Assignment1—Report

Li Yukun 558620

Introduction:

Flappy Bird is a simple yet challenging 2D casual game. The goal of the game is to control a bird through a series of randomly generated upper and lower pipes, and the player taps the screen to make the bird jump, avoiding the pipe and the screen boundary. The bird is affected by gravity and will fall naturally, and if it hits a pipe or goes beyond the screen (top or bottom), the game ends. For each pair of pipes passed, the score is increased by 1. In our implementation, the game window size is 800x600 pixels, the initial position of the bird is 1/4 of the left side of the screen, the pipe moves from right to left at a fixed speed (4 pixels/frame), and the gap height is 150 pixels. With the Pygame library, we drew a blue bird (10 pixels radius) and a green pipe.

Game Design:

1. Summary of the reward function  
   Positive Rewards:  
   Near the center of the gap (< 30 pixels): 50 - distance. Through the pipe: +50.

Negative Rewards:

Collision: -50. Away from the center of the gap (> 75 pixels): - (distance - 75)² / 100.  
No change: The distance is between 30-75 pixels

1. Learn how strategies can maximize cumulative rewards  
   Core Strategy: Bird learns through Q-Sheet:  
   As the pipe approaches, jump at the right time to maintain distance\_to\_gap\_center < 30 based on the current y-position, velocity, and relative distance from the center of the gap.  
   Try to stabilize your position after passing through the pipes to prepare for the next pair of pipes.  
   Reward-driven: Dynamic positive rewards near gaps (max 50/frame) and fixed rewards through pipes (+50) push the bird to continuously optimize its path, while collisions (-50) and distance away (squared penalties) reduce invalid behavior.  
   Result: After multiple episodes, total\_reward grew as the bird was able to maintain a positive reward in more frames and pass through more pipes.
2. The synergistic effect of ε-greedy, learning rate and discount factor  
   ε-greedy: early exploration to discover high-reward areas (such as gap centers), and later use to ensure strategy execution.  
   Learning rate: Smooth Q value update, so that the strategy gradually converges to the optimal path of maximizing total\_reward.  
   Discount Factor: Balances the small positive rewards per frame with the big rewards that pass through the pipe, extending the birdie's survival time and increasing the cumulative return.
3. Statespace composition

Bird's Y Position (bird\_y)：

Definition: The vertical position of the bird (bird.y) in the range of [0, HEIGHT] (0 to 600 pixels).

Discretization: Divide the y position by 60 and map to [0, 9] with an interval of every 60 pixels.

For example: y = 0 → 0, y = 120 → 2, y = 599 → 9.

Gap Relative Distance (gap\_relative)：

Definition: The distance difference between the bird's y position and the center (gap\_center) of the next pair of pipe gaps.

Calculate: gap\_center = next\_pipe.gap\_y + next\_pipe.gap // 2, then bird.y - gap\_center.

Discretization: distance divided by 30, range [-5, 4], plus 5 mapped to [0, 9].

Velocity (velocity)：

Definition: The vertical velocity of a bird (bird.velocity), affected by gravity (1.0) and jumping (-6).

Discretization: Range [-10, 10], plus 10 and divided by 2, maps to [0, 9].

The size of the state space:

Total statuses: STATE\_SIZE = 1000.

Calculation: 10(bird\_y) × 10(gap\_relative) × 10(velocity) = 1000.

Index: bird\_y \* 100 + gap\_relative \* 10 + velocity, which maps the 3D state to a 1D index.

(5)Composition of action space

Action 0: Do Nothing (Down)：

Effect: Without jumping, the bird falls naturally due to gravity (gravity = 1.0).

Implementation: bird.update() increases the speed and updates the position.

Action 1: Jump (Up)：

Effect: Perform a jump that sets the speed to jump\_velocity = -6 to move the bird upwards.

Implementation: bird.jump() sets the speed, then bird.update() updates the position.

Characteristics of the action space

Size: ACTION\_SIZE = 2, only two actions, suitable for the simple control mechanism of the Flappy Bird.

Dynamics: Actions are combined with states, and Q-sheet learning is used to choose "jump" or "no jump" in different situations to maximize rewards.

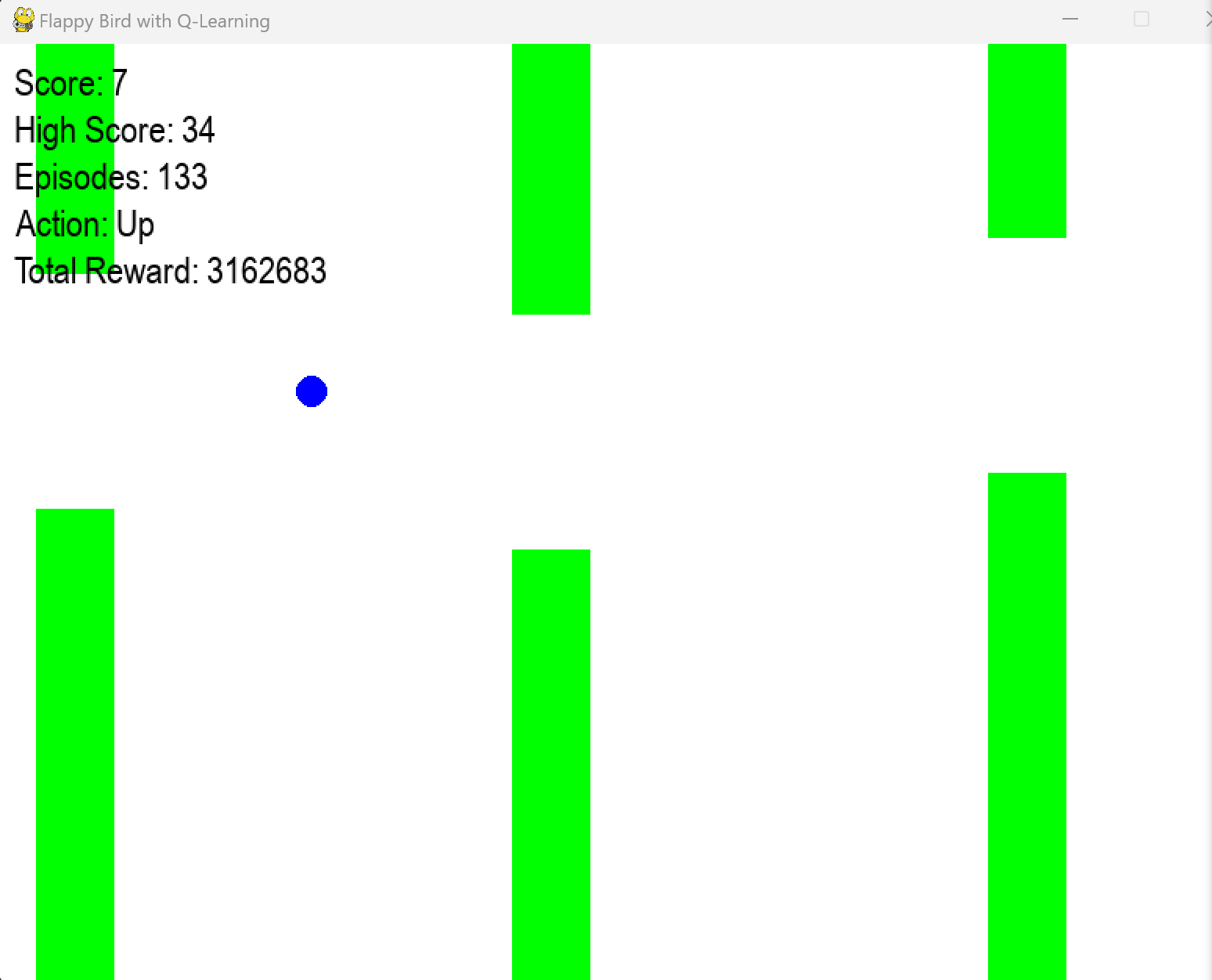
The role of the action space

Strategy formation: The bird chooses an action based on the Q table, e.g. jumping when gap\_relative < 5 (below) and not jumping when gap\_relative > 5 (above).

UI:

Blue: Bird(player)

Green: Tube(barrier)



Challenge:

1. Adjusting parameters: The settings of the initial exploration rate (Epsilon) and decay rate (Epsilon decay) were difficult to balance, and the bird either cured the strategy too early or underexplored, and it took several adjustments to determine the combination of 0.3 and 0.98.
2. Adjust the reward function: In the early days, the reward function was too simple (e.g., only through the pipe, and the bird could not learn to align the gaps, so it was modified several times to add a positive reward (50 - distance) near the center and a squared penalty to be far away, so as to effectively guide the strategy.
3. Adjust the learning rate: The learning rate of 0.1 fluctuates greatly at the beginning, and after trying 0.5, it is overfitted, and finally called back to 0.1 to stabilize the Q value update.
4. Movement optimization: Birds often jump or fall for no reason, and the ineffective actions are reduced by refining the state space (adding speed) and movement stability (adjusting gravity and jump strength).

Process of Adjusting parameters:  
Debugging and parameter tuning are the core aspects of the implementation process, and the following are the main adjustment stages:  
Initial (0-70 times):  
Problem: Birdie scores no more than 5 and either falls or jumps.  
Adjust:  
Status expanded from 100 to 1000, adding a speed dimension.  
The learning rate increased from 0.1 to 0.5, accelerating convergence.  
Epsilon decay changed from 0.999 to 0.95 to reduce invalid exploration.  
Result: The score increased to 8, but it was still unstable.  
Medium (70-190 times):  
Problem: Occasionally, the bird flies up wildly, or passes through a pipe only to misjudge death.  
Adjust:  
Fix the custom\_pipe error in the collide method to bottom\_pipe.  
The learning rate is reduced to 0.2 to avoid Q fluctuations.  
The reward function is optimized to reduce the passing reward (1000 → 50) and increase the reward near the center (10 → 100).  
Epsilon decay is adjusted to 0.98 for extended exploration.  
Results: At Episode 190, a High Score of 27 and an Avg Pass Distance of 23.94 were the best performers.

Post-optimization (after 190 cycles):

The ball is getting better and better, playing more and more like a normal person, and the number of passes has increased greatly

Evaluation and Summary

The intelligent control of Flappy Bird through Q-Learning gave me a deep understanding of the practical challenges of reinforcement learning. The state space needs to fully reflect the dynamics of the environment, the reward function must accurately guide the target behavior, and parameter adjustment is the art of balancing efficiency and stability. The process of tuning parameters halfway through was a lot of trial and error: from state to reward optimization, to Epsilon and learning rate fine-tuning, each step brought me closer to my goal.

After countless training sessions, my Flappy Bird Q-Learning model performed well, achieving a maximum score of 104 when submitting this evaluation report, far exceeding the previous one, showing strong optimization capabilities. Reflects the birdie effectively accumulating rewards for alignment and passage through the pipe. The average passing distance is 23.10 pixels, precisely aligned to the center of the gap. The Q table and initial Q value indicate that the strategy is stable, and the exploration rate of 0.05 indicates that the training is mature. The UI verifies the effectiveness of dynamic adjustments, and the state and action space design support efficient learning and successfully maximize the cumulative reward.