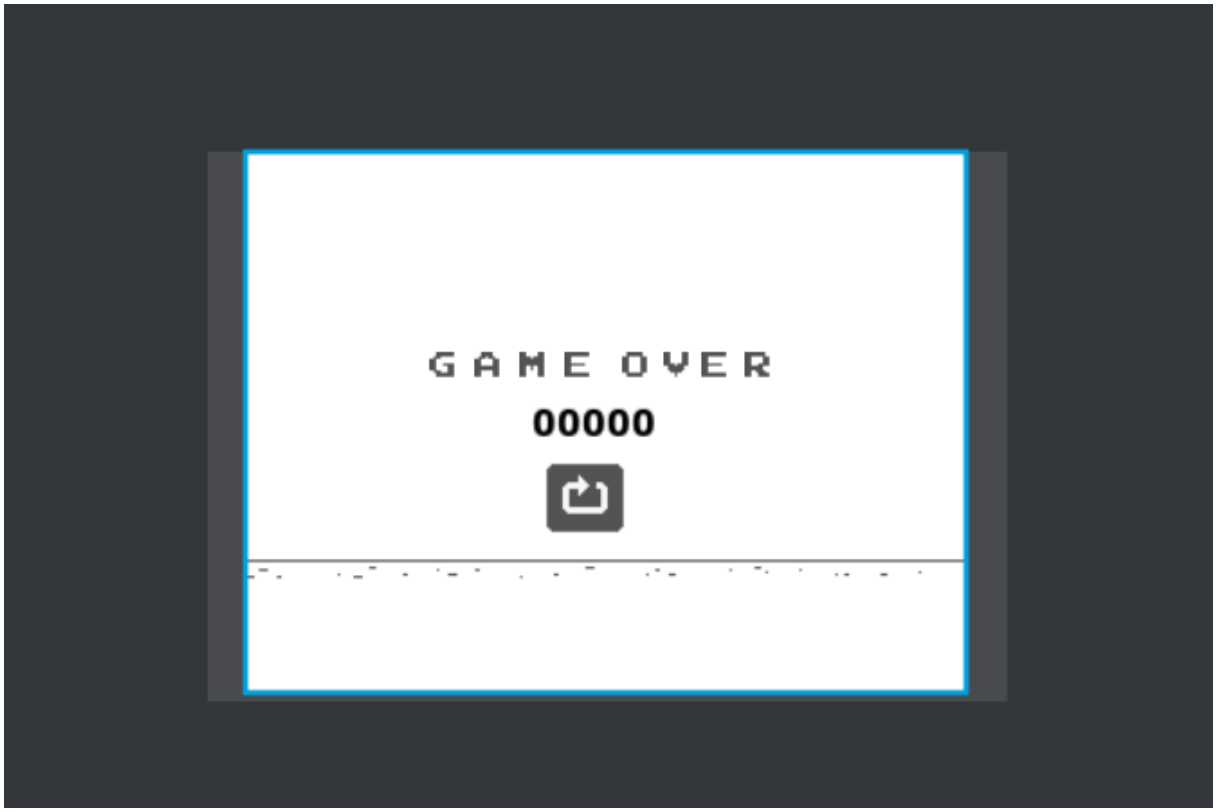
Joystick	Vcc	GND	X	Y
STM32F429	3V	GND	PC3	PA5

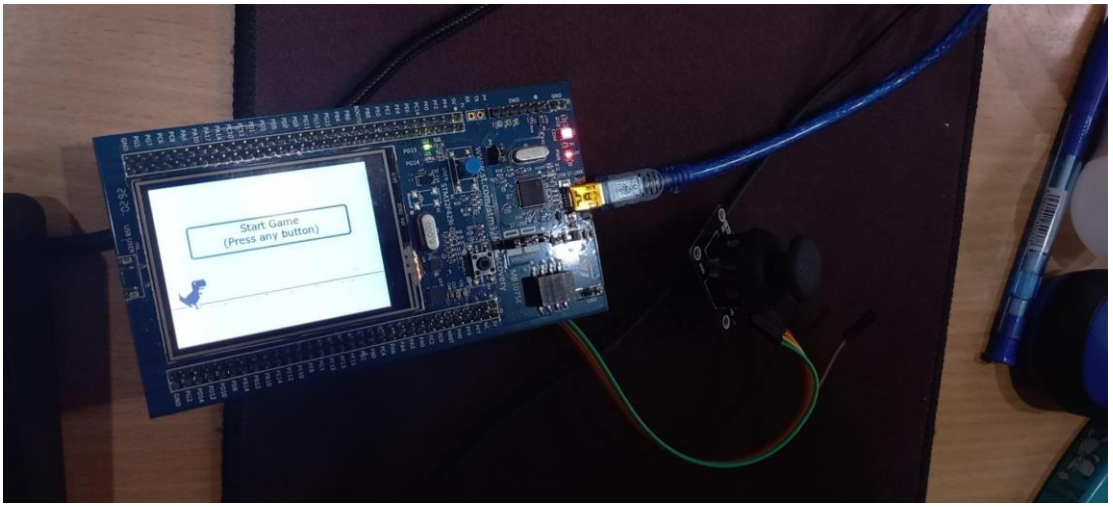
- To interface with the joystick, the *hardware_polling_task* is used along with a *joyStickQueue* that holds 16 items of type *uint32_t*. When the joystick is used, each piece of information is pushed into the queue as a single number. This number is a concatenation of two lower-precision numbers representing the X and Y coordinates sent from the joystick.

III. Conclusion

Here is the results of our project:







Project link github: *https://github.com/trannhuthai/Group5-DinosaurGame*

In conclusion, the project to remake Dinosaur Game on the STM32F429I microcontroller has been simply accomplished, resulting in a functional and entertaining application. Through this project, we have demonstrated the versatility of the STM32F429I microcontroller by implementing a simple yet engaging game using its resources.

Throughout the project, we faced various challenges, such as figuring out what TouchGFX can do, how to solve problems with hardware, software. However, these challenges allowed us to gain valuable experience in embedded systems development and problem-solving.

As we conclude this project, we reflect on the knowledge gained, the challenges overcome, and the satisfaction of creating a tangible piece of software within the embedded systems realm. The project serves as a testament to the possibilities that arise when blending software development skills with hardware innovation.