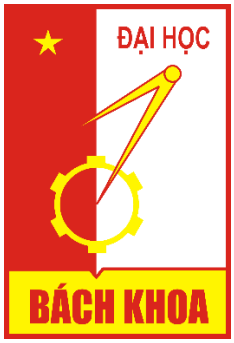
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION COMMUNICATION AND TECHNOLOGY

----- 🕮 -----



EMBEDDED SYSTEMS REPORT

Topic: **Chrome Dino Game using TouchGFX library on STM32F429**

Lecturer: **PhD Ngo Lam Trung**

Name	**StudentID**
Cao Dang Dat	20205177
Nguyen Tuan Dung	20205148

Ha Noi, 08/2023

Table of Contents

INTRODUCTION

The topic for our capstone project is remaking the popular chrome dino game on STM32F429 using the TouchGFX library. The project is developed incrementally, from scaffolding graphics through the TouchGFX designer, to gradual additions of game functionality and connection to external devices.

Working through the project has equipped us with practical knowledge and skills in developing an embedded application with an ARM CPU and will be the motivation for developing other embedded systems in the future.

I. Project Overview

1. Purpose

The purpose of the project is to remake the chrome dino game to run on the STM32F4 hardware.

2. Content

The project can be divided into 3 main parts :

- Scaffolding graphics and generate source code : We use the TouchGFX Designer software to create screens, place image and sprites on each screen and define basic transitions between screens. After that, we generate the source code for the STM32 application from the TouchGFX project.
- Developing game logic : After generating the project, we continue to develop logic and functionality for the dino game. From movement of the sprites to handling collision and define logic for scoring, difficulty and game ending condition. Each new functionality is developed incrementally and tested by connecting to the STM32.
- Connect to external hardware : In the game logic development section, we use existing hardware of the STM32 to control the movement of the sprite, which is limited and not ideal. So we introduce external hardware that connects to the STM32 to control the sprite.

3. Workload

Work	**Contributor**
Design UI	Dat, Dung
Movement of object	Dung, Dat
Check collision	Dung
Spawn enemy	Dung
Scoring	Dat
External hardware connection	Dat

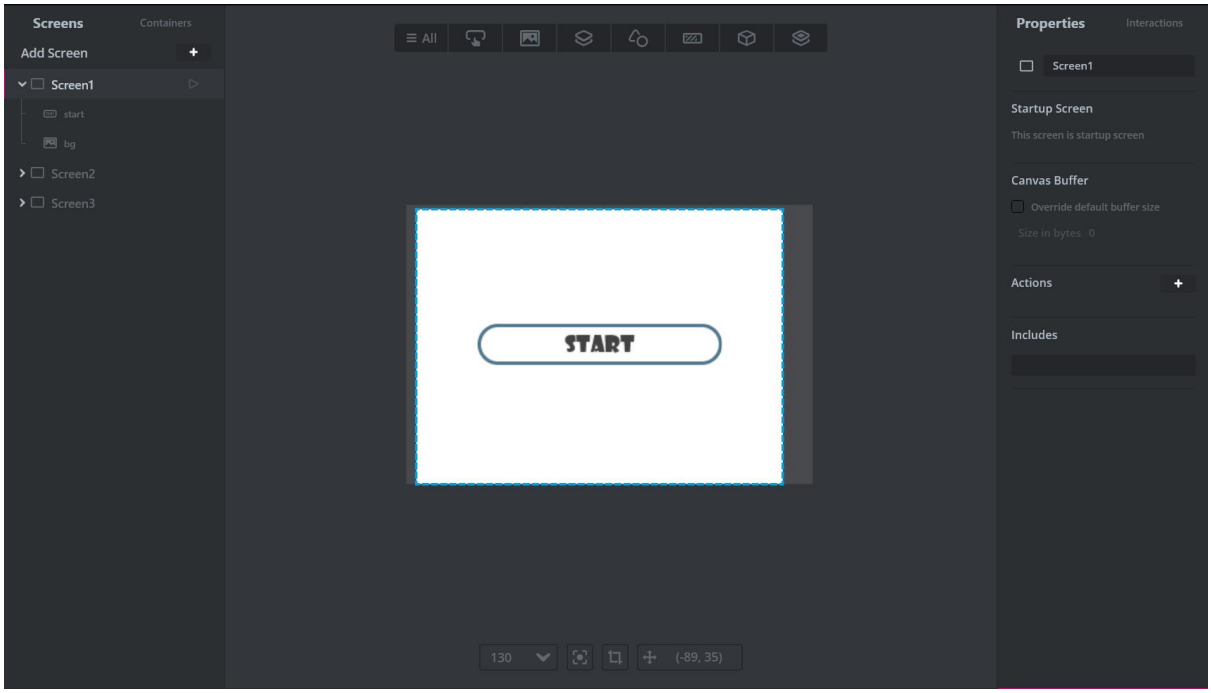
II. Project Design

1. Scaffolding game UI

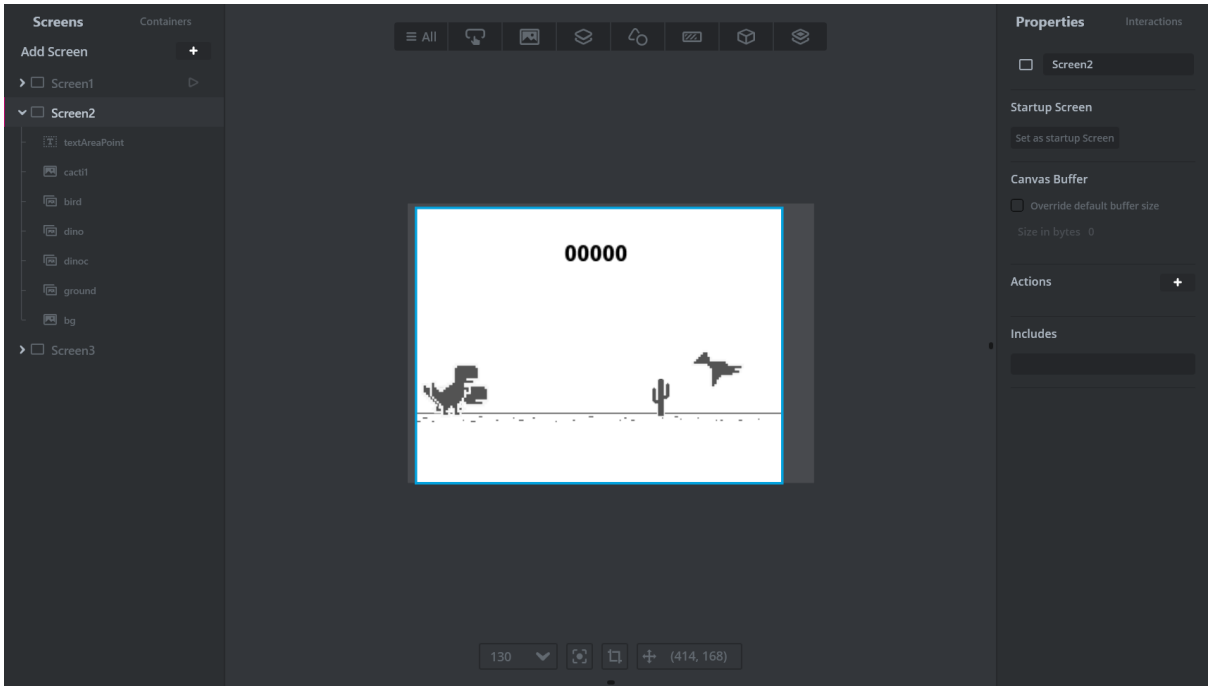
After obtaining sprite images for all the objects in the game, we began to put them together into the TouchGFX designer software to build the user interface for the game.

The application contains three screens :

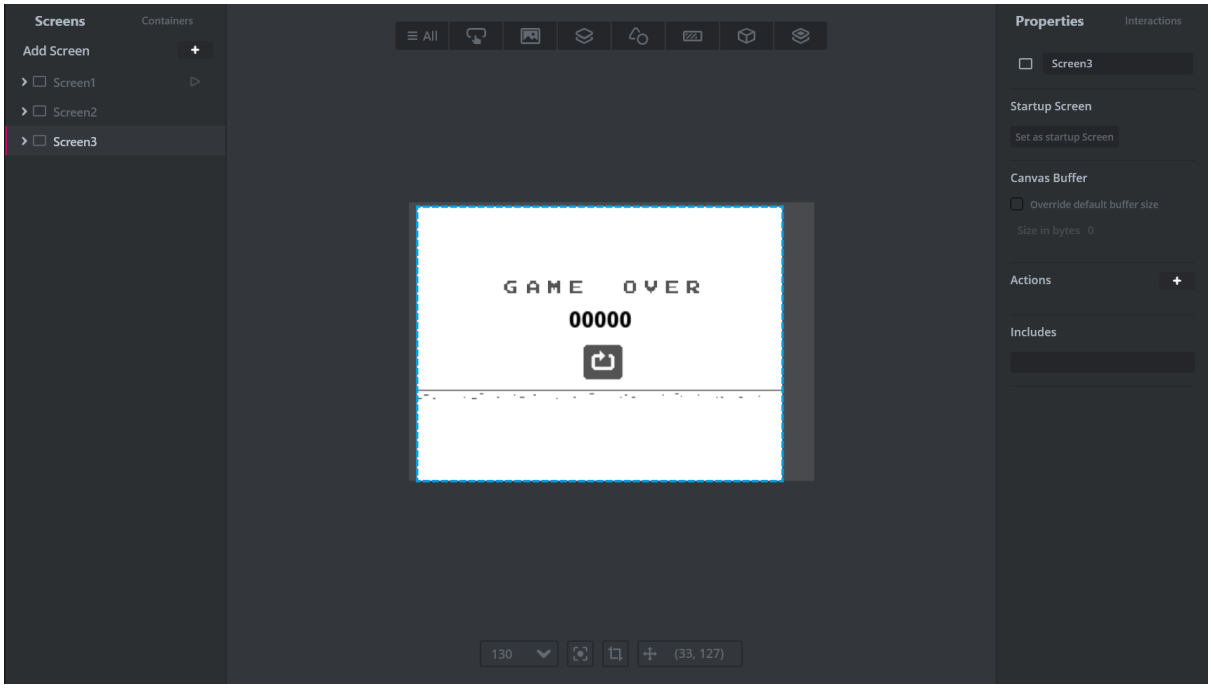
- Screen 1 is the start screen which only contains the start button for the user to click on to begin playing the game.



- Screen 2 is the gameplay screen which is comprised of all the gameplay objects.

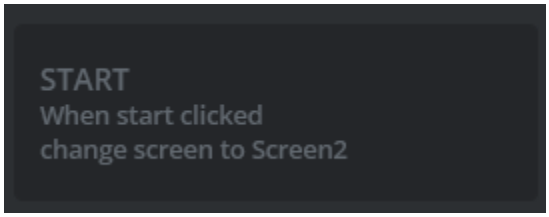


- Screen 3 is the end screen, which will display the gameover interface, the user score and a button for them to restart the game.

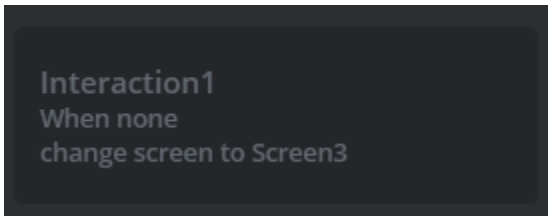


Besides putting together sprite images and screens, we also define 3 screen transitional interactions :

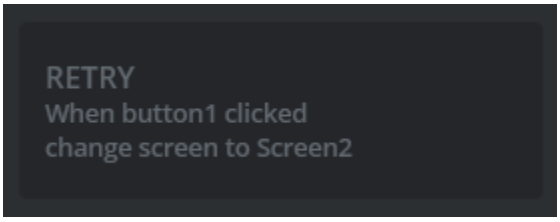
- The first interaction happens when the user click on the start button in the first screen which will take them to the gameplay screen :



- 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. Dino

The dino is the main character that the user will be able to control. It only has two basic movements : jumping and crouching.

- Jumping : In this game, the dino jumps by moving the dino image vertically upwards then downwards. We use quadratic function to control the height of the dino for each 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

We check the position wether it equal of greater than BASE_HEIGHT. If no, the dino landed, then we reset the isJump state to false.

- Crouching : The crouching movement is handled a bit differently. Basically, the Dino has two sprite image, one is the normal standing posture and the other one is the crouched posture. Normally, the crouched image is made invisble. When user press crouch, the crouched image will become visible and the standing image will be made invisible, thus, make it seems like the dino has made a crouching animation. When the crouch button is released, the visibility of the two images will be reverted back.

b. Enemies

There are two types of enemy in this game, one is the cactus and the other is the bird. Their movement is handled as follow :

- A variable currentObs is used to determine the next type of enemy that will appear on the screen.

- Both types of enemy will start at the right edge of the screen.

- If the value of currentObs is 0, the next enemy is the cactus, otherwise it is the bird.

- A variable horizontal is used to keep track of the X position of the present enemy on the screen, with its initial value being equal to the width of the screen.

- For each passed tick, the present enemy's location is updated by decrement the horizontal value by a enemySpeed value and set it to the enemy's X position.

- After the enemy has gone out of the screen, a new enemy is chosen by setting the currentObs value to rand() % 2.

2) Collision detection

To determine the game ending condition, we need to handle the collision between the dino and the enemies.

To make the collision detection simple, we treat the body of the objects as a box and apply the bounding box collision detection technique. The logic is as follow :

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

- In the case of dino, we need to check whether the dino 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 : dino's right value >= enemy's left value, dino's left value <= enemy's right value, dino's bottom value >= enemy's top value and dino's top value <= enemy's bottom value.

- If the condition expression evaluates to true, then collision has occurred, else there is no collision happening.

```
bool Screen2View::checkCollision(){
    int16_t dinoT, dinoR, dinoB, dinoL;
    if (dino.isVisible()){
        dinoL = dino.getX();
        dinoT = dino.getY() + 10;
        dinoR = dino.getX() + dino.getWidth() - 10;
        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 (currentObs == 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;
}
```

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

The collision detection procedure is included in the main loop. If the game detects collision, it will transitions 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())->gotoScreen3ScreenNoTransition();
```

3) Scoring

We have two variables for recording scores : *score* and *highestScore*.

For each five tick, we increase score by 1. When gameover occur, we check wether score *higher* than *highestScore*, if yes, assign value for *highestScore*.

Beside displaying the score in the gameplay screen, we also need to display it in the gameover screen. We write highest score into flash memory at 0x081E0000, and read at the location using HAL library.

4) Difficulty

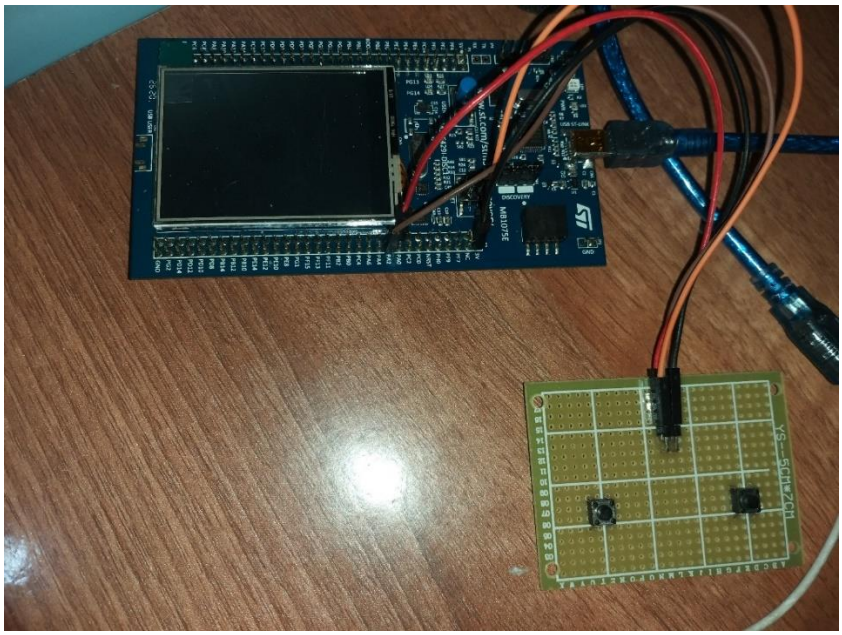
There are two difficulty modifications that are currently added to the game : enemy speed increase and varying of bird height.

- Enemy speed increase : A variable obsNumber is used to keep track of the number of enemy the dino has gone passed. For every 10 enemies, we increase the speed of the enemies – which basically is the number of pixels they move per tick, by 1.

- Varying of bird height : To make the game more fun and close to the original, we set the bird to have varying height. There are 3 height options of bird : ground level – the dino have to jump over it, base height level – the dino have to crouch under it and the sky level – the dino does not have to jump or crouch to go over. Determination of the bird height is also achieved through the use of our custom random function rand().

3. External hardware connection

We set PA1, PA2 at input mode connect with 2 4-pin buttons calling Btn1, Btn2 (as image below).



Then we create new task *hardwarePolling* for handle external button. In this task, we read the state of Btn1, Btn2 by using HAL library. If button is pressed, the state number (1, 2, 3) will be put in the *buttonQueue* for movement handling :

1 : jumping

2 : crouching

3 : reset state after crouch

We use two flags to ensure that only one state sent in 1 click (even when user hold the button). The code below is for polling task.

```
void startPolling(void *argument)
{
  /* USER CODE BEGIN startPolling */
  /* Infinite loop */
  int flag1 = 0, flag2 = 0;
  for(;;)
  {
        GPIO_PinState Btn1State = HAL_GPIO_ReadPin(BTN_1_GPIO_Port, BTN_1_Pin);
        GPIO_PinState Btn2State = HAL_GPIO_ReadPin(BTN_2_GPIO_Port, BTN_2_Pin);

        if(Btn1State == GPIO_PIN_SET){
              HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin, GPIO_PIN_SET);
              if(flag1 == 0){
                    int buffer = 1;
                    HAL_Delay(10);
                    osMessageQueuePut(buttonQueueHandle, &buffer, 0, 10);
              }
              flag1 = 1;
        }else{
```

```
                HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin, GPIO_PIN_RESET);
                flag1 = 0;
        }
        if(Btn2State == GPIO_PIN_SET){
                HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_SET);
                if(flag2 == 0){
                        int buffer = 2;
                        HAL_Delay(10);
                        osMessageQueuePut(buttonQueueHandle, &buffer, 0, 10);
                }
                flag2 = 1;
        }else{
                     int buffer = 3;
                     osMessageQueuePut(buttonQueueHandle, &buffer, 0, 10);
                HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_RESET);
                flag2 = 0;
        }
        osDelay(1);
  }
  /* USER CODE END startPolling */
}
```

III. Conclusion

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 figure out what TouchGFX can do, how to solve problem 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.