Flappy Bird

WEEK 1

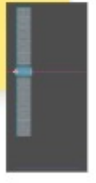
First, I created the "start.tscn" as the game's starting page.
During this stage, I learned how to use Godot's signal mechanism (pressed.connect()) to implement the button click event.
I also set up input detection:

```
if Input.is_action_just_pressed("ui_accept"):
    _on_RestartButton()
```

This enabled me to understand the function of Input Map, which allows the same action to be bound to multiple keys (such as the space bar and buttons).
Reflection and Analysis:
At first, I wasn't clear about the execution timing of _process(delta) and _ready(). This led to the repeated triggering of the button logic. After debugging, I gradually understood the difference between the frame loop and the initialization function.

WEEK 2

In play.gd, I wrote the core game loop logic.
The key parts include:
Control the movement of the obstacle: barrier.position.x -= 80 * delta
Determine the boundary and generate a new obstacle: spawn_barrier()
Score system: update_score() → score += 1
By constantly adjusting the movement speed and refresh interval, I made the game pace more natural.
Learning gain: I understood the meaning of delta - it enables the game to maintain consistency across different frame rates.
Encountered difficulties:
The positions of the obstacles are fixed and remain the same every time the game is run.
Solution: Add the "randomize()" function in the "_ready()" section and use "randi_range(200, 800)" to generate random spacing.
This experience made me realize the significance of randomness in game experience design.



WEEK 3

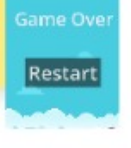
The control logic for the bird is the most challenging part of the entire project.
I achieved the falling and rising movements by setting the gravity parameter.
position.y += g
g += .3
if Input.is_action_pressed("ui_space"):
g = -9

This process enabled me to learn the basic concept of acceleration control and also allowed me to appreciate the significance of "feel adjustment".
Debugging results:
When the value of g is too high, the bird falls too fast and the player is almost unable to control it.
When the g value is too small and the game pace is too slow.
Through experiments with different values, I gradually found the balance.
I also added the texture switching logic, which makes the bird have different appearances when it ascends and descends, enhancing the visual feedback effect of the game.

WEEK 4

The implementation of the end page is relatively simple and is triggered by a button:
get_tree().change_scene_to_file("res://play.tscn")

Initially, I attempted to detect death in bird.gd and directly switch scenes, but this led to logical confusion.
Later, I moved the death detection to play.gd for management, and the problem was solved.
This made me understand the importance of modular programming - the main scene should uniformly manage the game state.



WEEK 5

During the classroom, the teacher pointed out that after players entered the game, they were unsure of which keys to press to start the game, and there was no prompt on how to restart the game after it ended.
This made me realize that although the functional logic had been completed, there were still significant issues with the user experience (UX).
I then carried out the repair:
In the "start" scene, the following screen prompt text has been added: "Press the space bar or click the button to start."
In the "lost" scenario, add the following prompt: "Press the R key or click the button to restart."
And in the code, an "Input.is_action_just_pressed("ui_accept")" detection was added to ensure that the keyboard operation is valid.
Reflection:
This revision experience made me realize for the first time that "programming is not just about making the functions work, but also about teaching users how to use them".
I began to think about problems from the perspective of a "designer" rather than a developer. This is a higher level of growth in code literacy.